
Fractals

Simple

Game

Graphics

Library

SGG Library

Επιμέλεια:

Α.Δρακόπουλος

Πίνακας περιεχομένων

Fractals....7
1. Mandelbrot Set....7
2. Julia Set....7
3. Sierpinski Triangle....7
4. Sierpinski Carpet....7
5. Koch Snowflake....7
6. Pythagoras Tree....8
7. Barnsley Fern....8
8. Dragon Curve....8
9. Cantor Set....8
10. Apollonian Gasket....8
11. Peano Curve....8
12. Hilbert Curve....8
13. T-Square Fractal....9
14. H-Fractal....9
15. Vicsek Fractal....9
16. Menger Sponge....9
17. Hexaflake....9
18. Gosper Curve (Gosper Island)....9
19. Lévy C Curve....9
20. Pentaflake....9
Sierpinski Triangle....11
Κώδικας....11
Επεξήγηση Κώδικα....13
Sierpinsky Carpet....15
Κώδικας....15
Αναλυτική Επεξήγηση Κώδικα....17
Koch Snowflake....18
Κώδικας....18
Επεξήγηση Κώδικα....20
Barnsley Fern....22
Κώδικας....23
Επεξήγηση Κώδικα....24
Dragon Curve....26
Κώδικας....26
Αναλυτική Επεξήγηση Κώδικα....27
Cantor Set....29
Κώδικας....29
Επεξήγηση Κώδικα....30
Apollonian Gasket....32
Κώδικας....32
Επεξήγηση Κώδικα....34
Peano Curve....35
Κώδικας....35
Αναλυτική Επεξήγηση Κώδικα....37
Hilbert Curve....38
Κώδικας....38
Αναλυτική Επεξήγηση Κώδικα....40
T-Square Fractals....42
Κώδικας....42

Επεξήγηση Κώδικα..... 44
H-Fractal..... 45
Κώδικας..... 45
Επεξήγηση Κώδικα..... 47
Vicsek Fractals..... 49
Κώδικας..... 49
Αναλυτική Επεξήγηση Κώδικα..... 51
Menger Sponge..... 52
Κώδικας για το Menger Sponge (2D Προβολή)..... 52
Αναλυτική Επεξήγηση Κώδικα..... 54
Hexaflake..... 56
Κώδικας..... 56
Επεξήγηση Κώδικα..... 58
Επεξήγηση της Συνάρτησης drawPolygon..... 59
Gosper Curve..... 60
Κώδικας..... 60
Επεξήγηση Κώδικα..... 62
Levy C Curve..... 64
Κώδικας..... 64
Επεξήγηση Κώδικα..... 66
Κώδικας..... 67
Επεξήγηση της Συνάρτησης drawPolygon..... 69
Golden Dragon Fractals..... 70
Οδηγίες:..... 70
Κώδικας..... 70
Επεξήγηση Κώδικα..... 72
Butterfly Fractals..... 73
Κώδικας..... 73
Επεξήγηση Κώδικα..... 75
Plasma Fractals..... 76
Κώδικας..... 76
Επεξήγηση Κώδικα..... 78
Octahedron Fractal..... 79
Κώδικας..... 79
Επεξήγηση Κώδικα..... 81
Snowflake Curve..... 82
Κώδικας..... 82
Επεξήγηση Κώδικα..... 85
L System fractal..... 86
Κώδικας..... 86
Επεξήγηση Κώδικα..... 89
Vortex fractal..... 90
Κώδικας..... 90
Επεξήγηση Κώδικα..... 92
Durer Pentagon Fractal..... 93
Κώδικας..... 93
Περιγραφή Κώδικα..... 95
Barnsley Fern Fractal..... 97
Κώδικας..... 97
Περιγραφή Προγράμματος..... 99
Sierpinski Hexagon Fractal..... 100
Κώδικας..... 100

Περιγραφή Προγράμματος..... 102
Crystal Growth Fractal..... 103
Κώδικας..... 103
Περιγραφή Κώδικα..... 105
Fractal Hands..... 107
Κώδικας..... 107
Περιγραφή Κώδικα..... 109
Fractal Mountains..... 110
Κώδικας..... 110
Περιγραφή Κώδικα..... 112
Σημειώσεις..... 112
Pentagon Fractal..... 113
Κώδικας..... 113
Επεξήγηση Κώδικα..... 115
Σημειώσεις..... 115
Thue-Morse Curve..... 116
Κώδικας..... 116
Επεξήγηση Κώδικα..... 118
Προσαρμογές..... 118
Jellyfish Fractal..... 119
Κώδικας..... 119
Επεξήγηση του Κώδικα..... 121
Σημειώσεις..... 121
Gosper Island Fractal..... 122
Περιγραφή του Gosper Island Fractal..... 122
Κώδικας..... 123
Επεξήγηση Κώδικα..... 125
Σχόλια..... 125
Pentadendrite Fractal..... 126
Περιγραφή του Pentadendrite Fractal..... 126
Κώδικας για τη Σχεδίαση του Pentadendrite Fractal..... 126
Κώδικας..... 126
Επεξήγηση Κώδικα..... 129
Σχόλια..... 129
Pinwheel Tiling Fractal..... 130
Κώδικας..... 130
Επεξήγηση του Κώδικα..... 132
Σχόλια..... 132
Vicsek Cross Fractal..... 133
Κώδικας..... 133
Επεξήγηση του Κώδικα..... 135
Σχόλια..... 135
Hexagon Tiling Fractal..... 136
Κώδικας..... 136
Επεξήγηση Κώδικα:..... 138
Square Fractal..... 139
Κώδικας..... 139
Επεξήγηση Κώδικα:..... 140
Carpet Fractal..... 142
Κώδικας..... 142
εξήγηση Κώδικα..... 144
Circle Packing Fractal..... 145

Κώδικας.....145
Επεξήγηση Κώδικα.....147
H-Tree Fractal.....148
Κώδικας.....148
Επεξήγηση Κώδικα.....150
Gasket of Triangles Fractal.....151
Κώδικας.....151
Επεξήγηση Κώδικα.....153
Zeno’s Paradox Fractal.....154
Κώδικας.....154
Επεξήγηση Κώδικα.....155
Carpet Tiling Fractal.....156
Κώδικας.....156
Επεξήγηση Κώδικα.....158
Ice Fractal.....159
Κώδικας.....159
Επεξήγηση Κώδικα.....161
Spiral of Archimedes.....162
Κώδικας.....162
Επεξήγηση Κώδικα.....164
Cubic Fractal.....165
Κώδικας.....165
Επεξήγηση Κώδικα.....167
Torus knot fractal.....168
Κώδικας.....168
Περιγραφή και Επεξήγηση Κώδικα.....170
Επεξήγηση της Λειτουργίας του Torus Knot.....170
Knot theory fractal.....171
Κώδικας.....171
Περιγραφή και Λειτουργία Κώδικα.....173
Επεξήγηση.....173
Butterfly Effect Fractal.....174
Κώδικας.....174
Περιγραφή Κώδικα:.....176
Lorenz Attractor Fractal.....177
Κώδικας.....177
Επεξήγηση Κώδικα.....179
Περιγραφή.....179
Ring Fractal.....180
Κώδικας.....180
Επεξήγηση Κώδικα.....182
Περιγραφή.....182
Pythagorean Tree Fractal.....183
Κώδικας.....183
Περιγραφή.....185
Flower Fractal.....186
Κώδικας.....186
Περιγραφή του Κώδικα.....188
Crystal Fractal.....189
Κώδικας.....189
Περιγραφή του Κώδικα.....191
Fibonacci Spiral Fractal.....192

Κώδικας.....192
Περιγραφή του Κώδικα.....194
Triangular Spiral Fractal.....195
Κώδικας.....195
Περιγραφή του Κώδικα.....197
Binary Tree Fractal.....199
Κώδικας.....199
Περιγραφή του Κώδικα.....201
Fractal.h header file and main function example.....202
Header file: fractals.h.....202
Δείγμα main που καλεί fractals.....350

Fractals

Τα fractals είναι γεωμετρικές μορφές που χαρακτηρίζονται από επανάληψη μοτίβων σε διαφορετικές κλίμακες. Ακολουθούν κάποια fractals και η υλοποίησή τους χρησιμοποιώντας την SGG Library.

1. Mandelbrot Set

- **Περιγραφή**: Δημιουργήθηκε από τον Benoît Mandelbrot και είναι το πιο γνωστό fractal. Χρησιμοποιεί τη συνάρτηση z=z2+cz = z^2 + cz=z2+c, όπου z και c είναι μιγαδικοί αριθμοί. Το σύνολο αυτό παράγει απίστευτα περίπλοκες εικόνες με συνεχείς αναλογίες σε κάθε ζουμ.
- **Χρησιμότητα**: Βρίσκει εφαρμογές στη δημιουργία τέχνης και οπτικοποιήσεων σε μαθηματικά και επιστήμες.

2. Julia Set

- **Περιγραφή**: Παρόμοιο με το Mandelbrot set, το Julia set παράγεται από παρόμοιες συναρτήσεις, αλλά το αποτέλεσμα εξαρτάται από την αρχική τιμή του c. Διαθέτει διαφορετικά σχήματα και στυλ ανάλογα με την παράμετρο.
- **Χρησιμότητα**: Χρησιμοποιείται στην τέχνη και τα γραφικά για τη δημιουργία εξαιρετικών οπτικών αποτελεσμάτων.

3. Sierpinski Triangle

- **Περιγραφή**: Δημιουργείται από τη διαίρεση ενός τριγώνου σε τέσσερα μικρότερα τρίγωνα και την αφαίρεση του κεντρικού. Η διαδικασία επαναλαμβάνεται, οδηγώντας σε ένα αυτοόμοιο σχήμα.
- **Χρησιμότητα**: Βρίσκει εφαρμογές στα γραφικά και την εκπαίδευση, εξηγώντας βασικές αρχές της γεωμετρίας και των fractals.

4. Sierpinski Carpet

- **Περιγραφή**: Επέκταση του Sierpinski triangle σε δύο διαστάσεις, σχηματίζοντας ένα μοτίβο τύπου "χαλιού" μέσω αφαίρεσης κεντρικών τετραγώνων σε κάθε τμήμα.
- **Χρησιμότητα**: Χρησιμοποιείται σε μαθηματικές μελέτες και απεικονίσεις fractal γεωμετρίας.

5. Koch Snowflake

- **Περιγραφή**: Ξεκινά από ένα ισόπλευρο τρίγωνο, κάθε πλευρά χωρίζεται και αναπληρώνεται με ένα μικρότερο τρίγωνο. Το αποτέλεσμα θυμίζει χιονονιφάδα.
- **Χρησιμότητα**: Χρησιμοποιείται σε μαθήματα μαθηματικών για την εισαγωγή στις έννοιες της απείρου και της γεωμετρικής αναδρομής.

6. Pythagoras Tree

- **Περιγραφή**: Ξεκινά με ένα τετράγωνο, στο οποίο τοποθετούνται μικρότερα τετράγωνα και συνεχίζει με τη δημιουργία νέων τετραγώνων σε διαδοχικές επαναλήψεις.
- **Χρησιμότητα**: Χρησιμοποιείται για μοντελοποίηση δομών όπως τα δέντρα και άλλες φυσικές γεωμετρίες.

7. Barnsley Fern

- **Περιγραφή**: Χρησιμοποιεί μαθηματικούς κανόνες για να αναπαραστήσει το σχήμα ενός φύλλου φτέρης. Είναι ένα από τα πιο γνωστά fractals στη βιολογία.
- **Χρησιμότητα**: Χρησιμοποιείται στην ανάλυση βιολογικών δομών όπως τα φύλλα και τα δέντρα.

8. Dragon Curve

- **Περιγραφή**: Δημιουργείται από διαδοχικές καμπύλες σε προκαθορισμένες γωνίες, παράγοντας ένα μοτίβο που θυμίζει δράκο.
- **Χρησιμότητα**: Χρησιμοποιείται σε τέχνη και σχεδιασμό για τη δημιουργία πολύπλοκων καμπυλών.

9. Cantor Set

- **Περιγραφή**: Πολύ απλό fractal που παράγεται αφαιρώντας το μεσαίο τμήμα μιας γραμμής και επαναλαμβάνοντας τη διαδικασία.
- **Χρησιμότητα**: Χρησιμοποιείται για να εξηγηθεί η έννοια της διάστασης και του απείρου.

10. Apollonian Gasket

- **Περιγραφή**: Αποτελείται από κύκλους που τοποθετούνται σε ένα μοτίβο λαβυρίνθου, με κάθε επόμενο κύκλο να είναι μικρότερος από τον προηγούμενο.
- **Χρησιμότητα**: Χρησιμοποιείται σε μαθηματικές μελέτες και στη θεωρία της γεωμετρίας των κύκλων.

11. Peano Curve

- **Περιγραφή**: Μια καμπύλη που γεμίζει το επίπεδο καλύπτοντας κάθε σημείο σε ένα τετραγωνικό επίπεδο.
- **Χρησιμότητα**: Χρησιμοποιείται στα μαθηματικά για να εξηγηθούν οι ιδιότητες των καμπυλών που γεμίζουν το επίπεδο.

12. Hilbert Curve

- **Περιγραφή**: Μια συνεχής καμπύλη που γεμίζει το επίπεδο και χρησιμοποιείται σε μοτίβα διάταξης δεδομένων.
- **Χρησιμότητα**: Χρησιμοποιείται σε συστήματα αρχείων και συμπίεση εικόνας.

13. T-Square Fractal

- **Περιγραφή**: Δημιουργείται τοποθετώντας τετράγωνα κατά τρόπο που θυμίζει το γράμμα T και επαναλαμβάνεται.
- **Χρησιμότητα**: Εφαρμόζεται στη μελέτη της διάταξης δεδομένων και της γεωμετρίας.

14. H-Fractal

- **Περιγραφή**: Ξεκινά με το γράμμα H και προστίθενται επαναληπτικά τμήματα, οδηγώντας σε ένα αυτοόμοιο σχήμα.
- **Χρησιμότητα**: Χρησιμοποιείται στην εκπαίδευση ως απλό fractal.

15. Vicsek Fractal

- **Περιγραφή**: Μοιάζει με το Sierpinski carpet και δημιουργείται αφαιρώντας κεντρικά τμήματα από τετράγωνα.
- **Χρησιμότητα**: Χρησιμοποιείται σε μαθηματικές μελέτες και στη γεωμετρία των fractals.

16. Menger Sponge

- **Περιγραφή**: Επέκταση του Sierpinski carpet σε τρεις διαστάσεις, δημιουργώντας ένα σφουγγάρι με απείρως πολλές οπές.
- **Χρησιμότητα**: Χρησιμοποιείται για να εξηγηθεί η έννοια της διάστασης και των γεωμετρικών παραμορφώσεων.

17. Hexaflake

- **Περιγραφή**: Δημιουργείται από ένα εξάγωνο, το οποίο χωρίζεται σε μικρότερα εξάγωνα με τη μορφή "flake".
- **Χρησιμότητα**: Χρησιμοποιείται στην τέχνη και τα γραφικά.

18. Gosper Curve (Gosper Island)

- **Περιγραφή**: Καμπύλη που δημιουργεί ένα fractal μοτίβο που θυμίζει νησί ή ακανόνιστη επιφάνεια.
- **Χρησιμότητα**: Χρησιμοποιείται σε 3D μοντέλα και στην ανάλυση φυσικών τοπίων.

19. Lévy C Curve

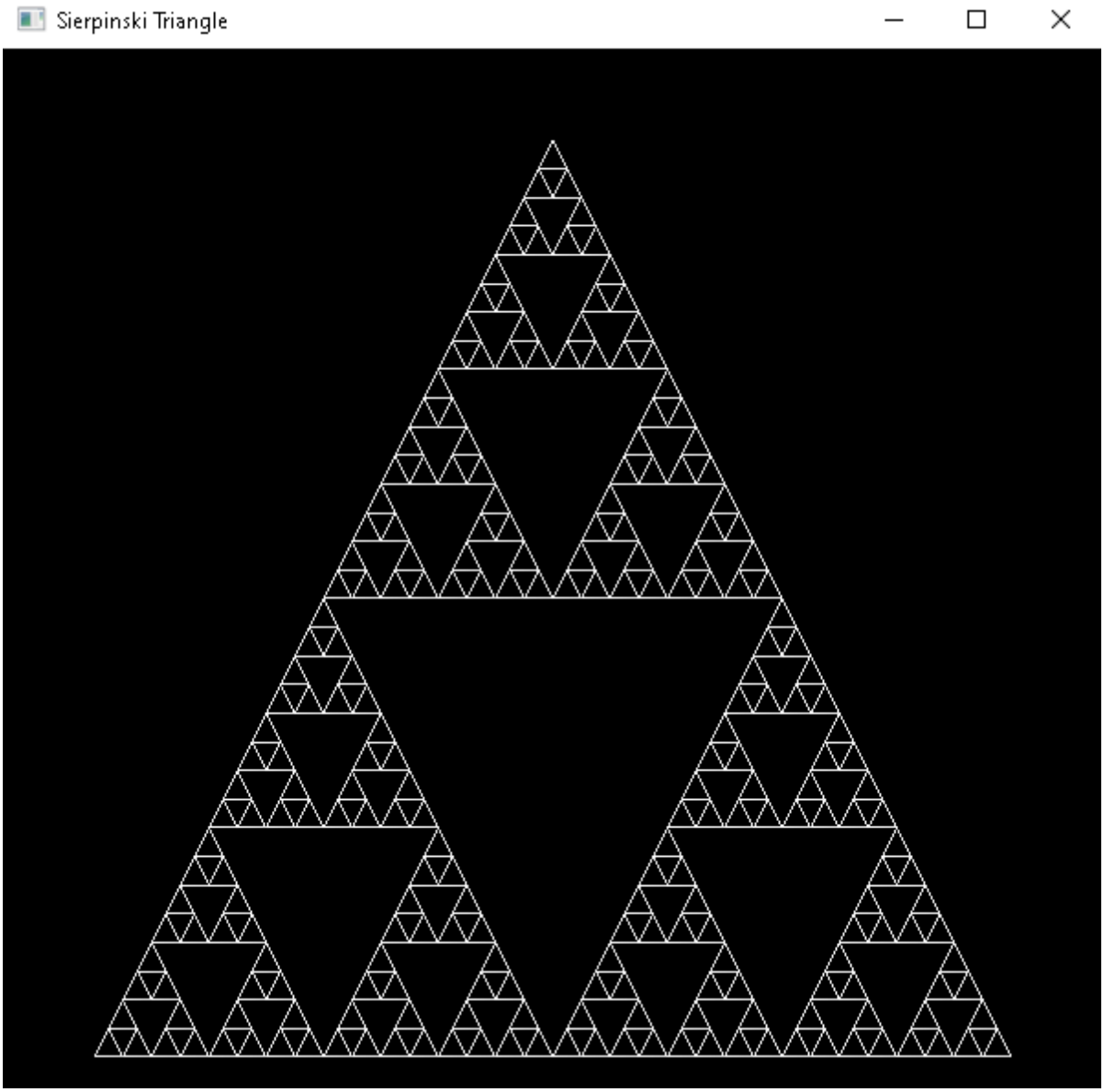
- **Περιγραφή**: Δημιουργείται από επαναλαμβανόμενες καμπύλες σε σχήμα C. Παράγει ένα πολύπλοκο μοτίβο σε μορφή σπείρας.
- **Χρησιμότητα**: Εφαρμόζεται στη δημιουργία διακοσμητικών σχεδίων και τη γεωμετρία.

20. Pentaflake

- **Περιγραφή**: Παρόμοιο με το Hexaflake, αλλά δημιουργείται από ένα πεντάγωνο, όπου κάθε πλευρά χωρίζεται σε πεντάγωνα.
- **Χρησιμότητα**: Χρησιμοποιείται στην τέχνη και για τη μελέτη των συμμετρικών δομών.

Τα fractals αυτά έχουν ευρεία χρησιμότητα στις τέχνες, τα γραφικά, τα μαθηματικά και τη βιολογία. Εξυπηρετούν στην κατανόηση του κόσμου και στην οπτικοποίηση σύνθετων συστημάτων και φυσικών φαινομένων.

Sierpinski Triangle



Για να σχεδιάσουμε το **Fractal του Sierpinski** χρησιμοποιώντας τη βιβλιοθήκη SGG, θα χρησιμοποιήσουμε μια αναδρομική (recursive) προσέγγιση . Το fractal αυτό δημιουργείται χωρίζοντας ένα τρίγωνο σε μικρότερα τρίγωνα, και αφαιρώντας το κεντρικό τμήμα, επαναλαμβάνοντας τη διαδικασία για κάθε τμήμα.

Ακολουθεί το πρόγραμμα για το Fractal του Sierpinski με αναλυτικά σχόλια :

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;
const int windowHeight = 600;

// Ορισμός του μέγιστου βάθους αναδρομής για το fractal
const int max_depth = 5; // Αυξάνοντας το βάθος, αυξάνεται η λεπτομέρεια του fractal

/**
```

```
 * Συνάρτηση σχεδίασης τριγώνου που σχεδιάζει ένα τρίγωνο, δεδομένων των συντεταγμένων
των κορυφών του.
 * @param x1, y1 Συντεταγμένες πρώτης κορυφής
 * @param x2, y2 Συντεταγμένες δεύτερης κορυφής
 * @param x3, y3 Συντεταγμένες τρίτης κορυφής
 * @param br Το πινέλο (Brush) που χρησιμοποιείται για το χρώμα και το στυλ του
τριγώνου
 */
void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3,
graphics::Brush& br) {
    graphics::drawLine(x1, y1, x2, y2, br);
    graphics::drawLine(x2, y2, x3, y3, br);
    graphics::drawLine(x3, y3, x1, y1, br);
}

/**
 * Αναδρομική συνάρτηση σχεδίασης του Sierpinski Triangle.
 * Σε κάθε επίπεδο αναδρομής, η συνάρτηση υποδιαιρεί το τρίγωνο σε τρία μικρότερα
τρίγωνα
 * και σχεδιάζει μόνο όταν φτάσει στο προκαθορισμένο βάθος.
 * @param x1, y1 Συντεταγμένες πρώτης κορυφής
 * @param x2, y2 Συντεταγμένες δεύτερης κορυφής
 * @param x3, y3 Συντεταγμένες τρίτης κορυφής
 * @param depth Το τρέχον βάθος αναδρομής
 * @param br Το πινέλο (Brush) για το χρώμα και το στυλ του fractal
 */
void sierpinski(float x1, float y1, float x2, float y2, float x3, float y3, int depth,
graphics::Brush& br) {
    // Βασική περίπτωση: αν το βάθος είναι μηδενικό, σχεδιάζουμε το τρίγωνο
    if (depth == 0) {
        drawTriangle(x1, y1, x2, y2, x3, y3, br);
    }
    else {
        // Υπολογισμός των μέσων σημείων κάθε πλευράς του τριγώνου
        float mid_x1 = (x1 + x2) / 2.0f;
        float mid_y1 = (y1 + y2) / 2.0f;
        float mid_x2 = (x2 + x3) / 2.0f;
        float mid_y2 = (y2 + y3) / 2.0f;
        float mid_x3 = (x3 + x1) / 2.0f;
        float mid_y3 = (y3 + y1) / 2.0f;

        // Αναδρομική κλήση για κάθε ένα από τα τρία νέα τρίγωνα που δημιουργούνται
        sierpinski(x1, y1, mid_x1, mid_y1, mid_x3, mid_y3, depth - 1, br);
        sierpinski(mid_x1, mid_y1, x2, y2, mid_x2, mid_y2, depth - 1, br);
        sierpinski(mid_x3, mid_y3, mid_x2, mid_y2, x3, y3, depth - 1, br);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από την βιβλιοθήκη για την απόδοση του παραθύρου.
 * Χρησιμοποιεί το πινέλο και τις συντεταγμένες για τη σχεδίαση του Sierpinski
Triangle.
 */
void draw() {
    // Ορισμός του πινέλου για τη σχεδίαση (λευκό χρώμα για τις γραμμές του τριγώνου)
    graphics::Brush br;
    br.fill_color[0] = 1.0f;  // Κόκκινο
    br.fill_color[1] = 1.0f;  // Πράσινο
    br.fill_color[2] = 1.0f;  // Μπλε (RGB για λευκό χρώμα)

    // Συντεταγμένες κορυφών του βασικού τριγώνου στην οθόνη
    float x1 = windowWidth / 2.0f;
    float y1 = 50.0f;
    float x2 = 50.0f;
    float y2 = windowHeight - 50.0f;
```

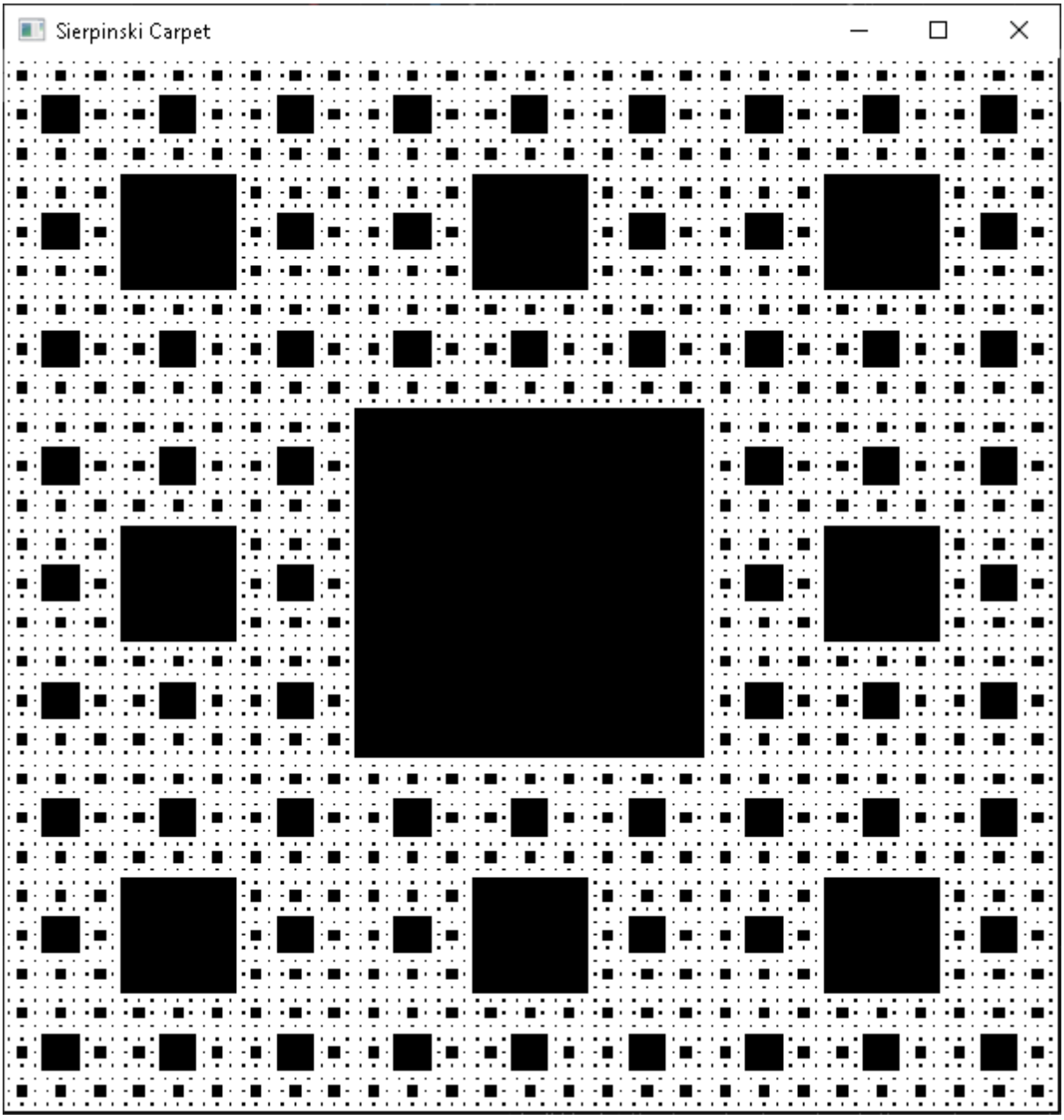
```
    float x3 = windowWidth - 50.0f;
    float y3 = windowHeight - 50.0f;

    // Κλήση της συνάρτησης για να σχεδιάσει το Sierpinski Triangle
    sierpinski(x1, y1, x2, y2, x3, y3, max_depth, br);
}

/**
 * Κύρια συνάρτηση του προγράμματος.
 * Δημιουργεί το παράθυρο, ρυθμίζει την αναδρομική συνάρτηση σχεδίασης του fractal και
 * ξεκινά τον κύριο βρόχο του παραθύρου.
 */
int main() {
    // Δημιουργία παραθύρου με τις προκαθορισμένες διαστάσεις
    graphics::createWindow(windowWidth, windowHeight, "Sierpinski Triangle");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων για την απόδοση του γραφικού περιβάλλοντος
    graphics::startMessageLoop();

    // Καταστροφή του παραθύρου μετά την έξοδο από τον κύκλο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και χρησιμοποιούμε το `max_depth` για να ορίσουμε το μέγιστο βάθος αναδρομής (π.χ., 5). Μπορούμε να αυξήσουμε αυτή την τιμή για περισσότερες λεπτομέρειες στο fractal.
2. **Συνάρτηση** `drawTriangle`:
    - Αυτή η συνάρτηση σχεδιάζει ένα τρίγωνο με τις κορυφές που δίνονται από τις συντεταγμένες (x1,y1)(x1, y1)(x1,y1), (x2,y2)(x2, y2)(x2,y2), και (x3,y3)(x3, y3)(x3,y3).
    - Η `drawLine()` χρησιμοποιείται για να σχεδιάσει τις πλευρές του τριγώνου.
3. **Συνάρτηση** `sierpinski`:
    - Αυτή η συνάρτηση χρησιμοποιεί αναδρομή για να σχεδιάσει το Fractal του Sierpinski.
    - Αν το `depth` είναι 0, τότε σχεδιάζουμε το τρίγωνο χρησιμοποιώντας τη `drawTriangle`.
    - Αν το `depth` δεν είναι 0, υπολογίζουμε τα μέσα σημεία κάθε πλευράς του τριγώνου και καλούμε τη `sierpinski` για τα τρία μικρότερα τρίγωνα που δημιουργούνται.
4. **Συνάρτηση** `draw`:
    - Εδώ ορίζουμε το χρώμα για τις γραμμές και καλούμε τη `sierpinski` για το βασικό τρίγωνο.
    - Οι αρχικές συντεταγμένες του βασικού τριγώνου ορίζονται στο κέντρο και στα κάτω άκρα του παραθύρου, σχηματίζοντας ένα ισοσκελές τρίγωνο.

5. **Κύριος Βρόχος**:
    - Ορίζουμε τη συνάρτηση `draw` για να σχεδιάσει το fractal στο παράθυρο και καλούμε το `startMessageLoop()` για να εκκινήσει το πρόγραμμα.

Sierpinsky Carpet.



Το **Sierpinski Carpet** είναι ένα fractal που δημιουργείται επαναληπτικά μέσω της διαίρεσης ενός τετραγώνου σε εννέα μικρότερα τετράγωνα και αφαίρεσης του κεντρικού τετραγώνου. Στη συνέχεια, η ίδια διαδικασία εφαρμόζεται στα υπόλοιπα οκτώ τετράγωνα. Ακολουθεί το πρόγραμμα που σχεδιάζει το Sierpinski Carpet χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός των διαστάσεων του παραθύρου σε pixels
const int windowWidth = 600;
const int windowHeight = 600;

// Ορισμός του μέγιστου βάθους αναδρομής για τη σχεδίαση του fractal
const int max_depth = 5; // Αυξάνοντας το βάθος, αυξάνεται η λεπτομέρεια του fractal
```

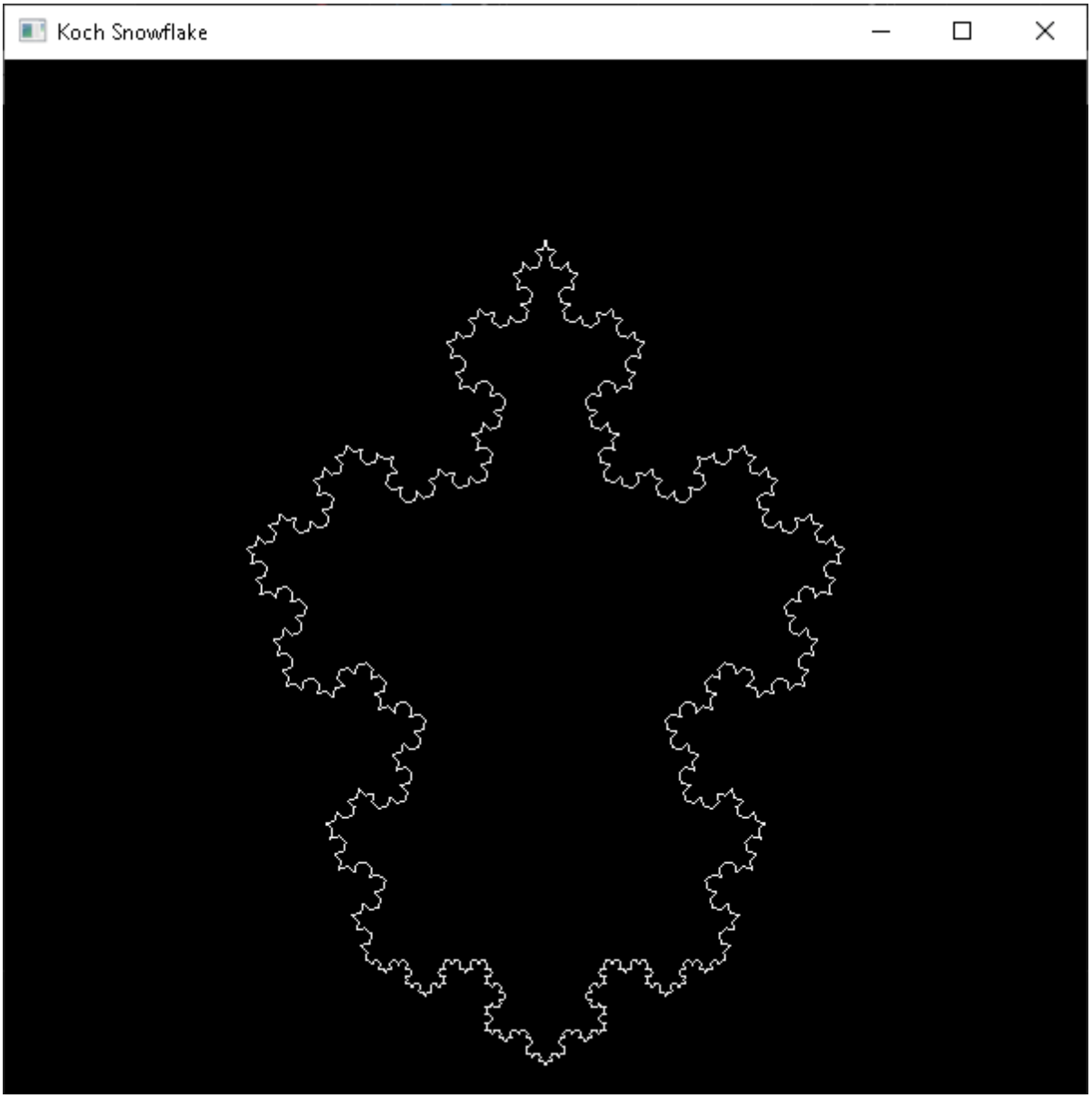
```
/**
 * Αναδρομική συνάρτηση που σχεδιάζει το Sierpinski Carpet Fractal.
 * Σε κάθε επίπεδο αναδρομής, η συνάρτηση υποδιαιρεί το τετράγωνο σε 9 μικρότερα
 * τετράγωνα, αφήνοντας το κεντρικό κενό και σχεδιάζει τα υπόλοιπα.
 * @param x Το x του κεντρικού σημείου του τετραγώνου προς σχεδίαση
 * @param y Το y του κεντρικού σημείου του τετραγώνου προς σχεδίαση
 * @param size Το μέγεθος της πλευράς του τετραγώνου
 * @param depth Το τρέχον βάθος αναδρομής
 */
void sierpinski_carpet(float x, float y, float size, int depth) {
    // Βασική περίπτωση: Όταν φτάσουμε στο μέγιστο βάθος, σχεδιάζουμε το τετράγωνο
    if (depth == 0) {
        graphics::Brush br;
        br.fill_color[0] = 1.0f;  // Ορισμός χρώματος (λευκό) για τα τετράγωνα του
fractal
        br.fill_color[1] = 1.0f;
        br.fill_color[2] = 1.0f;
        graphics::drawRect(x, y, size, size, br);  // Σχεδίαση του λευκού τετραγώνου
    }
    else {
        // Υπολογισμός του νέου μεγέθους για τα μικρότερα τετράγωνα
        float newSize = size / 3.0f;

        // Διάσχιση των εννέα θέσεων για να τοποθετηθούν τα νέα τετράγωνα γύρω από το
κεντρικό
        for (int dx = 0; dx < 3; dx++) {
            for (int dy = 0; dy < 3; dy++) {
                // Παράκαμψη του κεντρικού τετραγώνου
                if (dx == 1 && dy == 1) continue;

                // Αναδρομική κλήση για τη σχεδίαση του τετραγώνου στα υπόλοιπα οκτώ
σημεία
                sierpinski_carpet(x + dx * newSize, y + dy * newSize, newSize, depth -
1);
            }
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
 * Ορίζει το φόντο και ξεκινά τη σχεδίαση του Sierpinski Carpet.
 */
void draw() {
    // Ορισμός του φόντου του παραθύρου σε μαύρο χρώμα για αντίθεση
    graphics::Brush background;
    background.fill_color[0] = 0.0f; // Κόκκινο
    background.fill_color[1] = 0.0f; // Πράσινο
    background.fill_color[2] = 0.0f; // Μπλε (RGB για μαύρο χρώμα)
    graphics::setWindowBackground(background);

    // Κλήση της συνάρτησης για τη σχεδίαση του Sierpinski Carpet, ξεκινώντας από το
κέντρο του παραθύρου
    sierpinski_carpet(0, 0, windowWidth, max_depth);
}

/**
 * Κύρια συνάρτηση του προγράμματος.
 * Δημιουργεί το παράθυρο, ρυθμίζει τη συνάρτηση σχεδίασης του fractal
 * και ξεκινά τον κύριο βρόχο του παραθύρου.
 */
int main() {
    // Δημιουργία παραθύρου με προκαθορισμένες διαστάσεις και τίτλο
    graphics::createWindow(windowWidth, windowHeight, "Sierpinski Carpet");
```

```
    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων της γραφικής βιβλιοθήκης
    graphics::startMessageLoop();

    // Καταστροφή του παραθύρου μετά την έξοδο από τον κύκλο μηνυμάτων
    graphics::destroyWindow();

    return 0;
}
```

Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Μέγιστου Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η αναδρομή γίνεται μέχρι το `max_depth`. Αυτή η παράμετρος καθορίζει πόσα επίπεδα θα δημιουργηθούν. Όσο μεγαλύτερο το `max_depth`, τόσο περισσότερα τετράγωνα και λεπτομέρειες θα έχει το fractal.
2. **Συνάρτηση** `sierpinski_carpet()`:
    - Αυτή η συνάρτηση χρησιμοποιεί αναδρομή για να σχεδιάσει το Sierpinski Carpet.
    - Αν το `depth` είναι 0, σημαίνει ότι βρισκόμαστε στο τέλος της αναδρομής, και απλά σχεδιάζουμε το τετράγωνο στη θέση `(x, y)` με μέγεθος `size`.
    - Αν το `depth` δεν είναι 0, διαιρούμε το `size` σε 3 μικρότερα τετράγωνα και επαναλαμβάνουμε τη διαδικασία για κάθε ένα από τα οκτώ τετράγωνα, παραλείποντας το κεντρικό.
3. **Συνάρτηση** `draw()`:
    - Η `draw()` ορίζεται από τη βιβλιοθήκη SGG και καλείται αυτόματα για τη σχεδίαση στο παράθυρο.
    - καλεί τη συνάρτηση `sierpinski_carpet()` για να αρχίσει τη σχεδίαση του fractal από το κέντρο του παραθύρου.
4. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Ο κύριος βρόχος αρχίζει με την `startMessageLoop()`, που διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Koch Snowflake



Το **Koch Snowflake** είναι ένα γνωστό fractal που δημιουργείται ξεκινώντας από ένα ισόπλευρο τρίγωνο και αντικαθιστώντας το κέντρο κάθε πλευράς με ένα μικρότερο τρίγωνο. Αυτό επαναλαμβάνεται επ’ άπειρον, δημιουργώντας ένα σχήμα που θυμίζει χιονονιφάδα.

Για την υλοποίηση του Koch Snowflake με τη βιβλιοθήκη SGG, θα χρησιμοποιήσουμε μια αναδρομική συνάρτηση που υποδιαιρεί κάθε πλευρά ενός ισόπλευρου τριγώνου σε τρία ίσα τμήματα και προσθέτει ένα τρίγωνο στη μέση, σχηματίζοντας τη χαρακτηριστική μορφή της χιονονιφάδας.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου σε pixels
const int windowWidth = 600;
const int windowHeight = 600;
```

```
// Μέγιστος αριθμός επαναλήψεων για τη σχεδίαση του fractal Koch Snowflake
const int max_depth = 4; // Αυξάνοντας το βάθος, προστίθενται περισσότερες
λεπτομέρειες

/**
 * Συνάρτηση που σχεδιάζει το Koch Snowflake αναδρομικά.
 * Διαιρεί κάθε γραμμή σε τρία τμήματα, δημιουργώντας ένα ισόπλευρο τρίγωνο στο μέσο
της γραμμής.
 * @param x1 Η x-συντεταγμένη του αρχικού σημείου της γραμμής
 * @param y1 Η y-συντεταγμένη του αρχικού σημείου της γραμμής
 * @param x2 Η x-συντεταγμένη του τελικού σημείου της γραμμής
 * @param y2 Η y-συντεταγμένη του τελικού σημείου της γραμμής
 * @param depth Το τρέχον βάθος αναδρομής
 * @param br Το πινέλο (brush) που χρησιμοποιείται για τη σχεδίαση
 */
void koch_snowflake(float x1, float y1, float x2, float y2, int depth,
graphics::Brush& br) {
    // Βασική περίπτωση: Όταν φτάσουμε στο μέγιστο βάθος, σχεδιάζουμε την απλή γραμμή
    if (depth == 0) {
        graphics::drawLine(x1, y1, x2, y2, br);
    }
    else {
        // Υπολογισμός των τριών σημείων διαίρεσης της γραμμής
        float dx = (x2 - x1) / 3.0f;
        float dy = (y2 - y1) / 3.0f;

        // Σημεία στα 1/3 και 2/3 της γραμμής
        float xA = x1 + dx;
        float yA = y1 + dy;
        float xB = x1 + 2 * dx;
        float yB = y1 + 2 * dy;

        // Υπολογισμός του σημείου κορυφής του ισόπλευρου τριγώνου που δημιουργείται
        float midX = (x1 + x2) / 2.0f;
        float midY = (y1 + y2) / 2.0f;
        float peakX = midX + sqrt(3.0f) * (yA - yB) / 2.0f;
        float peakY = midY + sqrt(3.0f) * (xB - xA) / 2.0f;

        // Αναδρομική κλήση για τις τέσσερις γραμμές που αποτελούν το επόμενο επίπεδο
        koch_snowflake(x1, y1, xA, yA, depth - 1, br);
        koch_snowflake(xA, yA, peakX, peakY, depth - 1, br);
        koch_snowflake(peakX, peakY, xB, yB, depth - 1, br);
        koch_snowflake(xB, yB, x2, y2, depth - 1, br);
    }
}

/**
 * Συνάρτηση σχεδίασης για το Koch Snowflake, βασισμένη σε ισόπλευρο τρίγωνο.
 * Αναδρομικά καλεί τη `koch_snowflake` για τη σχεδίαση και των τριών πλευρών.
 */
void drawKochSnowflake() {
    graphics::Brush br;
    br.fill_color[0] = 1.0f;  // Λευκό χρώμα για τις γραμμές του fractal
    br.fill_color[1] = 1.0f;
    br.fill_color[2] = 1.0f;

    // Συντεταγμένες για τις κορυφές του βασικού ισόπλευρου τριγώνου
    float x1 = windowWidth / 2.0f;
    float y1 = 100.0f;
    float x2 = 200.0f;
    float y2 = windowHeight - 100.0f;
    float x3 = windowWidth - 200.0f;
    float y3 = windowHeight - 100.0f;
```

```
    // Κλήση της συνάρτησης για σχεδίαση του Koch Snowflake στις τρεις πλευρές του τριγώνου
    koch_snowflake(x1, y1, x2, y2, max_depth, br);
    koch_snowflake(x2, y2, x3, y3, max_depth, br);
    koch_snowflake(x3, y3, x1, y1, max_depth, br);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
 * Καθορίζει το φόντο και ξεκινά τη σχεδίαση του Koch Snowflake.
 */
void draw() {
    graphics::Brush background;
    background.fill_color[0] = 0.0f; // Μαύρο χρώμα φόντου για αντίθεση
    background.fill_color[1] = 0.0f;
    background.fill_color[2] = 0.0f;
    graphics::setWindowBackground(background);

    // Κλήση της συνάρτησης για να σχεδιάσει το Koch Snowflake
    drawKochSnowflake();
}

/**
 * Κύρια συνάρτηση του προγράμματος.
 * Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και εκκινεί τον βρόχο μηνυμάτων.
 */
int main() {
    // Δημιουργία παραθύρου με προκαθορισμένες διαστάσεις και τίτλο
    graphics::createWindow(windowWidth, windowHeight, "Koch Snowflake");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η αναδρομή γίνεται μέχρι το `max_depth`, που καθορίζει τη λεπτομέρεια της χιονονιφάδας. Όσο μεγαλύτερο το `max_depth`, τόσο περισσότερες λεπτομέρειες θα έχει το fractal.
2. **Συνάρτηση** `koch_snowflake()`:
    - Αυτή η συνάρτηση σχεδιάζει κάθε πλευρά του fractal αναδρομικά.
    - Αν το `depth` είναι 0, η συνάρτηση σχεδιάζει απλά μια γραμμή από το σημείο `(x1, y1)` στο `(x2, y2)`.
    - Αν το `depth` δεν είναι 0, η συνάρτηση διαιρεί τη γραμμή σε τρία ίσα τμήματα και υπολογίζει το σημείο κορυφής του τριγώνου που πρέπει να προστεθεί στη μέση της γραμμής.

- Στη συνέχεια, η συνάρτηση καλείται αναδρομικά για τις τέσσερις νέες γραμμές που δημιουργούνται.

3. **Συνάρτηση** `drawKochSnowflake()`:
    - Αυτή η συνάρτηση ορίζει τις συντεταγμένες για τις τρεις κορυφές του αρχικού ισόπλευρου τριγώνου και καλεί την `koch_snowflake()` για κάθε μία από τις τρεις πλευρές του τριγώνου.
4. **Συνάρτηση** `draw()`:
    - Η `draw()` ορίζεται από τη βιβλιοθήκη SGG και καλείται αυτόματα για να σχεδιάσει το fractal στο παράθυρο.
    - καλεί τη `drawKochSnowflake()` για να αρχίσει τη σχεδίαση της χιονονιφάδας.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Ο κύριος βρόχος αρχίζει με την `startMessageLoop()`, που διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Barnsley Fern

Το **Barnsley Fern** είναι ένα fractal που αναπαριστά το σχήμα ενός φύλλου φτέρης. Δημιουργείται μέσω επαναλαμβανόμενων συναρτήσεων, όπου κάθε σημείο (pixel) υπολογίζεται με βάση μία από τις τέσσερις γραμμικές μετασχηματιστικές συναρτήσεις που καθορίζουν το σχήμα του φύλλου.

Η σχεδίαση του Barnsley Fern μπορεί να γίνει με την τεχνική της **Iterated Function System (IFS)**, δηλαδή ένα σύστημα επαναλαμβανόμενων συναρτήσεων. Στο πρόγραμμα αυτό, χρησιμοποιούμε μια τυχαία διαδικασία για να καθορίσουμε ποια από τις συναρτήσεις θα εφαρμοστεί κάθε φορά, δημιουργώντας ένα φύλλο φτέρης.

Κώδικας

```
#include "sgg/graphics.h"
#include <cstdlib>   // Χρησιμοποιείται για παραγωγή τυχαίων αριθμών
#include <ctime>     // Χρησιμοποιείται για αρχικοποίηση των τυχαίων αριθμών

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 350;   // Πλάτος παραθύρου
const int windowHeight = 700;  // Ύψος παραθύρου

// Αριθμός επαναλήψεων
const int num_iterations = 100000; // Ο αριθμός σημείων που θα σχεδιαστούν για το
Barnsley Fern

// Συντελεστές για μετασχηματισμό συντεταγμένων στον καμβά σχεδίασης
const float scale_x = 60.0f;   // Κλιμάκωση στον άξονα x για κατάλληλη αναλογία
const float scale_y = 60.0f;   // Κλιμάκωση στον άξονα y για κατάλληλη αναλογία
const float offset_x = windowWidth / 2.0f;       // Μετατόπιση κέντρου στον x άξονα
const float offset_y = windowHeight - 50.0f;     // Μετατόπιση προς τα κάτω στον y
άξονα

/**
 * Συνάρτηση barnsley_fern
 * Εφαρμόζει τέσσερις διαφορετικούς affine μετασχηματισμούς αναδρομικά για να
δημιουργήσει το Barnsley Fern.
 * Οι μετασχηματισμοί επιλέγονται τυχαία, δημιουργώντας τον χαρακτηριστικό σχηματισμό
του φτέρη (fern).
 * Σχεδιάζει ένα σημείο κάθε φορά στο παράθυρο.
 */
void barnsley_fern() {
    graphics::Brush br;  // Το πινέλο για το χρώμα του fractal (προεπιλεγμένο χρώμα)

    float x = 0.0f;   // Αρχική x-συντεταγμένη
    float y = 0.0f;   // Αρχική y-συντεταγμένη

    // Βρόχος για τον υπολογισμό και σχεδίαση των σημείων
    for (int i = 0; i < num_iterations; ++i) {
        float next_x, next_y;
        float r = static_cast<float>(rand()) / RAND_MAX;  // Τυχαίος αριθμός [0,1] για
επιλογή μετασχηματισμού

        // Επιλογή μετασχηματισμού βάσει της πιθανότητας που ορίζεται για το κάθε
fractal τμήμα
        if (r < 0.01f) {  // Πιθανότητα F1
            // F1: Μικρό τμήμα στο κάτω μέρος της φτέρης
            next_x = 0.0f;
            next_y = 0.16f * y;
        }
        else if (r < 0.86f) {  // Πιθανότητα F2
            // F2: Κύριο τμήμα της φτέρης, καθορίζει τη δομή
            next_x = 0.85f * x + 0.04f * y;
            next_y = -0.04f * x + 0.85f * y + 1.6f;
        }
        else if (r < 0.93f) {  // Πιθανότητα F3
            // F3: Τμήμα που δημιουργεί τις πλευρικές "φύτρες" της φτέρης
            next_x = 0.2f * x - 0.26f * y;
            next_y = 0.23f * x + 0.22f * y + 1.6f;
        }
        else {  // Πιθανότητα F4
            // F4: Τμήμα που προσθέτει περισσότερες μικρές "φύτρες"
            next_x = -0.15f * x + 0.28f * y;
            next_y = 0.26f * x + 0.24f * y + 0.44f;
        }
```

```
        // Ενημέρωση των τιμών των x και y για την επόμενη επανάληψη
        x = next_x;
        y = next_y;

        // Σχεδιασμός του σημείου στο παράθυρο με μετασχηματισμό για την αντιστοίχιση στον καμβά
        graphics::drawRect(offset_x + x * scale_x, offset_y - y * scale_y, 1, 1, br);
    }
}

/**
 * Συνάρτηση σχεδίασης draw
 * Καλεί τη συνάρτηση barnsley_fern για τη σχεδίαση του fractal Barnsley Fern στον καμβά.
 */
void draw() {
    barnsley_fern();  // Κλήση της συνάρτησης που σχεδιάζει το Barnsley Fern
}

int main() {
    // Αρχικοποίηση του γεννήτριας τυχαίων αριθμών για διαφορετικά fractals σε κάθε εκτέλεση
    srand(static_cast<unsigned int>(time(0)));

    // Δημιουργία παραθύρου για το fractal Barnsley Fern
    graphics::createWindow(windowWidth, windowHeight, "Barnsley Fern");

    // Ορισμός της συνάρτησης σχεδίασης draw, που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

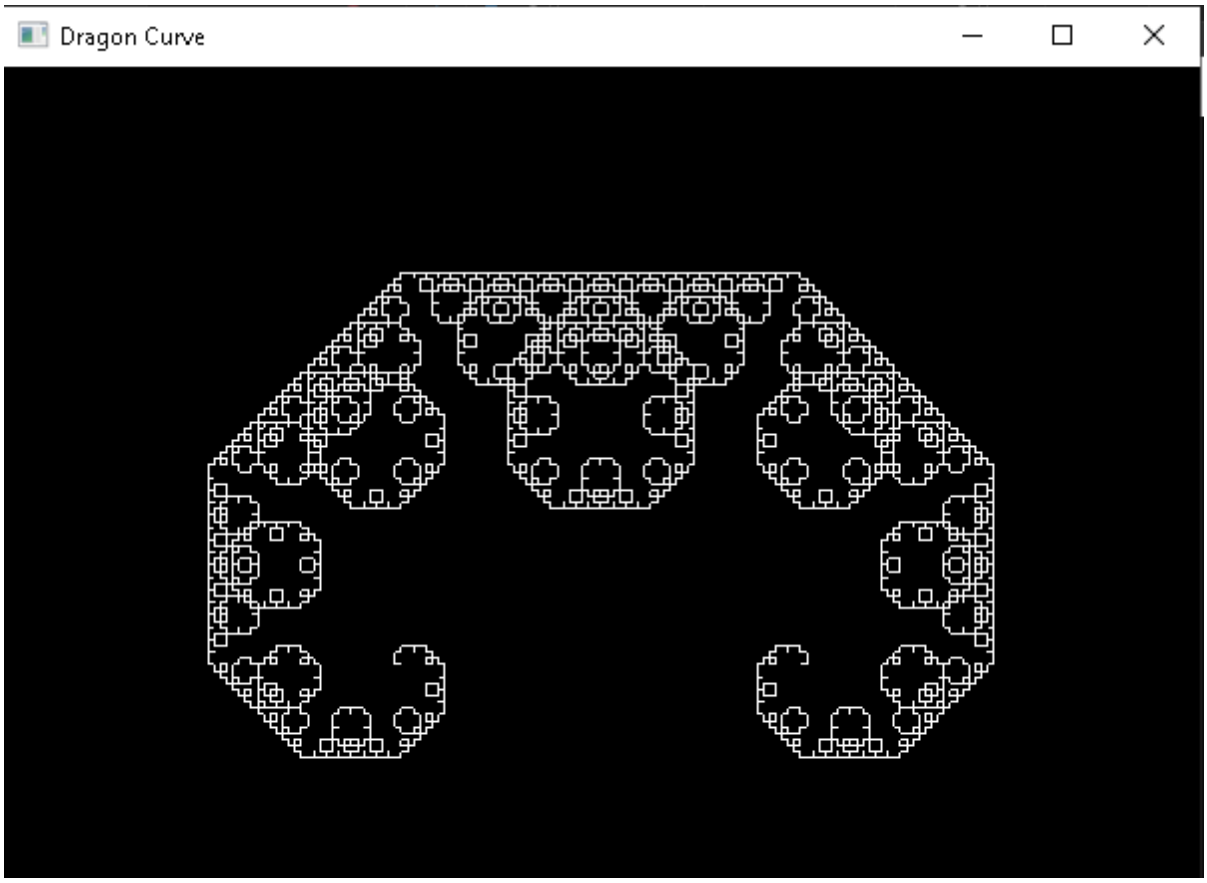
Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Αριθμού Επαναλήψεων**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και ο αριθμός επαναλήψεων (`num_iterations`) καθορίζει πόσα σημεία θα σχεδιάσουμε. Περισσότερες επαναλήψεις προσδίδουν μεγαλύτερη λεπτομέρεια στο fractal.
2. **Κλιμάκωση και Μετατόπιση**:
    - Οι παράμετροι `scale_x` και `scale_y` καθορίζουν την κλίμακα των αξόνων xxx και yyy, ώστε το fractal να έχει το κατάλληλο μέγεθος για το παράθυρο.
    - Οι παράμετροι `offset_x` και `offset_y` καθορίζουν τη θέση του fractal στο παράθυρο.
3. **Συνάρτηση** `barnsley_fern()`:
    - Αυτή η συνάρτηση είναι υπεύθυνη για τον υπολογισμό και τη σχεδίαση των σημείων που δημιουργούν το Barnsley Fern.
    - Ξεκινάμε από το σημείο (0,0)(0, 0)(0,0) και για κάθε επανάληψη επιλέγουμε τυχαία μία από τις τέσσερις μετασχηματιστικές συναρτήσεις (F1, F2, F3, F4) με βάση τον τυχαίο αριθμό rrr.

- Οι συναρτήσεις αυτές καθορίζουν τη μορφή και την ανάπτυξη του fractal, μετακινώντας το σημείο σε νέες θέσεις.

4. **Επιλογή Συναρτήσεων F1 έως F4**:
    - Η F1 εφαρμόζεται με πιθανότητα 1%, η F2 με 85%, η F3 με 7%, και η F4 με 7%.
    - Αυτές οι συναρτήσεις παράγουν τα διαφορετικά τμήματα του fractal, όπως ο κεντρικός κορμός, οι κλαδιά και τα φύλλα.
5. **Σχεδίαση του Σημείου**:
    - Το κάθε νέο σημείο σχεδιάζεται στο παράθυρο με συντεταγμένες προσαρμοσμένες από το κέντρο του παραθύρου και κλιμακωμένες για να δημιουργήσουν το σχήμα της φτέρης.
6. **Συνάρτηση** `draw()`:
    - Καλεί τη `barnsley_fern()` για να σχεδιάσει το fractal.
7. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Το παράθυρο παραμένει ενεργό μέσω της `startMessageLoop()` και καταστρέφεται όταν ο χρήστης το κλείσει.

Dragon Curve



Το **Dragon Curve** είναι ένα fractal που δημιουργείται μέσω αναδρομής. Ξεκινά με μία γραμμή και σε κάθε επίπεδο αναδρομής προστίθεται ένα νέο τμήμα γραμμής σε γωνία 90 μοιρών. Κάθε νέο επίπεδο δημιουργεί περισσότερες καμπύλες και οδηγεί στο χαρακτηριστικό σχήμα του "δράκου".

Για την υλοποίηση του **Dragon Curve** με τη βιβλιοθήκη SGG, θα χρησιμοποιήσουμε αναδρομή για να σχεδιάσουμε τις γραμμές με τις απαιτούμενες γωνίες. Το Dragon Curve μπορεί να κατασκευαστεί με μια αναδρομική συνάρτηση που διαχωρίζει και περιστρέφει το αρχικό σχήμα σε κάθε επίπεδο βάθους.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;  // Πλάτος παραθύρου
const int windowHeight = 600; // Ύψος παραθύρου

// Μέγιστο βάθος αναδρομής για τη λεπτομέρεια του Dragon Curve
const int max_depth = 12; // Αύξηση αυτής της τιμής αυξάνει τη λεπτομέρεια του fractal

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Dragon Curve fractal.
 * @param x1 Η αρχική x-συντεταγμένη της γραμμής
 * @param y1 Η αρχική y-συντεταγμένη της γραμμής
 * @param x2 Η τελική x-συντεταγμένη της γραμμής
 * @param y2 Η τελική y-συντεταγμένη της γραμμής
```

```
 * @param depth Το τρέχον βάθος αναδρομής (σταματάει όταν φτάσει στο 0)
 */
void drawDragonCurve(float x1, float y1, float x2, float y2, int depth) {
    if (depth == 0) {
        // Εάν το βάθος είναι 0, σχεδιάζουμε τη γραμμή από (x1, y1) έως (x2, y2)
        graphics::Brush br;  // Δημιουργία αντικειμένου πινέλου για τον καθορισμό στυλ
        graphics::drawLine(x1, y1, x2, y2, br);  // Σχεδιάζει γραμμή με τις δεδομένες
συντεταγμένες
    }
    else {
        // Υπολογισμός των συντεταγμένων του μέσου σημείου με περιστροφή κατά 45
μοίρες
        float mid_x = (x1 + x2) / 2.0f + (y2 - y1) / 2.0f;
        float mid_y = (y1 + y2) / 2.0f - (x2 - x1) / 2.0f;

        // Αναδρομική κλήση για τα δύο μέρη του fractal
        drawDragonCurve(x1, y1, mid_x, mid_y, depth - 1);  // Πρώτο μισό
        drawDragonCurve(mid_x, mid_y, x2, y2, depth - 1);  // Δεύτερο μισό
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ για τη σχεδίαση του Dragon Curve.
 */
void draw() {
    graphics::Brush background; // Αντικείμενο πινέλου για το φόντο
    graphics::setWindowBackground(background); // Ορισμός μαύρου φόντου για αντίθεση

    // Ορισμός αρχικών και τελικών σημείων για την αρχική γραμμή
    float startX = windowWidth / 3.0f;  // Αρχική x-συντεταγμένη
    float startY = windowHeight / 2.0f; // Αρχική y-συντεταγμένη
    float endX = 2.0f * windowWidth / 3.0f; // Τελική x-συντεταγμένη
    float endY = windowHeight / 2.0f;       // Τελική y-συντεταγμένη

    // Κλήση της αναδρομικής συνάρτησης για τη σχεδίαση του Dragon Curve
    drawDragonCurve(startX, startY, endX, endY, max_depth);
}

int main() {
    // Δημιουργία παραθύρου για τη σχεδίαση του Dragon Curve fractal
    graphics::createWindow(windowWidth, windowHeight, "Dragon Curve");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου της βιβλιοθήκης για τη διαχείριση του παραθύρου
    graphics::startMessageLoop();

    // Καταστροφή του παραθύρου όταν ολοκληρωθεί ο βρόχος
    graphics::destroyWindow();

    return 0;
}
```

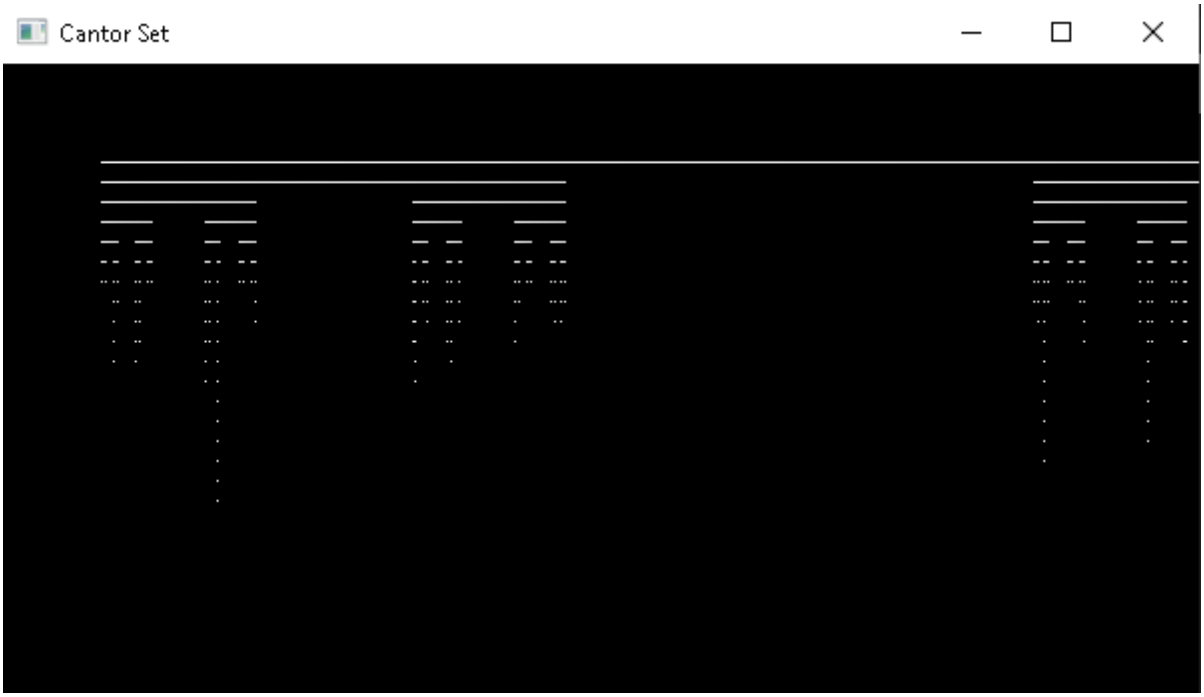
Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει το επίπεδο αναδρομής για το Dragon Curve. Όσο μεγαλύτερο το βάθος, τόσο περισσότερες λεπτομέρειες θα έχει η καμπύλη.
2. **Συνάρτηση** `drawDragonCurve()`:

- Αυτή η συνάρτηση χρησιμοποιεί αναδρομή για να σχεδιάσει το Dragon Curve.
- Αν το `depth` είναι 0, σχεδιάζουμε απλά μια γραμμή από το (x1,y1)(x1, y1)(x1,y1) στο (x2,y2)(x2, y2)(x2,y2).
- Αν το `depth` δεν είναι 0, υπολογίζουμε το μέσο σημείο των δύο σημείων και εφαρμόζουμε περιστροφή 45 μοιρών, προσθέτοντας έτσι το χαρακτηριστικό "γύρισμα" της καμπύλης.
- Μετά την περιστροφή, η συνάρτηση καλεί τον εαυτό της αναδρομικά για να σχεδιάσει το πρώτο και δεύτερο μισό της καμπύλης, δημιουργώντας το σχήμα του "δράκου".

3. **Συνάρτηση** `draw()`:
    - Η `draw()` είναι η κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
    - καλεί τη `drawDragonCurve()` με αρχικές συντεταγμένες, έτσι ώστε η καμπύλη να ξεκινά από το κέντρο του παραθύρου.
4. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Cantor Set



Το **Cantor Set** είναι ένα απλό αλλά ενδιαφέρον fractal, το οποίο δημιουργείται με τη συνεχή διαίρεση μιας γραμμής. Ξεκινάμε με μια γραμμή και σε κάθε επίπεδο αφαιρούμε το μεσαίο τρίτο κάθε γραμμής. Η διαδικασία συνεχίζεται επαναληπτικά για τα υπόλοιπα δύο τμήματα, δημιουργώντας ένα fractal σύνολο από διακεκομμένες γραμμές.

Παρακάτω είναι το πρόγραμμα για τη σχεδίαση του **Cantor Set** χρησιμοποιώντας τη βιβλιοθήκη SGG, με αναλυτικά σχόλια .

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;  // Πλάτος παραθύρου
const int windowHeight = 600; // Ύψος παραθύρου

// Ορισμός του μέγιστου βάθους αναδρομής για το Cantor Set
const int max_depth = 15;  // Ο αριθμός των επιπέδων διακεκομμένων γραμμών
// Καθορίζει την κατακόρυφη απόσταση μεταξύ των γραμμών
const int line_spacing = 10;  // Απόσταση μεταξύ των διαδοχικών επιπέδων

/**
 * Συνάρτηση που σχεδιάζει το Cantor Set χρησιμοποιώντας αναδρομή
 * @param x Η αρχική x-συντεταγμένη της γραμμής
 * @param y Η αρχική y-συντεταγμένη της γραμμής
 * @param length Το μήκος της τρέχουσας γραμμής
 * @param depth Το τρέχον βάθος αναδρομής (σταματά όταν φτάσει στο 0)
 */
void drawCantorSet(float x, float y, float length, int depth) {
    if (depth == 0) {
        return; // Σταματάει αν φτάσουμε το μέγιστο βάθος
    }

    // Δημιουργία πινέλου για σχεδίαση
    graphics::Brush br;
```

```
    // Σχεδίαση γραμμής στο τρέχον επίπεδο
    graphics::drawLine(x, y, x + length, y, br);

    // Υπολογισμός μήκους για το επόμενο επίπεδο
    float newLength = length / 3.0f;  // Μειώνουμε το μήκος κάθε νέας γραμμής στο ένα
τρίτο

    // Αναδρομική κλήση για τις δύο νέες γραμμές
    drawCantorSet(x, y + line_spacing, newLength, depth - 1);  // Αριστερή γραμμή
    drawCantorSet(x + 2 * newLength, y + line_spacing, newLength, depth - 1);  //
Δεξιά γραμμή
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη για κάθε καρέ
 */
void draw() {
    // Δημιουργία πινέλου για το φόντο
    graphics::Brush background;
    graphics::setWindowBackground(background); // Ορισμός μαύρου φόντου

    // Αρχικές συντεταγμένες και μήκος για το Cantor Set
    float startX = 50.0f;    // Αρχικό x για την πρώτη γραμμή
    float startY = 50.0f;    // Αρχικό y για την πρώτη γραμμή
    float startLength = 500.0f;  // Μήκος της πρώτης γραμμής

    // Κλήση της συνάρτησης σχεδίασης για το Cantor Set
    drawCantorSet(startX, startY, startLength, max_depth);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Cantor Set");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύριου βρόχου της βιβλιοθήκης
    graphics::startMessageLoop();

    // Καθαρισμός πόρων και καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

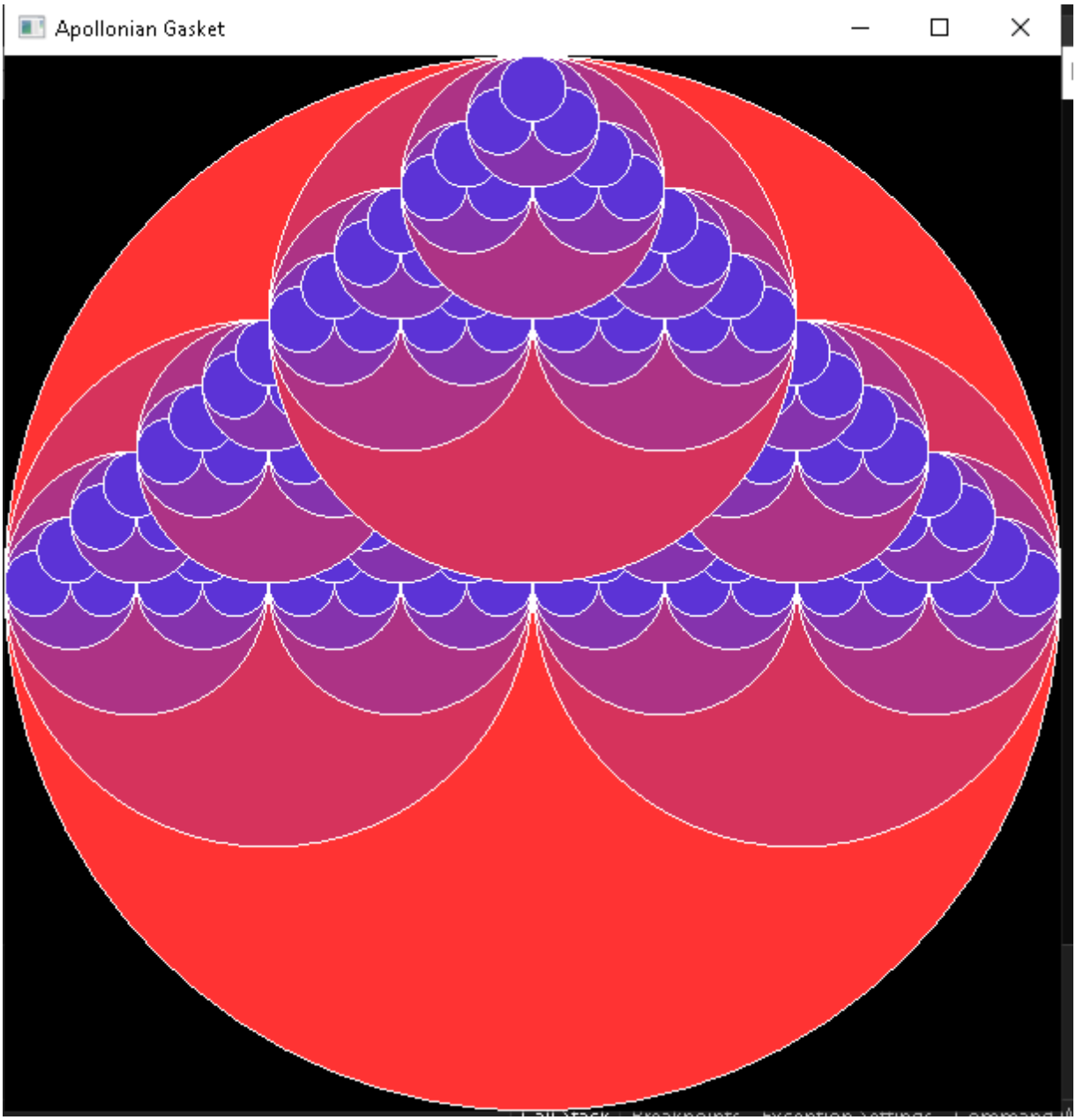
1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει πόσα επίπεδα του Cantor Set θα δημιουργηθούν. Όσο μεγαλύτερο το βάθος, τόσο περισσότερες γραμμές θα εμφανιστούν.
2. **Απόσταση Μεταξύ των Γραμμών**:
    - Η παράμετρος `line_spacing` ορίζει την κατακόρυφη απόσταση μεταξύ των γραμμών. Αυτή η απόσταση κρατά τις γραμμές διακριτές, ώστε να εμφανίζονται καθαρά.
3. **Συνάρτηση** `drawCantorSet()`:

- Αυτή η συνάρτηση είναι αναδρομική και δημιουργεί το Cantor Set.
- Αν το `depth` είναι 0, σταματάμε τη διαδικασία.
- Σχεδιάζουμε μια γραμμή στο τρέχον επίπεδο χρησιμοποιώντας τη `drawLine()`.
- Στη συνέχεια, υπολογίζουμε το νέο μήκος γραμμής (`newLength`), το οποίο είναι το ένα τρίτο του αρχικού μήκους.
- Καλούμε τη `drawCantorSet()` αναδρομικά για τις δύο νέες γραμμές, οι οποίες τοποθετούνται αριστερά και δεξιά με απόσταση από τη μεσαία γραμμή.

4. **Συνάρτηση** `draw()`:
    - Η `draw()` είναι η κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.

5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Το παράθυρο παραμένει ενεργό μέσω της `startMessageLoop()` και καταστρέφεται όταν ο χρήστης το κλείσει.

Apollonian Gasket



Το **Apollonian Gasket** είναι ένα fractal που δημιουργείται με τη συνεχή τοποθέτηση κύκλων μέσα σε ένα αρχικό τριγωνικό μοτίβο, όπου κάθε νέα γενιά κύκλων συμπληρώνει τον εναπομείναντα χώρο των προηγούμενων. Σε κάθε επίπεδο αναδρομής, προσθέτουμε μικρότερους κύκλους σε τριπλέτες κυρίων κύκλων, δημιουργώντας ένα περίπλοκο, γεμάτο μοτίβο με αλληλοεπικαλυπτόμενους κύκλους.

Ακολουθεί το πρόγραμμα για τη σχεδίαση του **Apollonian Gasket** χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
```

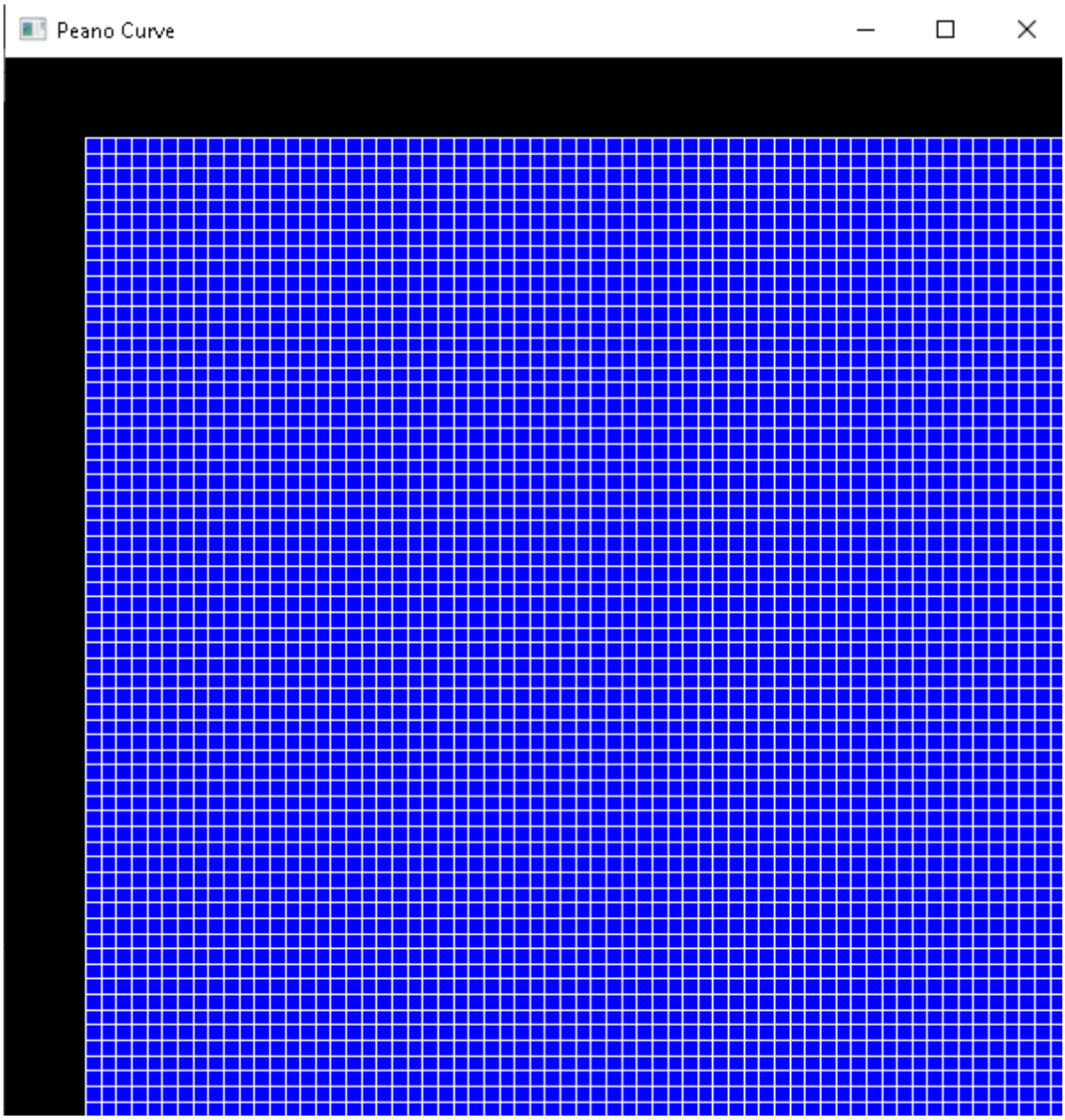
```
// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;  // Πλάτος παραθύρου
const int windowHeight = 600; // Ύψος παραθύρου

// Ορισμός του μέγιστου βάθους αναδρομής για το Apollonian Gasket
const int max_depth = 5;  // Ο αριθμός των επαναλήψεων για το βάθος του fractal

/**
 * Συνάρτηση που σχεδιάζει έναν κύκλο με καθορισμένη ακτίνα και θέση
 * @param x Η x-συντεταγμένη του κέντρου του κύκλου
 * @param y Η y-συντεταγμένη του κέντρου του κύκλου
 * @param radius Η ακτίνα του κύκλου
 * @param br Το πινέλο (Brush) που χρησιμοποιείται για το χρώμα του κύκλου
 */
void drawCircle(float x, float y, float radius, graphics::Brush& br) {
    graphics::drawDisk(x, y, radius, br);
}

/**
 * Αναδρομική συνάρτηση που δημιουργεί το Apollonian Gasket fractal
 * @param x Η x-συντεταγμένη του κέντρου του κύκλου
 * @param y Η y-συντεταγμένη του κέντρου του κύκλου
 * @param radius Η ακτίνα του τρέχοντος κύκλου
 * @param depth Το τρέχον βάθος αναδρομής. Η συνάρτηση σταματά όταν depth == 0.
 */
void apollonianGasket(float x, float y, float radius, int depth) {
    if (depth == 0 || radius < 1.0f) {  // Όριο για το βάθος και το μέγεθος του κύκλου
        return;
    }

    // Δημιουργία πινέλου για τον κύκλο
    graphics::Brush br;
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.8f; // Προσαρμογή
χρώματος
    br.fill_color[1] = 0.2f;
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.8f;

    // Σχεδίαση του κεντρικού κύκλου
    drawCircle(x, y, radius, br);

    // Υπολογισμός ακτίνας για τους επόμενους μικρότερους κύκλους
    float newRadius = radius / 2.0f;

    // Αναδρομική κλήση για τους τρεις νέους κύκλους σε διάφορες θέσεις
    apollonianGasket(x - newRadius, y, newRadius, depth - 1);        // Αριστερά
    apollonianGasket(x + newRadius, y, newRadius, depth - 1);        // Δεξιά
    apollonianGasket(x, y - newRadius, newRadius, depth - 1);        // Κάτω
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);  // Ορισμός μαύρου φόντου

    // Ορισμός κεντρικής θέσης και αρχικής ακτίνας για το Apollonian Gasket
    float centerX = windowWidth / 2.0f;
    float centerY = windowHeight / 2.0f;
    float initialRadius = 300.0f;  // Ακτίνα για τον πρώτο κύκλο

    // Κλήση της αναδρομικής συνάρτησης για να σχεδιάσει το Apollonian Gasket
    apollonianGasket(centerX, centerY, initialRadius, max_depth);
}
```

```
int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Apollonian Gasket");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του κύριου βρόχου της βιβλιοθήκης για τη διαχείριση παραθύρου
    graphics::startMessageLoop();

    // Καθαρισμός πόρων και καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει τον μέγιστο αριθμό αναδρομών για το Apollonian Gasket. Όσο μεγαλύτερο το βάθος, τόσο περισσότεροι μικρότεροι κύκλοι θα σχεδιαστούν.
2. **Συνάρτηση** `drawCircle()`:
    - Αυτή η συνάρτηση σχεδιάζει έναν κύκλο στο σημείο (x,y)(x, y)(x,y) με ακτίνα `radius` και χρώμα από το αντικείμενο `Brush`.
3. **Συνάρτηση** `apollonianGasket()`:
    - Η αναδρομική αυτή συνάρτηση σχεδιάζει το Apollonian Gasket.
    - Αν το `depth` είναι 0 ή η ακτίνα είναι μικρότερη από 1, η συνάρτηση σταματά για να αποφύγει την αναπαραγωγή μικροσκοπικών κύκλων.
    - Σχεδιάζει έναν κύκλο στο κέντρο του παραθύρου με συγκεκριμένη ακτίνα και χρώμα που αλλάζει ελαφρά σε κάθε επίπεδο βάθους.
    - Υπολογίζεται η ακτίνα για τους επόμενους μικρότερους κύκλους (το μισό του αρχικού κύκλου).
    - Καλείται αναδρομικά για τους τρεις εσωτερικούς κύκλους, τοποθετημένους γύρω από τον κεντρικό.
4. **Συνάρτηση** `draw()`:
    καλεί τη `apollonianGasket()` για να σχεδιάσει το fractal, ξεκινώντας από τον κεντρικό κύκλο στο κέντρο του παραθύρου.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Peano Curve



Η **Peano Curve** είναι ένα από τα πιο κλασικά παραδείγματα καμπυλών που γεμίζουν το επίπεδο. Η καμπύλη αυτή σχεδιάζεται αναδρομικά και διατρέχει ολόκληρη την περιοχή, γεμίζοντας το επίπεδο σε ένα τετράγωνο πλαίσιο.

Ακολουθεί το πρόγραμμα για τη σχεδίαση της **Peano Curve** χρησιμοποιώντας τη βιβλιοθήκη SGG, με αναλυτικά σχόλια .

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός των διαστάσεων του παραθύρου
```

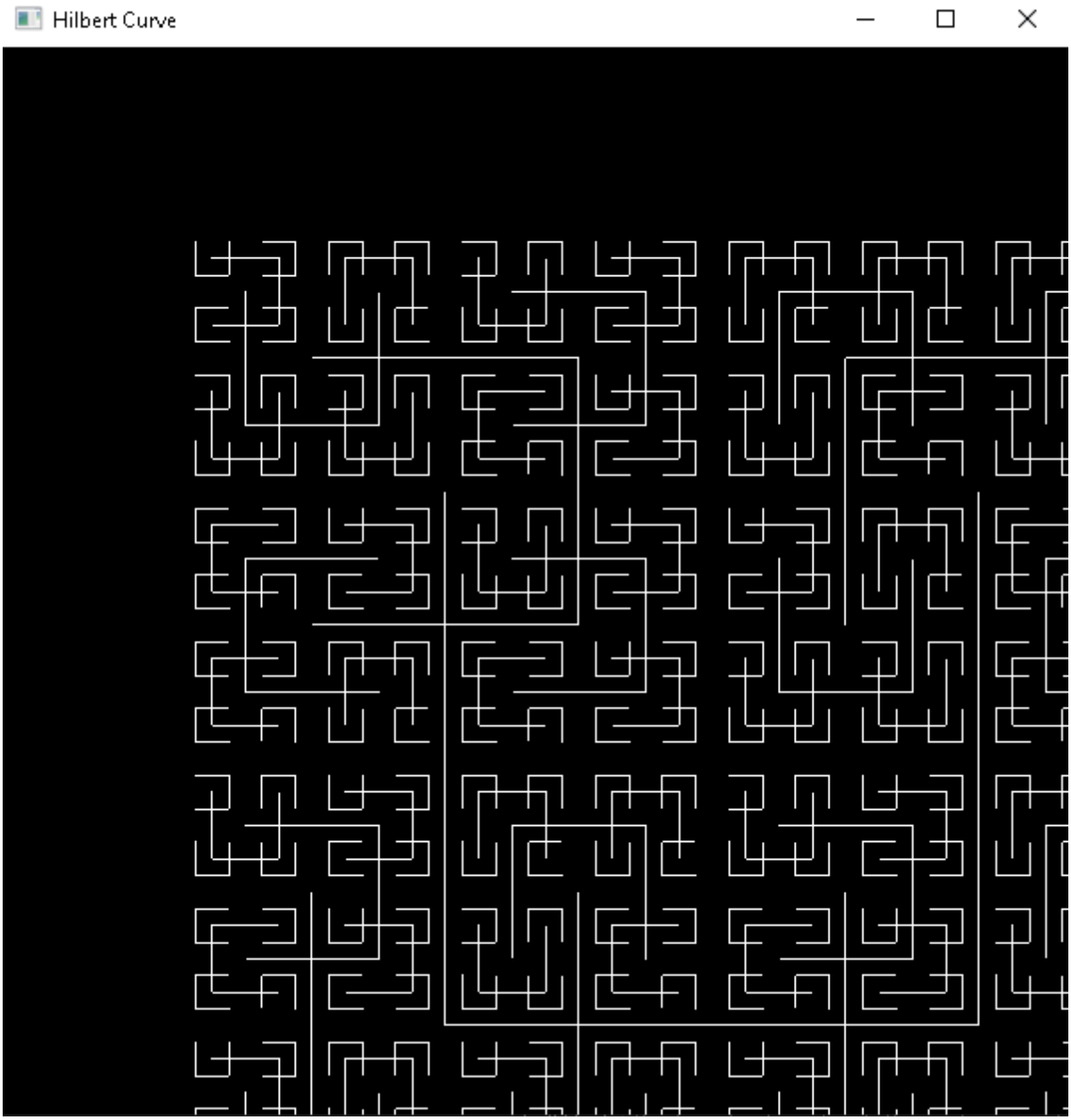
```
const int windowWidth = 600;   // Πλάτος παραθύρου σε pixel
const int windowHeight = 600;  // Ύψος παραθύρου σε pixel

// Μέγιστο βάθος αναδρομής για τη δημιουργία της Peano Curve
const int max_depth = 4;  // Μπορεί να αυξηθεί για περισσότερη λεπτομέρεια

/**
 * Αναδρομική συνάρτηση για σχεδίαση της Peano Curve
 * @param x Η x-συντεταγμένη για την αρχική θέση σχεδίασης
 * @param y Η y-συντεταγμένη για την αρχική θέση σχεδίασης
 * @param length Το μήκος της τρέχουσας πλευράς του τετραγώνου
 * @param depth Το τρέχον βάθος αναδρομής
 * @param orientation Προσανατολισμός της σχεδίασης (δεν χρησιμοποιείται εδώ)
 *
 * @details Αυτή η συνάρτηση σχεδιάζει τη Peano Curve με αναδρομή. Σε κάθε επίπεδο
 * βάθους, το τετράγωνο διαιρείται σε εννέα μικρότερα τετράγωνα και σχεδιάζεται
 * γραμμικά με μπλε χρώμα.
 */
void drawPeanoCurve(float x, float y, float length, int depth, int orientation) {
    if (depth == 0) {
        // Σχεδιάζουμε ένα τετράγωνο όταν φτάσουμε στο μέγιστο βάθος
        graphics::Brush br;
        br.fill_color[0] = 0.0f;
        br.fill_color[1] = 0.0f;
        br.fill_color[2] = 1.0f;  // Μπλε χρώμα για τη γραμμή
        graphics::drawRect(x, y, length, length, br);
    }
    else {
        // Υπολογισμός του νέου μήκους για το επόμενο επίπεδο
        float newLength = length / 3.0f;

        // Αναδρομική κλήση για κάθε μικρό τετράγωνο σύμφωνα με το μοτίβο Peano
        drawPeanoCurve(x, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + newLength, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + 2 * newLength, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + 2 * newLength, y + newLength, newLength, depth - 1,
orientation);
        drawPeanoCurve(x + newLength, y + newLength, newLength, depth - 1,
orientation);
        drawPeanoCurve(x, y + newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x, y + 2 * newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x + newLength, y + 2 * newLength, newLength, depth - 1,
orientation);
        drawPeanoCurve(x + 2 * newLength, y + 2 * newLength, newLength, depth - 1,
orientation);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
 *
 * @details Η συνάρτηση αυτή καλείται αυτόματα από τη βιβλιοθήκη και εκκινεί
 * τη σχεδίαση της Peano Curve από την αρχική θέση στην οθόνη.
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);  // Ορισμός μαύρου φόντου

    // Αρχικές συντεταγμένες και μήκος για την Peano Curve
    float startX = 50.0f;    // x-συντεταγμένη εκκίνησης
    float startY = 50.0f;    // y-συντεταγμένη εκκίνησης
    float startLength = 700.0f;  // Αρχικό μήκος πλευράς του τετραγώνου

    // Κλήση της συνάρτησης για σχεδίαση της Peano Curve
    drawPeanoCurve(startX, startY, startLength, max_depth, 0);
```

```
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Peano Curve");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων για τη διαχείριση παραθύρου
    graphics::startMessageLoop();

    // Καταστροφή του παραθύρου μετά την έξοδο και καθαρισμός πόρων
    graphics::destroyWindow();

    return 0;
}
```

Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει το μέγιστο επίπεδο αναδρομής. Όσο μεγαλύτερο το βάθος, τόσο περισσότερη λεπτομέρεια θα έχει η Peano Curve.
2. **Συνάρτηση** `drawPeanoCurve()`:
    - Η συνάρτηση αυτή χρησιμοποιεί αναδρομή για να σχεδιάσει την Peano Curve.
    - Αν το `depth` είναι 0, σχεδιάζει ένα τετραγωνάκι στο σημείο (x,y)(x, y)(x,y) με μήκος `length`.
    - Αν το `depth` δεν είναι 0, διαιρεί το μήκος σε τρία μέρη (`newLength`) και σχεδιάζει την Peano Curve σε εννέα μικρότερα τετράγωνα, σύμφωνα με τη δομή της καμπύλης, καλώντας την ίδια συνάρτηση αναδρομικά για κάθε τμήμα.
3. **Συνάρτηση** `draw()`:
    - Η `draw()` είναι η κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
    - καλεί τη `drawPeanoCurve()` για να σχεδιάσει την καμπύλη Peano από το κέντρο του παραθύρου.
4. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Hilbert Curve



Η **Hilbert Curve** είναι μια καμπύλη γεμίσματος του επιπέδου, η οποία περνά από όλα τα σημεία ενός τετραγώνου χωρίς να διασταυρώνεται με τον εαυτό της. Αυτή η καμπύλη δημιουργείται αναδρομικά και χαρακτηρίζεται από τις συνεχείς στροφές που την οδηγούν να γεμίσει το χώρο.

Ακολουθεί το πρόγραμμα για τη σχεδίαση της **Hilbert Curve** χρησιμοποιώντας τη βιβλιοθήκη SGG, με αναλυτικά σχόλια .

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;   // Πλάτος παραθύρου σε pixel
const int windowHeight = 600;  // Ύψος παραθύρου σε pixel

// Μέγιστο βάθος αναδρομής για τη δημιουργία της Hilbert Curve
```

```
const int max_depth = 5;  // Αυξάνεται για περισσότερη λεπτομέρεια στη σχεδίαση

/**
 * Αναδρομική συνάρτηση για σχεδίαση της Hilbert Curve
 * @param x Η x-συντεταγμένη της αρχικής θέσης σχεδίασης
 * @param y Η y-συντεταγμένη της αρχικής θέσης σχεδίασης
 * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου
 * @param depth Το τρέχον επίπεδο αναδρομής
 * @param rotation Η τρέχουσα κατεύθυνση περιστροφής (0, 1, 2, ή 3)
 *
 * @details Αυτή η συνάρτηση σχεδιάζει αναδρομικά την Hilbert Curve. Ανάλογα με το
`rotation`,
 * κάθε κλήση της συνάρτησης διαμορφώνει το μοτίβο σε διαφορετικό προσανατολισμό για
να
 * πετύχει τη διαδρομή χωρίς επικάλυψη.
 */
void drawHilbertCurve(float x, float y, float size, int depth, int rotation) {
    graphics::Brush br;

    if (depth == 0) {
        return; // Σταματάμε όταν φτάσουμε στο μέγιστο βάθος
    }

    // Υπολογισμός του νέου μεγέθους για την επόμενη αναδρομή
    float newSize = size / 2.0f;

    // Αναδρομικές κλήσεις για κάθε γωνία περιστροφής
    if (rotation == 0) {
        drawHilbertCurve(x, y, newSize, depth - 1, 1);
        graphics::drawLine(x + newSize / 2, y + newSize / 2, x + newSize / 2, y + 3 *
newSize / 2, br);
        drawHilbertCurve(x, y + newSize, newSize, depth - 1, 0);
        graphics::drawLine(x + newSize / 2, y + 3 * newSize / 2, x + 3 * newSize / 2,
y + 3 * newSize / 2, br);
        drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 0);
        graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + 3 * newSize /
2, y + newSize / 2, br);
        drawHilbertCurve(x + newSize, y, newSize, depth - 1, 3);
    }
    else if (rotation == 1) {
        drawHilbertCurve(x, y, newSize, depth - 1, 0);
        graphics::drawLine(x + newSize / 2, y + newSize / 2, x + 3 * newSize / 2, y +
newSize / 2, br);
        drawHilbertCurve(x + newSize, y, newSize, depth - 1, 1);
        graphics::drawLine(x + 3 * newSize / 2, y + newSize / 2, x + 3 * newSize / 2,
y + 3 * newSize / 2, br);
        drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 1);
        graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + newSize / 2,
y + 3 * newSize / 2, br);
        drawHilbertCurve(x, y + newSize, newSize, depth - 1, 2);
    }
    else if (rotation == 2) {
        drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 3);
        graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + newSize / 2,
y + 3 * newSize / 2, br);
        drawHilbertCurve(x, y + newSize, newSize, depth - 1, 2);
        graphics::drawLine(x + newSize / 2, y + 3 * newSize / 2, x + newSize / 2, y +
newSize / 2, br);
        drawHilbertCurve(x, y, newSize, depth - 1, 2);
        graphics::drawLine(x + newSize / 2, y + newSize / 2, x + 3 * newSize / 2, y +
newSize / 2, br);
        drawHilbertCurve(x + newSize, y, newSize, depth - 1, 1);
    }
    else if (rotation == 3) {
        drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 2);
```

```
        graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + 3 * newSize / 2, y + newSize / 2, br);
        drawHilbertCurve(x + newSize, y, newSize, depth - 1, 3);
        graphics::drawLine(x + 3 * newSize / 2, y + newSize / 2, x + newSize / 2, y + newSize / 2, br);
        drawHilbertCurve(x, y, newSize, depth - 1, 3);
        graphics::drawLine(x + newSize / 2, y + newSize / 2, x + newSize / 2, y + 3 * newSize / 2, br);
        drawHilbertCurve(x, y + newSize, newSize, depth - 1, 0);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
 *
 * @details Αυτή η συνάρτηση ξεκινάει τη σχεδίαση της Hilbert Curve από τη
 * θέση `(startX, startY)` και καλεί τη συνάρτηση `drawHilbertCurve` με τις
 * αρχικές παραμέτρους.
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);  // Μαύρο φόντο για αντίθεση

    // Αρχικές συντεταγμένες και μέγεθος για την Hilbert Curve
    float startX = 100.0f;    // x-συντεταγμένη εκκίνησης
    float startY = 100.0f;    // y-συντεταγμένη εκκίνησης
    float startLength = 600.0f;  // Μέγεθος της πλευράς του πρώτου τετραγώνου

    // Κλήση της συνάρτησης για σχεδίαση της Hilbert Curve
    drawHilbertCurve(startX, startY, startLength, max_depth, 0);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Hilbert Curve");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων για τη διαχείριση παραθύρου
    graphics::startMessageLoop();

    // Καταστροφή του παραθύρου μετά την έξοδο και καθαρισμός πόρων
    graphics::destroyWindow();

    return 0;
}
```

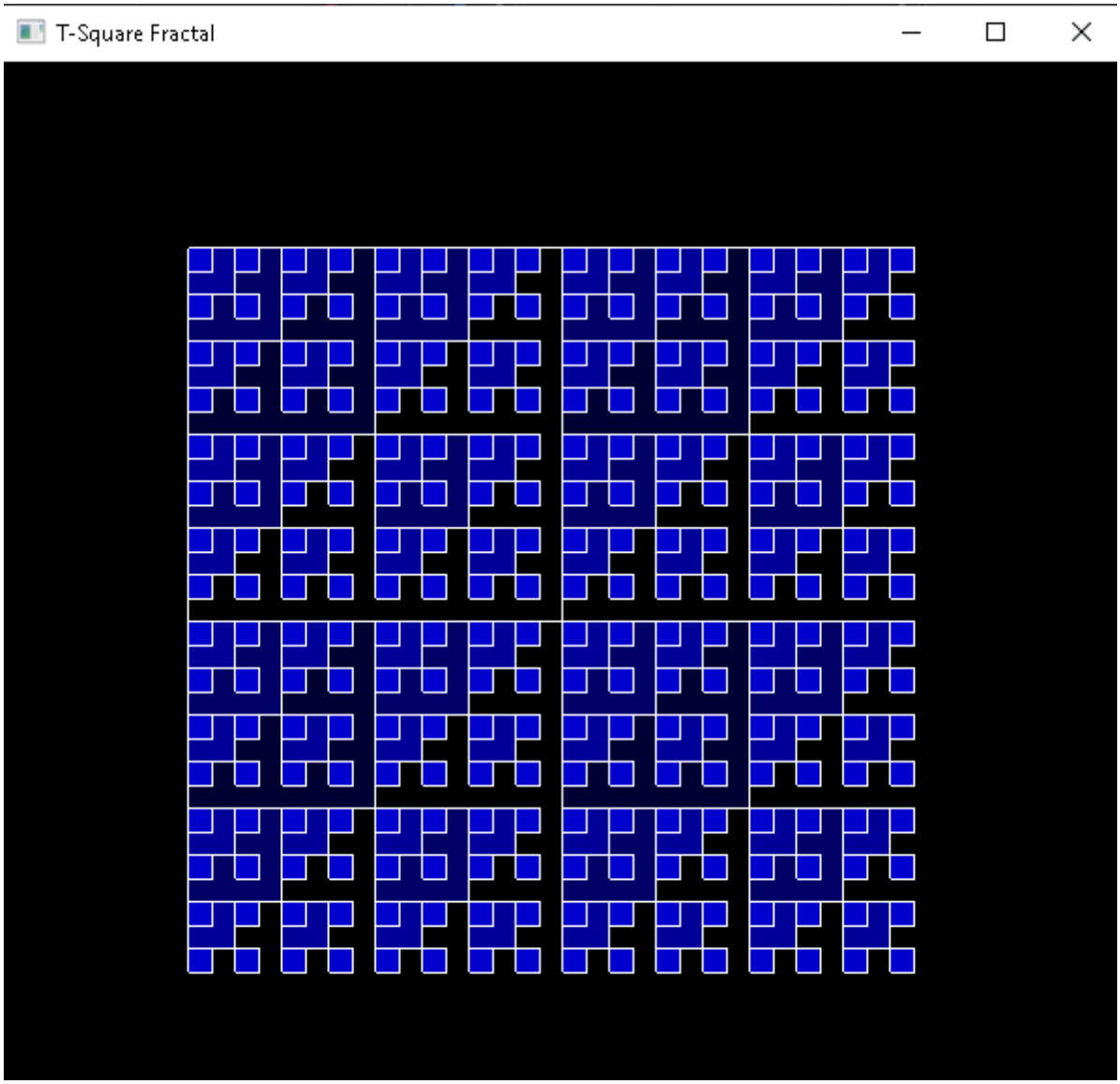
Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει το μέγιστο επίπεδο της αναδρομής για την καμπύλη Hilbert. Μεγαλύτερο βάθος σημαίνει περισσότερες λεπτομέρειες στην καμπύλη.
2. **Συνάρτηση** `drawHilbertCurve()`:
    - Αυτή η αναδρομική συνάρτηση σχεδιάζει την καμπύλη Hilbert. Αν το `depth` είναι 0, η συνάρτηση τερματίζει.

- Διαφορετικά, διαιρεί το τετράγωνο σε τέσσερα μικρότερα τμήματα και καθορίζει την κατεύθυνση και τη σειρά των αναδρομικών κλήσεων σύμφωνα με την τρέχουσα περιστροφή (`rotation`).
- Η `drawLine()` χρησιμοποιείται για τη σύνδεση των τετραγώνων μεταξύ των κλήσεων, σχηματίζοντας την καμπύλη.

3. **Συνάρτηση** `draw()`:
    - Η `draw()` ορίζεται από τη βιβλιοθήκη SGG και καλείται για να σχεδιάσει το περιεχόμενο του παραθύρου.
    - ξεκινά τη σχεδίαση της Hilbert Curve από το σημείο `(startX, startY)` με καθορισμένο μήκος και αρχική περιστροφή.
4. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Ο κύριος βρόχος ξεκινά με την `startMessageLoop()`, κρατώντας το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

T-Square Fractals



Το **T-Square Fractal** είναι ένα απλό και ενδιαφέρον fractal που ξεκινά με ένα τετράγωνο και σε κάθε επίπεδο αναδρομής προσθέτει τέσσερα μικρότερα τετράγωνα στις κορυφές του. Αυτά τα νέα τετράγωνα συνεχίζουν να επαναλαμβάνουν την ίδια διαδικασία, δημιουργώντας ένα γεωμετρικό μοτίβο σε σχήμα "T".

Παρακάτω είναι το πρόγραμμα για τη σχεδίαση του **T-Square Fractal** χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός των διαστάσεων του παραθύρου
const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
const int windowHeight = 600; // Ύψος του παραθύρου σε pixels

// Ορισμός του μέγιστου βάθους για τη σχεδίαση του fractal
```

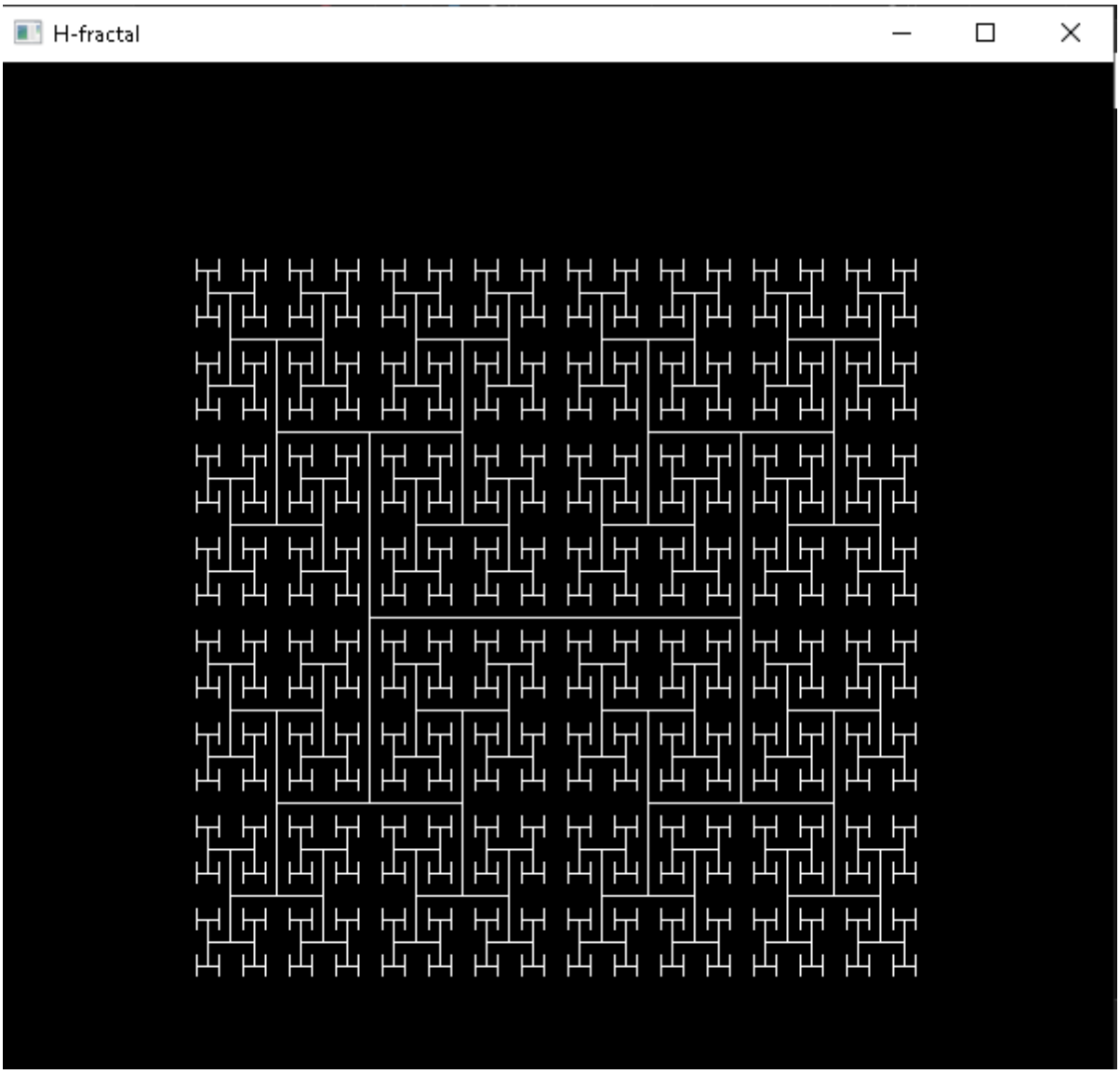
```
const int max_depth = 5; // Αύξηση του βάθους δημιουργεί περισσότερες λεπτομέρειες

/**
 * Συνάρτηση που σχεδιάζει ένα τετράγωνο στο κέντρο μιας δεδομένης θέσης
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Το μέγεθος της πλευράς του τετραγώνου
 * @param br Το `graphics::Brush` που καθορίζει το χρώμα και το στυλ του τετραγώνου
 */
void drawSquare(float x, float y, float size, graphics::Brush& br) {
    graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του T-Square Fractal
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του τετραγώνου
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του τετραγώνου
 * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου
 * @param depth Το επίπεδο βάθους της αναδρομής
 *
 * @details Η συνάρτηση καλείται για κάθε τετράγωνο με μειωμένο `size` και βάθος,
σχεδιάζοντας
 * νέα τετράγωνα σε κάθε γωνία του προηγούμενου τετραγώνου, σχηματίζοντας τη μορφή του
fractal.
 */
void tSquareFractal(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
    }

    // Ορισμός του χρώματος για το τρέχον τετράγωνο
    graphics::Brush br;
    br.fill_color[0] = 0.0f;                          // Red
    br.fill_color[1] = 0.0f;                          // Green
    br.fill_color[2] = 1.0f - (depth / (float)max_depth); // Μείωση του μπλε με το
βάθος

    // Σχεδίαση του κεντρικού τετραγώνου
    drawSquare(x, y, size, br);

    // Υπολογισμός του μεγέθους των επόμενων τετραγώνων
    float newSize = size / 2.0f;

    // Αναδρομική κλήση για τα τέσσερα τετράγωνα που σχηματίζουν το "T"
    tSquareFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
    tSquareFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
    tSquareFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
    tSquareFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το T-Square Fractal στο κέντρο.
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background);  // Μαύρο φόντο για αντίθεση

    // Αρχικές συντεταγμένες και μέγεθος για το κεντρικό τετράγωνο
    float startX = windowWidth / 2.0f; // Κεντρική x-συντεταγμένη
    float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
    float startSize = 200.0f; // Αρχικό μέγεθος για το πρώτο τετράγωνο

    // Κλήση της συνάρτησης για σχεδίαση του T-Square Fractal
```

```
    tSquareFractal(startX, startY, startSize, max_depth);
}

int main() {
    // Δημιουργία παραθύρου με τις καθορισμένες διαστάσεις
    graphics::createWindow(windowWidth, windowHeight, "T-Square Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου για τη διαχείριση του παραθύρου και την ανανέωση
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου και απελευθέρωση πόρων μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει το μέγιστο βάθος αναδρομής για το fractal. Όσο μεγαλύτερο το βάθος, τόσο περισσότερα τετράγωνα θα δημιουργηθούν.
2. **Συνάρτηση** `drawSquare()`:
    - Αυτή η συνάρτηση σχεδιάζει ένα τετράγωνο στο σημείο (x,y)(x, y)(x,y) με μέγεθος `size` και χρώμα που ορίζεται από το αντικείμενο `Brush`.
3. **Συνάρτηση** `tSquareFractal()`:
    - Αυτή η συνάρτηση είναι αναδρομική και δημιουργεί το T-Square Fractal.
    - Αν το `depth` είναι 0, σταματάμε τη διαδικασία αναδρομής.
    - Σχεδιάζουμε το κεντρικό τετράγωνο στο σημείο (x,y)(x, y)(x,y) με το καθορισμένο μέγεθος `size`.
    - Υπολογίζουμε το μέγεθος για τα επόμενα τετράγωνα (`newSize`), το οποίο είναι το μισό του τρέχοντος τετραγώνου.
    - Καλούμε αναδρομικά τη `tSquareFractal()` για τέσσερα νέα τετράγωνα, που τοποθετούνται στις γωνίες του αρχικού τετραγώνου.
4. **Συνάρτηση** `draw()`:
    - Ξεκινά τη σχεδίαση του T-Square Fractal, καλώντας τη `tSquareFractal()` από το κέντρο του παραθύρου και με αρχικό μέγεθος τετραγώνου 200 pixels.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Το παράθυρο διατηρείται ενεργό μέσω της `startMessageLoop()` μέχρι να το κλείσει ο χρήστης.

H-Fractal



Το **H-fractal** είναι ένα fractal που αποτελείται από επαναλαμβανόμενα μοτίβα σε σχήμα "H". Ξεκινά από έναν κεντρικό "H" και σε κάθε επίπεδο αναδρομής προστίθενται μικρότερα σχήματα "H" στις άκρες των γραμμών, δημιουργώντας έτσι μια γεωμετρική συμμετρία.

Ακολουθεί το πρόγραμμα για τη σχεδίαση του **H-fractal** χρησιμοποιώντας τη βιβλιοθήκη SGG, με αναλυτικά σχόλια .

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
const int windowHeight = 600; // Ύψος του παραθύρου σε pixels

// Ορισμός του μέγιστου βάθους αναδρομής για το H-fractal
const int max_depth = 5; // Αύξηση του βάθους δημιουργεί περισσότερες λεπτομέρειες
```

```
/**
 * Συνάρτηση που σχεδιάζει το γράμμα "Η"
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του "Η"
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του "Η"
 * @param size Το μήκος της κάθε γραμμής του "Η"
 * @param br Το `graphics::Brush` που καθορίζει το χρώμα και το στυλ του "Η"
 */
void drawH(float x, float y, float size, graphics::Brush& br) {
    float half = size / 2.0f;

    // Σχεδιάζουμε την οριζόντια γραμμή του "Η"
    graphics::drawLine(x - half, y, x + half, y, br);

    // Σχεδιάζουμε τις δύο κάθετες γραμμές του "Η"
    graphics::drawLine(x - half, y - half, x - half, y + half, br);
    graphics::drawLine(x + half, y - half, x + half, y + half, br);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του H-fractal
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του "Η"
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του "Η"
 * @param size Το μήκος της κάθε γραμμής του "Η"
 * @param depth Το επίπεδο βάθους της αναδρομής
 *
 * @details Σε κάθε επίπεδο, η συνάρτηση καλείται για τέσσερα νέα "Η" που
τοποθετούνται
 * στις τέσσερις άκρες του προηγούμενου, μειώνοντας το μέγεθός τους κατά το ήμισυ.
 */
void hFractal(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
    }

    // Ορισμός του χρώματος για το τρέχον "Η"
    graphics::Brush br;
    br.fill_color[0] = 0.0f;                          // Red
    br.fill_color[1] = 0.0f;                          // Green
    br.fill_color[2] = 1.0f - (depth / (float)max_depth);  // Μείωση του μπλε καθώς
αυξάνεται το βάθος

    // Σχεδίαση του κεντρικού "Η" με τις καθορισμένες παραμέτρους
    drawH(x, y, size, br);

    // Υπολογισμός του νέου μεγέθους για τα επόμενα "Η"
    float newSize = size / 2.0f;

    // Αναδρομική κλήση για τα τέσσερα "Η" στις γωνίες
    hFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
    hFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
    hFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
    hFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το H-fractal ξεκινώντας από το κέντρο.
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background); // Ορισμός του μαύρου φόντου

    // Αρχικές συντεταγμένες και μέγεθος για το πρώτο κεντρικό "Η"
```

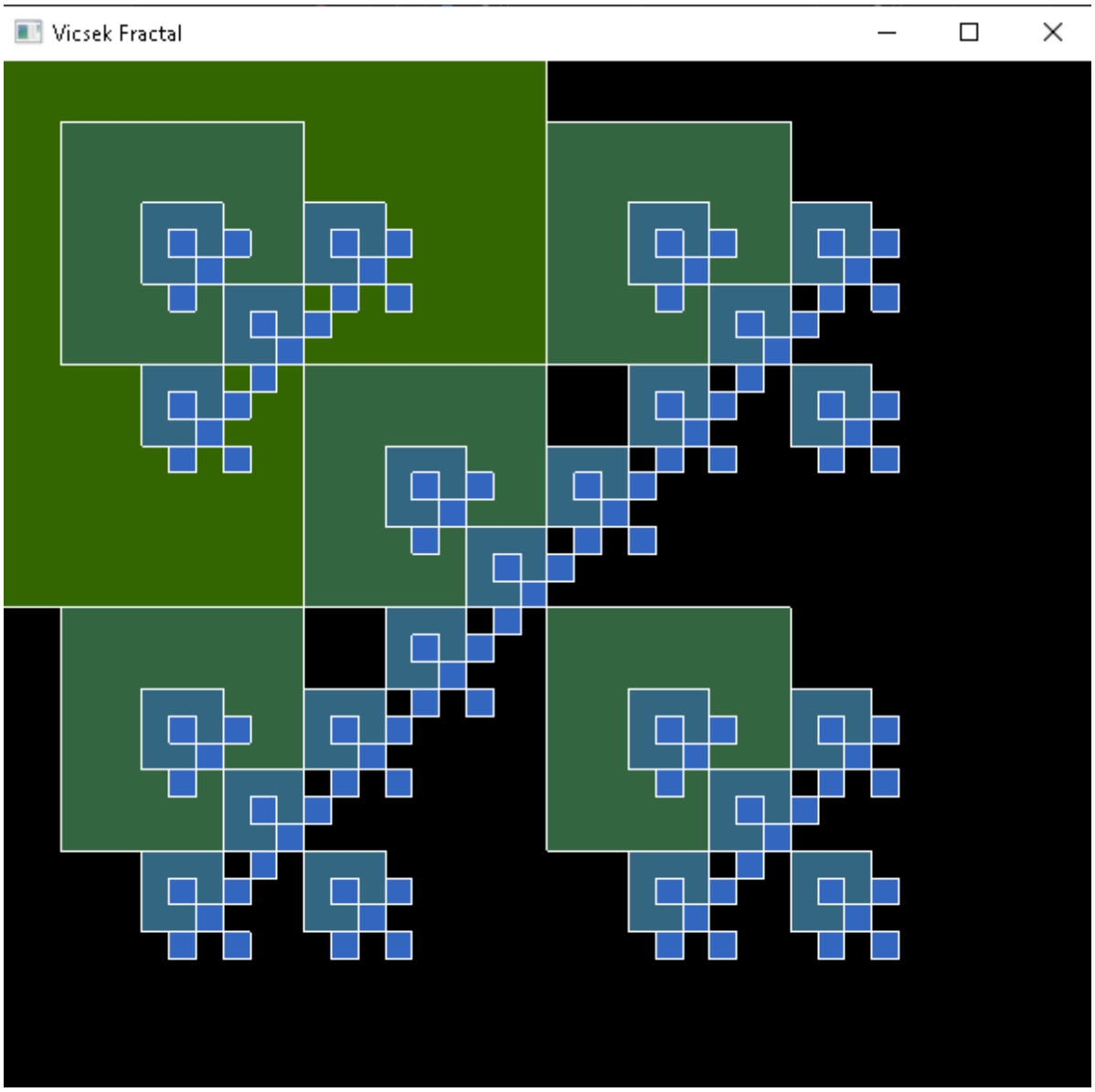
```
    float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
    float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
    float startSize = 200.0f; // Αρχικό μέγεθος του "H"

    // Κλήση της συνάρτησης για σχεδίαση του H-fractal
    hFractal(startX, startY, startSize, max_depth);
}

int main() {
    // Δημιουργία παραθύρου με τις καθορισμένες διαστάσεις
    graphics::createWindow(windowWidth, windowHeight, "H-fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου για τη διαχείριση του παραθύρου και την ανανέωση
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου και απελευθέρωση πόρων μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει το μέγιστο βάθος αναδρομής για το fractal. Όσο μεγαλύτερο το βάθος, τόσο περισσότερα "H" θα σχεδιαστούν.
2. **Συνάρτηση** `drawH()`:
    - Αυτή η συνάρτηση σχεδιάζει ένα "H" στο σημείο (x,y)(x, y)(x,y) με καθορισμένο μέγεθος `size` και χρώμα `Brush`.
    - Χρησιμοποιεί τη `drawLine()` για να σχεδιάσει την οριζόντια και τις δύο κάθετες γραμμές του "H".
3. **Συνάρτηση** `hFractal()`:
    - Η συνάρτηση αυτή είναι αναδρομική και δημιουργεί το H-fractal.
    - Αν το `depth` είναι 0, η διαδικασία σταματά.
    - Σχεδιάζει το κεντρικό "H" στο σημείο (x,y)(x, y)(x,y) με το καθορισμένο μέγεθος `size`.
    - Υπολογίζει το μέγεθος των επόμενων "H" (`newSize`), που είναι το μισό του αρχικού "H".
    - Καλεί τη `hFractal()` αναδρομικά για τα τέσσερα νέα "H" που τοποθετούνται στις τέσσερις γωνίες του κεντρικού "H".
4. **Συνάρτηση** `draw()`:
    - Ξεκινά τη σχεδίαση του H-fractal από το κέντρο του παραθύρου και με αρχικό μέγεθος 200 pixels.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.

- Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Vicsek Fractals



Το **Vicsek Fractal** είναι ένα συμμετρικό fractal που δημιουργείται με την αναπαραγωγή ενός βασικού μοτίβου. Ξεκινά από ένα κεντρικό τετράγωνο, το οποίο χωρίζεται σε εννέα ίσα τμήματα και γεμίζονται μόνο τα πέντε, δηλαδή το κεντρικό και τα τέσσερα στις γωνίες. Αυτό το μοτίβο επαναλαμβάνεται σε κάθε επίπεδο αναδρομής.

Ακολουθεί το πρόγραμμα για τη σχεδίαση του **Vicsek Fractal** χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
const int windowHeight = 600; // Ύψος του παραθύρου σε pixels
```

```
// Μέγιστο βάθος αναδρομής για το Vicsek Fractal
const int max_depth = 4; // Αύξηση του βάθους αυξάνει τις λεπτομέρειες του fractal

/**
 * Συνάρτηση που σχεδιάζει ένα τετράγωνο
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Η πλευρά του τετραγώνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα και το στυλ του τετραγώνου
 */
void drawSquare(float x, float y, float size, graphics::Brush& br) {
    graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
}

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Vicsek Fractal
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Η πλευρά του τετραγώνου
 * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
 *
 * @details Η συνάρτηση χωρίζει το τετράγωνο σε πέντε μικρότερα, τοποθετώντας τα τέσσερα στις γωνίες και ένα στο κέντρο.
 * Καλείται αναδρομικά για κάθε υποτετράγωνο, μειώνοντας το `depth` κατά 1 μέχρι να φτάσει στο `0`.
 */
void vicsekFractal(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός της αναδρομής στο βάθος 0
    }

    // Ορισμός του χρώματος για το τρέχον τετράγωνο, προσαρμόζοντας τη φωτεινότητα με το βάθος
    graphics::Brush br;
    br.fill_color[0] = 0.2f;                          // Red component
    br.fill_color[1] = 0.4f;                          // Green component
    br.fill_color[2] = 1.0f - (depth / (float)max_depth);  // Blue component μειώνεται όσο αυξάνεται το βάθος

    // Σχεδίαση του κεντρικού τετραγώνου με τις καθορισμένες παραμέτρους
    drawSquare(x, y, size, br);

    // Υπολογισμός του νέου μεγέθους για τα υποτετράγωνα στο επόμενο επίπεδο
    float newSize = size / 3.0f;

    // Αναδρομική κλήση για τα τέσσερα τετράγωνα στις γωνίες και το κεντρικό τετράγωνο
    vicsekFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
    vicsekFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
    vicsekFractal(x, y, newSize, depth - 1);                     // Κεντρικό
    vicsekFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
    vicsekFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το Vicsek Fractal ξεκινώντας από το κέντρο.
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background); // Μαύρο φόντο για καλύτερη αντίθεση

    // Αρχικές συντεταγμένες και μέγεθος για το πρώτο κεντρικό τετράγωνο
    float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
```

```
    float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
    float startSize = 400.0f;           // Αρχικό μέγεθος του τετραγώνου

    // Κλήση της συνάρτησης για σχεδίαση του Vicsek Fractal
    vicsekFractal(startX, startY, startSize, max_depth);
}

int main() {
    // Δημιουργία παραθύρου με τις καθορισμένες διαστάσεις
    graphics::createWindow(windowWidth, windowHeight, "Vicsek Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου για τη διαχείριση του παραθύρου και την ανανέωση
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου και απελευθέρωση πόρων μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει το μέγιστο επίπεδο αναδρομής για το fractal. Όσο μεγαλύτερο το βάθος, τόσο περισσότερα τετράγωνα θα δημιουργηθούν.
2. **Συνάρτηση** `drawSquare()`:
    - Αυτή η συνάρτηση σχεδιάζει ένα τετράγωνο στο σημείο (x,y)(x, y)(x,y) με μέγεθος `size` και χρώμα που ορίζεται από το αντικείμενο `Brush`.
3. **Συνάρτηση** `vicsekFractal()`:
    - Η συνάρτηση αυτή είναι αναδρομική και δημιουργεί το Vicsek Fractal.
    - Αν το `depth` είναι 0, σταματάμε τη διαδικασία αναδρομής.
    - Σχεδιάζουμε το κεντρικό τετράγωνο στο σημείο (x,y)(x, y)(x,y) με το καθορισμένο μέγεθος `size`.
    - Υπολογίζουμε το μέγεθος για τα επόμενα τετράγωνα (`newSize`), το οποίο είναι το ένα τρίτο του τρέχοντος τετραγώνου.
    - Καλούμε αναδρομικά τη `vicsekFractal()` για τα τέσσερα νέα τετράγωνα στις γωνίες και το κεντρικό τετράγωνο.
4. **Συνάρτηση** `draw()`:
    - Ξεκινά τη σχεδίαση του Vicsek Fractal από το κέντρο του παραθύρου με αρχικό μέγεθος 400 pixels.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Menger Sponge

Το **Menger Sponge** είναι ένα τρισδιάστατο fractal που δημιουργείται με την αναδρομική διαίρεση ενός κύβου σε 27 μικρότερους κύβους, όπου αφαιρούνται τα κεντρικά κομμάτια από κάθε πρόσωπο και από το εσωτερικό του κύβου. Για την υλοποίηση αυτού του fractal χρησιμοποιώντας τη βιβλιοθήκη SGG σε δισδιάστατο περιβάλλον, θα σχεδιάσουμε την προβολή του σε 2D (δηλαδή σε επίπεδο), η οποία αποτελείται από τετράγωνα διατάξεως 3x3, στα οποία αφαιρείται το κεντρικό τετράγωνο σε κάθε επίπεδο αναδρομής.

Κώδικας για το Menger Sponge (2D Προβολή)

```
#include "sgg/graphics.h"

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;  // Πλάτος παραθύρου
const int windowHeight = 600; // Ύψος παραθύρου

// Μέγιστο βάθος αναδρομής για το Menger Sponge
const int max_depth = 6; // Μεγαλύτερες τιμές προσφέρουν περισσότερη λεπτομέρεια

/**
```

```
 * Συνάρτηση που σχεδιάζει ένα τετράγωνο
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Το μέγεθος της πλευράς του τετραγώνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα του τετραγώνου
 */
void drawSquare(float x, float y, float size, graphics::Brush& br) {
    // Σχεδίαση τετραγώνου με κεντραρισμένη τη θέση του (x, y)
    graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
}

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Menger Sponge σε 2D
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Το μέγεθος της πλευράς του τετραγώνου
 * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
 *
 * @details Η συνάρτηση τοποθετεί ένα τετράγωνο στο κέντρο και δημιουργεί αναδρομικά
οκτώ μικρότερα
 * τετράγωνα γύρω από αυτό, παραλείποντας το κεντρικό τετράγωνο. Η διαδικασία
επαναλαμβάνεται
 * μέχρι το επίπεδο `depth` να φτάσει στο μηδέν.
 */
void mengerSponge(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός αναδρομής όταν φτάσουμε στο βάθος 0
    }

    // Ορισμός χρώματος για το τρέχον τετράγωνο, με την απόχρωση να εξαρτάται από το
βάθος
    graphics::Brush br;
    br.fill_color[0] = 0.1f + (depth / (float)max_depth) * 0.5f;  // Red component
    br.fill_color[1] = 0.1f;                                      // Green component
    br.fill_color[2] = 0.6f - (depth / (float)max_depth) * 0.5f;  // Blue component

    // Σχεδίαση του κεντρικού τετραγώνου
    drawSquare(x, y, size, br);

    // Υπολογισμός νέου μεγέθους για τα επόμενα μικρότερα τετράγωνα
    float newSize = size / 3.0f;

    // Αναδρομική κλήση για τα οκτώ τετράγωνα που περιβάλλουν το κεντρικό
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            // Παραλείπουμε το κεντρικό τετράγωνο (dx == 0 && dy == 0)
            if (dx != 0 || dy != 0) {
                mengerSponge(x + dx * newSize, y + dy * newSize, newSize, depth - 1);
            }
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το Menger Sponge ξεκινώντας από το
κέντρο
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background); // Μαύρο φόντο για αντίθεση

    // Αρχικές συντεταγμένες και μέγεθος για το κεντρικό τετράγωνο
    float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
```

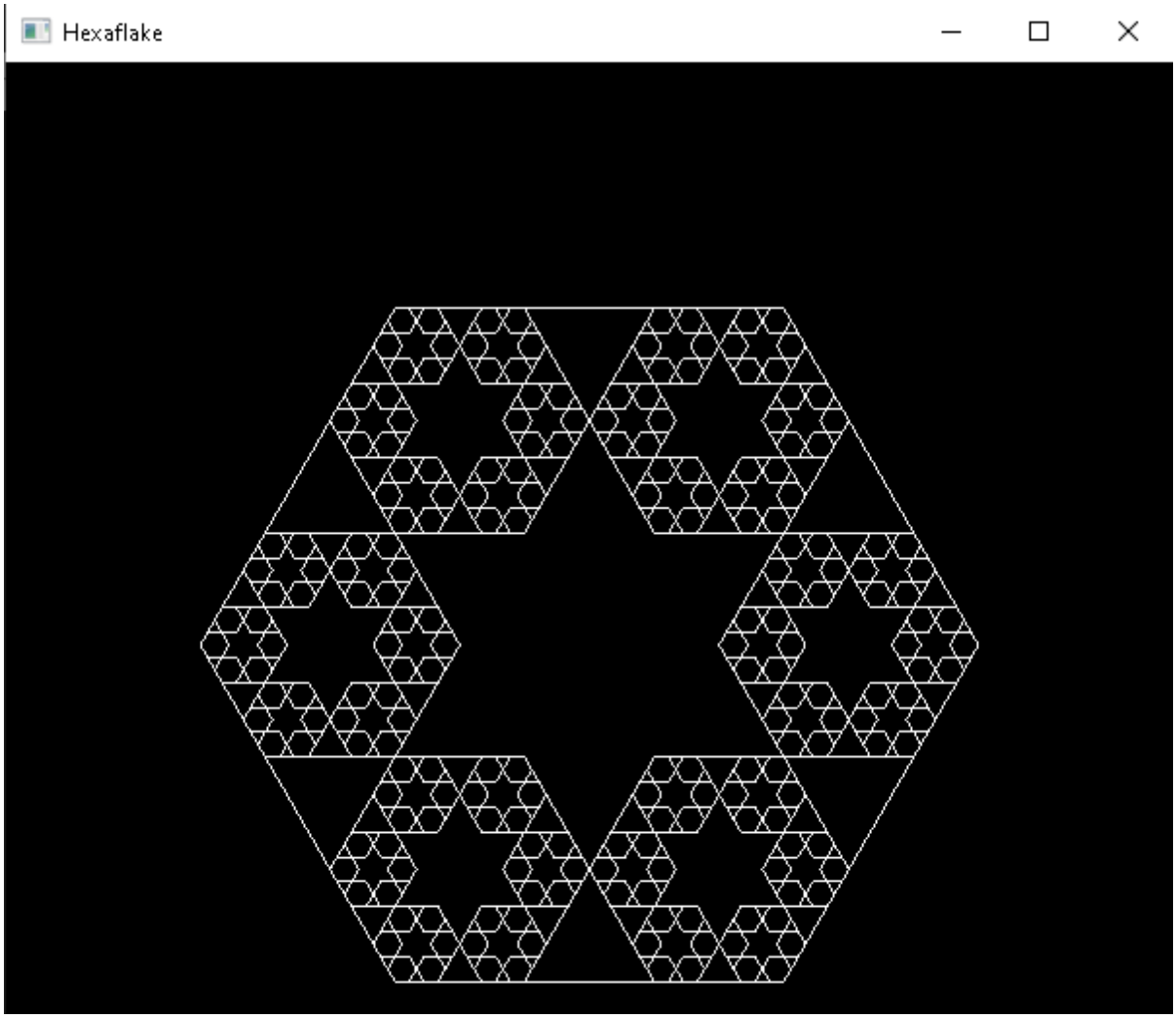
```
    float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
    float startSize = 400.0f;           // Αρχικό μέγεθος του τετραγώνου

    // Κλήση της συνάρτησης για σχεδίαση του Menger Sponge
    mengerSponge(startX, startY, startSize, max_depth);
}

int main() {
    // Δημιουργία παραθύρου με καθορισμένες διαστάσεις
    graphics::createWindow(windowWidth, windowHeight, "Menger Sponge (2D
Projection)");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου για ανανέωση και χειρισμό του παραθύρου
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου και απελευθέρωση πόρων μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Αναλυτική Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει το μέγιστο επίπεδο αναδρομής για το fractal. Όσο μεγαλύτερο το βάθος, τόσο περισσότερα τετράγωνα δημιουργούνται, δημιουργώντας ένα πιο λεπτομερές fractal.
2. **Συνάρτηση** `drawSquare()`:
    - Η συνάρτηση αυτή σχεδιάζει ένα τετράγωνο στο σημείο (x,y)(x, y)(x,y) με μέγεθος `size` και χρώμα που ορίζεται από το αντικείμενο `Brush`.
3. **Συνάρτηση** `mengerSponge()`:
    - Η συνάρτηση αυτή είναι αναδρομική και δημιουργεί το Menger Sponge σε 2D προβολή.
    - Αν το `depth` είναι 0, η διαδικασία αναδρομής σταματά.
    - Σχεδιάζει το κεντρικό τετράγωνο στο σημείο (x,y)(x, y)(x,y) με το καθορισμένο μέγεθος `size`.
    - Υπολογίζεται το μέγεθος για τα επόμενα μικρότερα τετράγωνα (`newSize`), το οποίο είναι το ένα τρίτο του τρέχοντος τετραγώνου.
    - Για κάθε τετράγωνο σε διάταξη 3x3 γύρω από το κεντρικό, καλείται αναδρομικά η `mengerSponge()`, με εξαίρεση το κεντρικό τετράγωνο.
4. **Συνάρτηση** `draw()`:
    - καλεί τη `mengerSponge()` με αρχικές συντεταγμένες, έτσι ώστε το fractal να ξεκινά από το κέντρο του παραθύρου.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.

- Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Hexaflake



Το **Hexaflake** είναι ένα fractal που δημιουργείται με τη διαδοχική διαίρεση ενός εξαγώνου σε μικρότερα εξάγωνα. Ξεκινάμε με ένα αρχικό εξάγωνο και σε κάθε επίπεδο αναδρομής προσθέτουμε έξι μικρότερα εξάγωνα γύρω από το κεντρικό, αφήνοντας ένα κενό στο κέντρο. Το Hexaflake είναι ένα εντυπωσιακό fractal με συμμετρία εξαγώνου.

Ακολουθεί ο κώδικας για τη σχεδίαση του **Hexaflake** χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Σταθερά για το π
#define M_PI 3.14159265358979323846264338327950288 4

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;
const int windowHeight = 600;

// Μέγιστο βάθος αναδρομής για το Hexaflake
const int max_depth = 4;
```

```
/**
 * Συνάρτηση για τον υπολογισμό των κορυφών ενός εξαγώνου
 * @param x Το x-κέντρο του εξαγώνου
 * @param y Το y-κέντρο του εξαγώνου
 * @param radius Η ακτίνα του εξαγώνου
 * @param vertices Πίνακας για αποθήκευση των συντεταγμένων των κορυφών
 * @details Υπολογίζει τις συντεταγμένες των κορυφών του εξαγώνου με βάση το κέντρο
και την ακτίνα.
 */
void getHexagonVertices(float x, float y, float radius, float vertices[12]) {
    for (int i = 0; i < 6; i++) {
        vertices[2 * i] = x + radius * cos(M_PI / 3 * i);         // Συντεταγμένη x
για κάθε κορυφή
        vertices[2 * i + 1] = y + radius * sin(M_PI / 3 * i);     // Συντεταγμένη y
για κάθε κορυφή
    }
}

/**
 * Συνάρτηση για σχεδίαση ενός πολύγωνου με τη χρήση της graphics::drawLine
 * @param vertices Πίνακας συντεταγμένων των κορυφών του πολύγωνου
 * @param vertex_count Ο αριθμός κορυφών του πολύγωνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα
 * @details Σχεδιάζει ένα πολύγωνο συνδέοντας τις κορυφές μεταξύ τους.
 */
void drawPolygon(float vertices[], int vertex_count, graphics::Brush& br) {
    for (int i = 0; i < vertex_count; i++) {
        int next_index = (i + 1) % vertex_count;
        graphics::drawLine(vertices[2 * i], vertices[2 * i + 1],
            vertices[2 * next_index], vertices[2 * next_index + 1], br);
    }
}

/**
 * Συνάρτηση που σχεδιάζει ένα εξάγωνο
 * @param x Το x-κέντρο του εξαγώνου
 * @param y Το y-κέντρο του εξαγώνου
 * @param radius Η ακτίνα του εξαγώνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα
 * @details Χρησιμοποιεί τις συντεταγμένες από την `getHexagonVertices` για να
σχεδιάσει το εξάγωνο.
 */
void drawHexagon(float x, float y, float radius, graphics::Brush& br) {
    float vertices[12];
    getHexagonVertices(x, y, radius, vertices);
    drawPolygon(vertices, 6, br); // Σχεδίαση του εξαγώνου χρησιμοποιώντας την
`drawPolygon`
}

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Hexaflake
 * @param x Το x-κέντρο του εξαγώνου
 * @param y Το y-κέντρο του εξαγώνου
 * @param radius Η ακτίνα του εξαγώνου
 * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
 * @details Καλεί αναδρομικά για τη σχεδίαση των έξι μικρότερων εξαγώνων γύρω από το
κεντρικό εξάγωνο.
 */
void hexaflake(float x, float y, float radius, int depth) {
    if (depth == 0) {
        return;
    }

    // Ορισμός χρώματος για το εξάγωνο, με απόχρωση που εξαρτάται από το βάθος
    graphics::Brush br;
```

```
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.6f;
    br.fill_color[1] = 0.2f;
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.6f;

    // Σχεδίαση του κεντρικού εξαγώνου
    drawHexagon(x, y, radius, br);

    // Υπολογισμός της ακτίνας για τα μικρότερα εξάγωνα
    float newRadius = radius / 3.0f;

    // Αναδρομική κλήση για τα έξι εξάγωνα γύρω από το κεντρικό
    for (int i = 0; i < 6; i++) {
        float newX = x + 2 * newRadius * cos(M_PI / 3 * i);
        float newY = y + 2 * newRadius * sin(M_PI / 3 * i);
        hexaflake(newX, newY, newRadius, depth - 1);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Ρυθμίζει το φόντο και εκκινεί τη σχεδίαση του Hexaflake στο κέντρο του
παραθύρου.
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background); // Μαύρο φόντο για αντίθεση

    // Αρχικές συντεταγμένες και ακτίνα για το κεντρικό εξάγωνο
    float startX = windowWidth / 2.0f;
    float startY = windowHeight / 2.0f;
    float startRadius = 200.0f;

    // Κλήση της συνάρτησης για σχεδίαση του Hexaflake
    hexaflake(startX, startY, startRadius, max_depth);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Hexaflake");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει το μέγιστο επίπεδο αναδρομής για το fractal.
2. **Συνάρτηση** `getHexagonVertices()`:

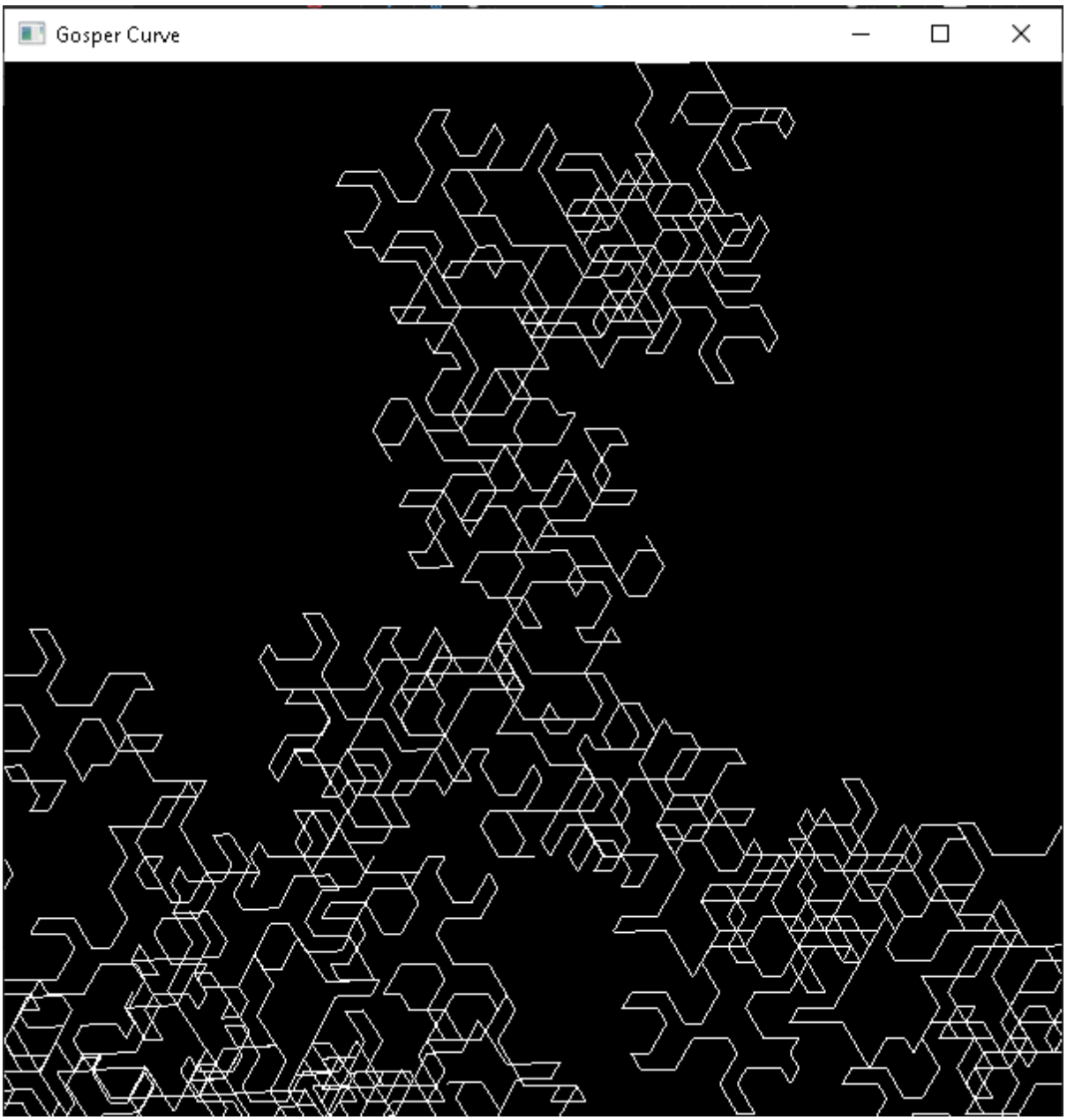
- Αυτή η συνάρτηση υπολογίζει τις συντεταγμένες των έξι κορυφών ενός εξαγώνου με βάση το κέντρο (x,y)(x, y)(x,y) και την ακτίνα `radius`.

3. **Συνάρτηση** `drawHexagon()`:
    - Η συνάρτηση σχεδιάζει ένα εξάγωνο στο σημείο (x,y)(x, y)(x,y) με ακτίνα `radius` και χρώμα `Brush` που παρέχεται από το αντικείμενο `Brush`.
4. **Συνάρτηση** `hexaflake()`:
    - Η συνάρτηση αυτή είναι αναδρομική και δημιουργεί το Hexaflake.
    - Αν το `depth` είναι 0, η διαδικασία αναδρομής σταματά.
    - Σχεδιάζει το κεντρικό εξάγωνο στο σημείο (x,y)(x, y)(x,y) με την καθορισμένη ακτίνα `radius`.
    - Υπολογίζει τη νέα ακτίνα `newRadius`, που είναι το ένα τρίτο της αρχικής.
    - Καλεί αναδρομικά τη `hexaflake()` για τα έξι μικρότερα εξάγωνα που τοποθετούνται γύρω από το κεντρικό.
5. **Συνάρτηση** `draw()`:
    - Η `draw()` είναι η κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
    - καλεί τη `hexaflake()` για να σχεδιάσει το fractal από το κέντρο του παραθύρου.
6. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Επεξήγηση της Συνάρτησης `drawPolygon`

- **Στόχος**: Η συνάρτηση `drawPolygon` χρησιμοποιεί την `graphics::drawLine` για να συνδέσει κάθε κορυφή ενός πολύγωνου με την επόμενη, σχηματίζοντας τα πλευρά του πολύγωνου.
- **Λειτουργία**:
    - Λαμβάνει τις συντεταγμένες των κορυφών μέσω του πίνακα `vertices`, τον αριθμό κορυφών (`vertex_count`), και το αντικείμενο `Brush` (`br`) για το χρώμα.
    - Επαναλαμβάνει για κάθε κορυφή και υπολογίζει την επόμενη κορυφή ως `next_index`. Συνδέει την τρέχουσα κορυφή με την επόμενη με τη χρήση της `graphics::drawLine`.
    - Όταν φτάσει στην τελευταία κορυφή, συνδέεται με την πρώτη, κλείνοντας έτσι το πολύγωνο.

Gosper Curve



Η **Gosper Curve**, γνωστή και ως καμπύλη του Gosper ή καμπύλη του εξαγώνου, είναι μια fractal καμπύλη γεμίσματος επιπέδου που παράγει ένα μοτίβο σε σχήμα εξαγώνου με κάθε επίπεδο αναδρομής. Η καμπύλη αυτή δημιουργείται αναδρομικά μέσω κανόνων αντικατάστασης, όπου κάθε "στροφή" προσθέτει περισσότερες λεπτομέρειες στο σχήμα.

Για να υλοποιήσουμε την Gosper Curve χρησιμοποιώντας τη βιβλιοθήκη SGG, μπορούμε να εφαρμόσουμε τους κανόνες αναδρομής και να σχεδιάσουμε την καμπύλη σε δισδιάστατο επίπεδο.

Κώδικας

```
#include "sgg/graphics.h"
```

```cpp
#include <cmath>
#include <string>
#define M_PI 3.14159265358979323846264338327950288 /* pi */

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;
const int windowHeight = 600;

// Ορισμός του μέγιστου βάθους για τη Gosper Curve
const int max_depth = 4;

// Μήκος γραμμής για την αρχική Gosper Curve
float lineLength = 10.0f;

/**
 * Συνάρτηση που μεταφράζει τις εντολές της Gosper Curve σε οδηγίες για σχεδίαση
 * @param x Τρέχουσα συντεταγμένη x
 * @param y Τρέχουσα συντεταγμένη y
 * @param angle Η γωνία σχεδίασης
 * @param length Το μήκος της γραμμής
 * @param instructions Το μοτίβο των εντολών της καμπύλης
 * @details Με βάση τις εντολές στο string `instructions`, η συνάρτηση αυτή σχεδιάζει γραμμές ή περιστρέφει τη γωνία σχεδίασης.
 */
void drawGosperCurve(float& x, float& y, float angle, float length, const std::string& instructions) {
    graphics::Brush br;
    br.fill_color[0] = 0.0f;  // Χρώμα (κόκκινο)
    br.fill_color[1] = 0.0f;
    br.fill_color[2] = 1.0f;

    // Ερμηνεία και σχεδίαση με βάση τις εντολές στο `instructions`
    for (char command : instructions) {
        if (command == 'F') {
            float newX = x + length * cos(angle);
            float newY = y + length * sin(angle);
            graphics::drawLine(x, y, newX, newY, br);

            // Ενημέρωση της θέσης της γραμμής
            x = newX;
            y = newY;
        }
        else if (command == '+') {
            angle -= M_PI / 3; // Περιστροφή -60 μοίρες
        }
        else if (command == '-') {
            angle += M_PI / 3; // Περιστροφή +60 μοίρες
        }
    }
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του μοτίβου της Gosper Curve
 * @param depth Το βάθος της αναδρομής
 * @param isA Αν αληθές, χρησιμοποιεί τον κανόνα A. Διαφορετικά, χρησιμοποιεί τον κανόνα B.
 * @return Ένα string με το μοτίβο των εντολών για τη Gosper Curve στο τρέχον βάθος
 * @details Κάθε βαθμίδα αναδρομής χρησιμοποιεί τους κανόνες `aRule` και `bRule` για τη δημιουργία του μοτίβου της καμπύλης.
 */
std::string generateGosperCurve(int depth, bool isA = true) {
    if (depth == 0) {
        return isA ? "F" : "F";
    }
```

```
    // Ορισμός των κανόνων A και B
    std::string aRule = "A-F--B+AF+A+FA--F-BF-AF+A+FA+FB--";
    std::string bRule = "B+AF--BFA+FBF--F-AF+BFA+FBF--F-";

    std::string result;
    std::string rule = isA ? aRule : bRule;

    for (char c : rule) {
        if (c == 'A') {
            result += generateGosperCurve(depth - 1, true);
        }
        else if (c == 'B') {
            result += generateGosperCurve(depth - 1, false);
        }
        else {
            result += c;
        }
    }
    return result;
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Θέτει τις αρχικές συντεταγμένες και τη γωνία, δημιουργεί τις οδηγίες της
Gosper Curve και εκκινεί τη διαδικασία σχεδίασης.
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background);

    // Αρχικές συντεταγμένες και γωνία για την καμπύλη
    float startX = 200.0f;
    float startY = 400.0f;
    float startAngle = 0.0f;

    // Δημιουργία των οδηγιών για τη Gosper Curve
    std::string gosperInstructions = generateGosperCurve(max_depth);

    // Σχεδίαση της Gosper Curve
    drawGosperCurve(startX, startY, startAngle, lineLength, gosperInstructions);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Gosper Curve");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:

- Το παράθυρο έχει διαστάσεις 600x600 pixels, και το `max_depth` καθορίζει το επίπεδο της αναδρομής για την Gosper Curve.

2. **Συνάρτηση** `generateGosperCurve()`:
    - Αυτή η συνάρτηση είναι αναδρομική και δημιουργεί το μοτίβο της Gosper Curve ως συμβολοσειρά.
    - Χρησιμοποιεί δύο κανόνες αντικατάστασης (`aRule` και `bRule`) για τα μοτίβα `"A"` και `"B"`, σύμφωνα με τον ορισμό της καμπύλης.
    - Το μοτίβο αυτό παράγει την ακολουθία των εντολών που χρησιμοποιείται για τη σχεδίαση.
3. **Συνάρτηση** `drawGosperCurve()`:
    - Η συνάρτηση αυτή ερμηνεύει τις οδηγίες της Gosper Curve και σχεδιάζει τη γραμμή.
    - Για κάθε εντολή:
        - `'F'`: Προχωρά σε μια νέα θέση σχεδιάζοντας μια γραμμή.
        - `'+'`: Περιστρέφεται κατά -60° (προς τα δεξιά).
        - `'-'`: Περιστρέφεται κατά +60° (προς τα αριστερά).
4. **Συνάρτηση** `draw()`:

    ορίζει το αρχικό σημείο και τη γωνία για την καμπύλη.
    - Καλεί τη `generateGosperCurve()` για να παράγει τις οδηγίες της καμπύλης με το επιθυμητό βάθος αναδρομής και στη συνέχεια τη `drawGosperCurve()` για να σχεδιάσει την καμπύλη στο παράθυρο.
5. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Levy C Curve

Η **Levy C Curve** είναι ένα fractal που δημιουργείται με τη διαδοχική αντικατάσταση μιας ευθείας γραμμής με δύο γραμμές τοποθετημένες σε γωνία 45 μοιρών μεταξύ τους. Με κάθε επίπεδο αναδρομής, το fractal αποκτά περισσότερες λεπτομέρειες και μοιάζει με ένα πολύπλοκο μοτίβο που θυμίζει καμπύλη "C".

Ακολουθεί το πρόγραμμα για τη σχεδίαση της **Levy C Curve** χρησιμοποιώντας τη βιβλιοθήκη SGG, με αναλυτικά σχόλια.

Κώδικας

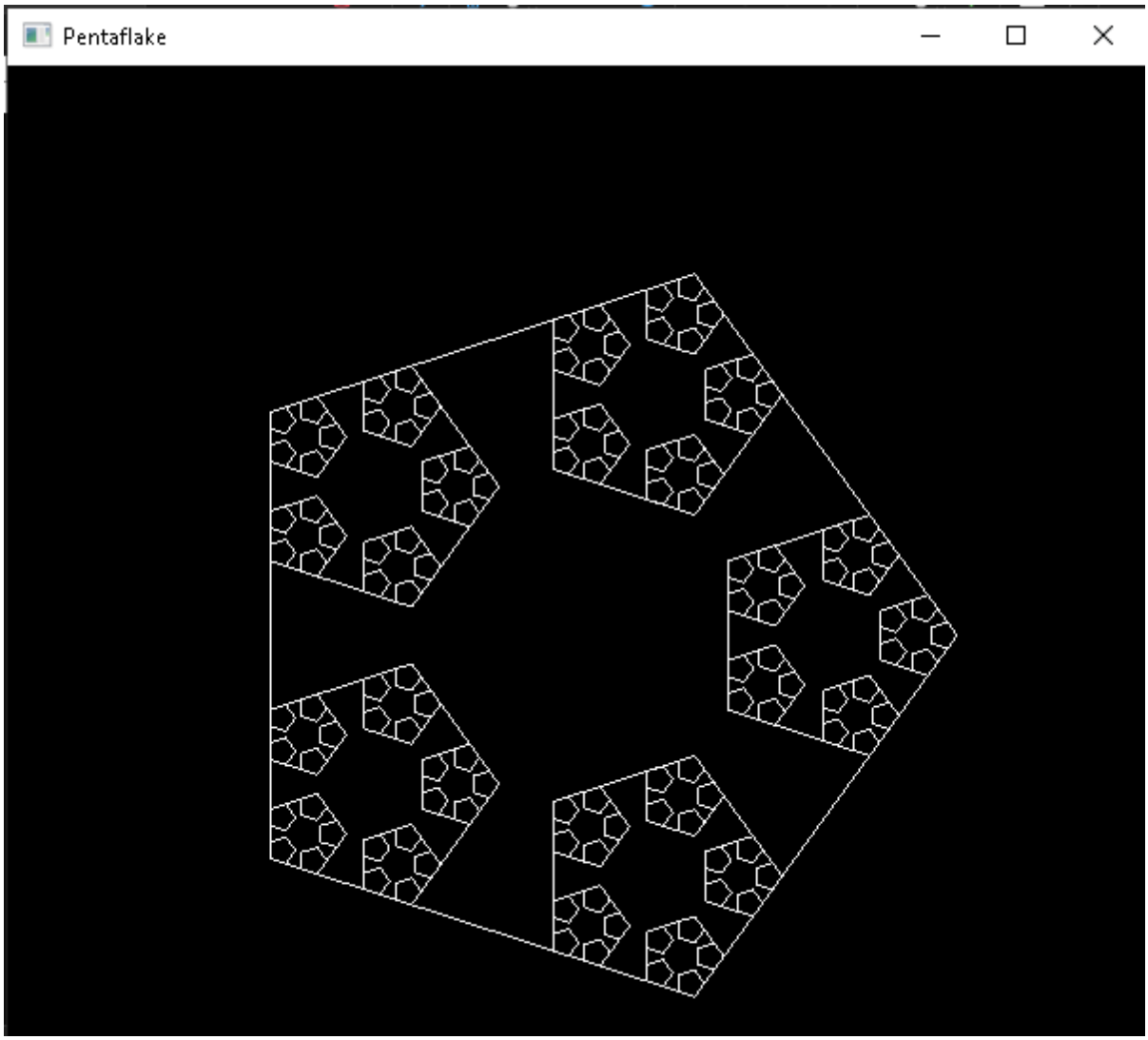
```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;
const int windowHeight = 600;

// Ορισμός του μέγιστου βάθους για τη Levy C Curve
const int max_depth = 12;

/**
 * Αναδρομική συνάρτηση για σχεδίαση της Levy C Curve
 * @param x1 Συντεταγμένη x του αρχικού σημείου
```

```
 * @param y1 Συντεταγμένη y του αρχικού σημείου
 * @param x2 Συντεταγμένη x του τελικού σημείου
 * @param y2 Συντεταγμένη y του τελικού σημείου
 * @param depth Βάθος αναδρομής
 * @details Αν το βάθος είναι 0, σχεδιάζει μια ευθεία γραμμή μεταξύ των δύο σημείων.
Σε μεγαλύτερο βάθος, υπολογίζει το μεσαίο σημείο με περιστροφή 45 μοιρών και καλεί
αναδρομικά την ίδια συνάρτηση για κάθε τμήμα.
 */
void drawLevyCurve(float x1, float y1, float x2, float y2, int depth) {
    // Όταν φτάσουμε στο βάθος 0, σχεδιάζουμε μια ευθεία γραμμή μεταξύ των σημείων
    if (depth == 0) {
        graphics::Brush br;
        br.fill_color[0] = 0.0f;
        br.fill_color[1] = 0.0f;
        br.fill_color[2] = 1.0f; // Μπλε χρώμα για τη γραμμή
        graphics::drawLine(x1, y1, x2, y2, br);
    }
    else {
        // Υπολογισμός του μεσαίου σημείου με γωνία 45 μοιρών
        float mid_x = (x1 + x2) / 2 + (y2 - y1) / 2 * std::sqrt(2) / 2;
        float mid_y = (y1 + y2) / 2 - (x2 - x1) / 2 * std::sqrt(2) / 2;

        // Αναδρομική κλήση για το πρώτο και το δεύτερο τμήμα της καμπύλης
        drawLevyCurve(x1, y1, mid_x, mid_y, depth - 1);
        drawLevyCurve(mid_x, mid_y, x2, y2, depth - 1);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Ορίζει το φόντο του παραθύρου και σχεδιάζει την Levy C Curve, ξεκινώντας
από προκαθορισμένες αρχικές και τελικές συντεταγμένες.
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background);

    // Αρχικές συντεταγμένες για τη Levy C Curve
    float startX = 300.0f;
    float startY = 400.0f;
    float endX = 500.0f;
    float endY = 400.0f;

    // Κλήση της συνάρτησης για τη σχεδίαση της Levy C Curve
    drawLevyCurve(startX, startY, endX, endY, max_depth);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Levy C Curve");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Ορισμός Παραθύρου και Βάθους Αναδρομής**:
    - Το παράθυρο έχει διαστάσεις 600x600 pixels, και η παράμετρος `max_depth` καθορίζει το επίπεδο αναδρομής για τη Levy C Curve. Όσο μεγαλύτερο το βάθος, τόσο περισσότερες λεπτομέρειες θα αποκτά η καμπύλη.
2. **Συνάρτηση** `drawLevyCurve()`:
    - Η συνάρτηση αυτή είναι αναδρομική και δημιουργεί την καμπύλη Levy C.
    - Αν το `depth` είναι 0, σχεδιάζει μια ευθεία γραμμή από το σημείο (x1,y1)(x1, y1)(x1,y1) στο σημείο (x2,y2)(x2, y2)(x2,y2).
    - Αν το `depth` δεν είναι 0, υπολογίζει το μεσαίο σημείο (`mid_x`, `mid_y`) τοποθετημένο σε γωνία 45 μοιρών από το μέσο της γραμμής.
    - Καλεί τη `drawLevyCurve()` αναδρομικά για να σχεδιάσει τις δύο γραμμές που προκύπτουν, κάνοντας έτσι την καμπύλη να "στρίβει".
3. **Συνάρτηση** `draw()`:
    - Η `draw()` είναι η κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
    - Ορίζει τις αρχικές συντεταγμένες της καμπύλης Levy C, οι οποίες αποτελούν τη βάση της καμπύλης.
    - Καλεί τη `drawLevyCurve()` με τα αρχικά σημεία και το επιθυμητό βάθος αναδρομής.
4. **Κύριος Βρόχος και Εκκίνηση Παραθύρου**:
    - Το πρόγραμμα ξεκινά με τη δημιουργία του παραθύρου και τον ορισμό της `draw()` ως συνάρτηση σχεδίασης.
    - Η λειτουργία `startMessageLoop()` διατηρεί το παράθυρο ενεργό μέχρι να το κλείσει ο χρήστης.

Pentaflake



Για να σχεδιάσουμε το **Pentaflake** με τη συνάρτηση `drawPolygon`, μπορούμε να χρησιμοποιήσουμε την `graphics::drawLine` για να συνδέσουμε κάθε κορυφή του πενταγώνου με την επόμενη, σχηματίζοντας τα πλευρά του. Παρακάτω είναι το πλήρες πρόγραμμα για το **Pentaflake** με την υλοποίηση της συνάρτησης `drawPolygon` που χρησιμοποιεί την `graphics::drawLine`.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
# define M_PI 3.14159265358979323846264338327950288 /* pi */

// Ορισμός διαστάσεων παραθύρου
const int windowWidth = 600;
const int windowHeight = 600;

// Μέγιστο βάθος αναδρομής για το Pentaflake
const int max_depth = 4;

/**
 * Συνάρτηση για τον υπολογισμό των κορυφών ενός πενταγώνου
 * @param x Κέντρο του πενταγώνου στον άξονα x
 * @param y Κέντρο του πενταγώνου στον άξονα y
```

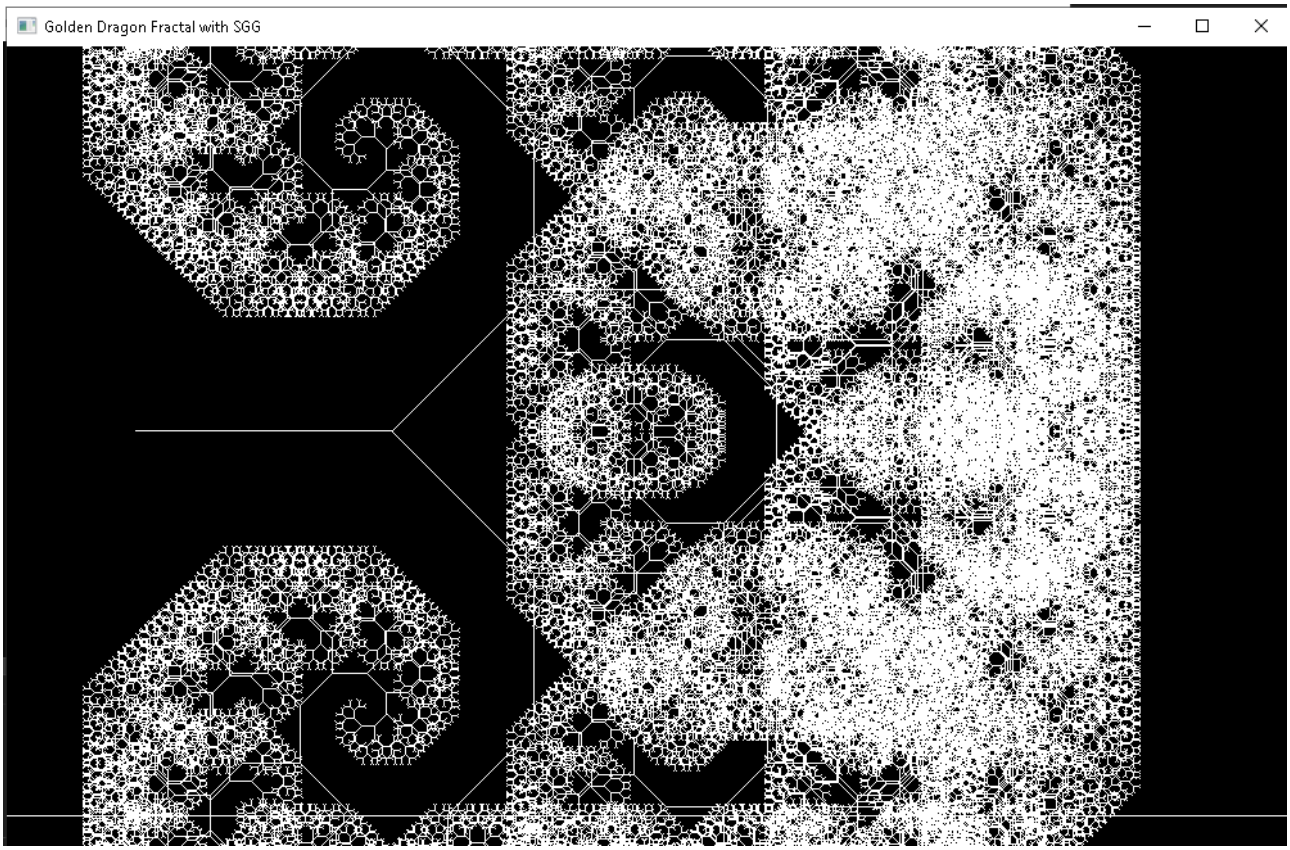
```
 * @param radius Ακτίνα του πενταγώνου
 * @param vertices Πίνακας που θα αποθηκεύσει τις συντεταγμένες των κορυφών
 */
void getPentagonVertices(float x, float y, float radius, float vertices[10]) {
    for (int i = 0; i < 5; i++) {
        vertices[2 * i] = x + radius * cos(2 * M_PI * i / 5);         // Συντεταγμένη
x για κάθε κορυφή
        vertices[2 * i + 1] = y + radius * sin(2 * M_PI * i / 5);     // Συντεταγμένη
y για κάθε κορυφή
    }
}

/**
 * Συνάρτηση για σχεδίαση ενός πολύγωνου χρησιμοποιώντας την graphics::drawLine
 * @param vertices Πίνακας με τις συντεταγμένες των κορυφών
 * @param vertex_count Αριθμός κορυφών του πολυγώνου
 * @param br Αντικείμενο γραφής για την εμφάνιση χρώματος
 */
void drawPolygon(float vertices[], int vertex_count, graphics::Brush& br) {
    for (int i = 0; i < vertex_count; i++) {
        int next_index = (i + 1) % vertex_count;  // Σύνδεση της τρέχουσας κορυφής με
την επόμενη
        graphics::drawLine(vertices[2 * i], vertices[2 * i + 1],
            vertices[2 * next_index], vertices[2 * next_index + 1], br);
    }
}

/**
 * Συνάρτηση που σχεδιάζει ένα πεντάγωνο
 * @param x Συντεταγμένη x του κέντρου του πενταγώνου
 * @param y Συντεταγμένη y του κέντρου του πενταγώνου
 * @param radius Ακτίνα του πενταγώνου
 * @param br Αντικείμενο γραφής για την εμφάνιση χρώματος
 */
void drawPentagon(float x, float y, float radius, graphics::Brush& br) {
    float vertices[10];  // Πίνακας για αποθήκευση συντεταγμένων των κορυφών
    getPentagonVertices(x, y, radius, vertices);  // Υπολογισμός συντεταγμένων των
κορυφών
    drawPolygon(vertices, 5, br); // Σχεδίαση του πενταγώνου
}

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Pentaflake
 * @param x Συντεταγμένη x του κέντρου του πενταγώνου
 * @param y Συντεταγμένη y του κέντρου του πενταγώνου
 * @param radius Ακτίνα του πενταγώνου
 * @param depth Τρέχον βάθος αναδρομής
 */
void pentaflake(float x, float y, float radius, int depth) {
    if (depth == 0) {
        return; // Σταματάμε αν φτάσουμε στο μέγιστο βάθος
    }

    // Ορισμός χρώματος για το πεντάγωνο με βάση το βάθος αναδρομής
    graphics::Brush br;
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.6f;
    br.fill_color[1] = 0.2f;
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.6f;

    // Σχεδίαση του κεντρικού πενταγώνου
    drawPentagon(x, y, radius, br);

    // Υπολογισμός της ακτίνας για τα μικρότερα πεντάγωνα
    float newRadius = radius / 3.0f;
```

```
    // Αναδρομική κλήση για τα πέντε πεντάγωνα γύρω από το κεντρικό
    for (int i = 0; i < 5; i++) {
        float newX = x + 2 * newRadius * cos(2 * M_PI * i / 5);  // Νέα x θέση
        float newY = y + 2 * newRadius * sin(2 * M_PI * i / 5);  // Νέα y θέση
        pentaflake(newX, newY, newRadius, depth - 1);  // Αναδρομική κλήση
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθορίζει το φόντο του παραθύρου και καλεί τη συνάρτηση pentaflake για να ξεκινήσει η σχεδίαση του fractal από το κέντρο.
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);

    // Έναρξη σχεδίασης του Pentaflake από το κέντρο του παραθύρου
    float startX = windowWidth / 2.0f;
    float startY = windowHeight / 2.0f;
    float startRadius = 200.0f; // Αρχική ακτίνα του πενταγώνου

    // Κλήση της συνάρτησης για σχεδίαση του Pentaflake
    pentaflake(startX, startY, startRadius, max_depth);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(windowWidth, windowHeight, "Pentaflake");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου
    graphics::startMessageLoop();

    // Καταστροφή παραθύρου μετά την έξοδο
    graphics::destroyWindow();

    return 0;
}
```

Επεξήγηση της Συνάρτησης `drawPolygon`

- **Στόχος**: Η συνάρτηση `drawPolygon` χρησιμοποιεί την `graphics::drawLine` για να συνδέσει κάθε κορυφή ενός πολύγωνου με την επόμενη, σχηματίζοντας τις πλευρές του πολύγωνου.
- **Λειτουργία**:
    - Λαμβάνει τις συντεταγμένες των κορυφών μέσω του πίνακα `vertices`, τον αριθμό κορυφών (`vertex_count`), και το αντικείμενο `Brush` (`br`) για το χρώμα.
    - Επαναλαμβάνει για κάθε κορυφή και υπολογίζει την επόμενη κορυφή ως `next_index`. Συνδέει την τρέχουσα κορυφή με την επόμενη με τη χρήση της `graphics::drawLine`.
    - Όταν φτάσει στην τελευταία κορυφή, συνδέεται με την πρώτη, κλείνοντας έτσι το πολύγωνο.

Golden Dragon Fractals



Το *Golden Dragon Fractal* είναι ένα fractal που μπορεί να δημιουργηθεί μέσω αναδρομής και περιστροφών, βασισμένο σε κανόνες προσανατολισμού και κλίμακας. Θα χρησιμοποιήσουμε αναδρομική προσέγγιση και τις βασικές συναρτήσεις σχεδίασης της *SGG Library* για να υλοποιήσουμε το fractal.

Οδηγίες:

1. Το fractal ξεκινά από μια αρχική γραμμή.
2. Κάθε γενιά γραμμών σχηματίζει το επόμενο επίπεδο fractal με συγκεκριμένη περιστροφή και μείωση κλίμακας.
3. Χρησιμοποιούμε την *SGG Library* για να σχεδιάσουμε το fractal με αναδρομική κλήση.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <cmath>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 1000;
const int WINDOW_HEIGHT = 600;

// Παράμετροι για το fractal
const int MAX_DEPTH = 18;          // Μέγιστο βάθος αναδρομής
const float SCALE_FACTOR = 0.78f;  // Παράγοντας κλίμακας για κάθε νέο τμήμα
```

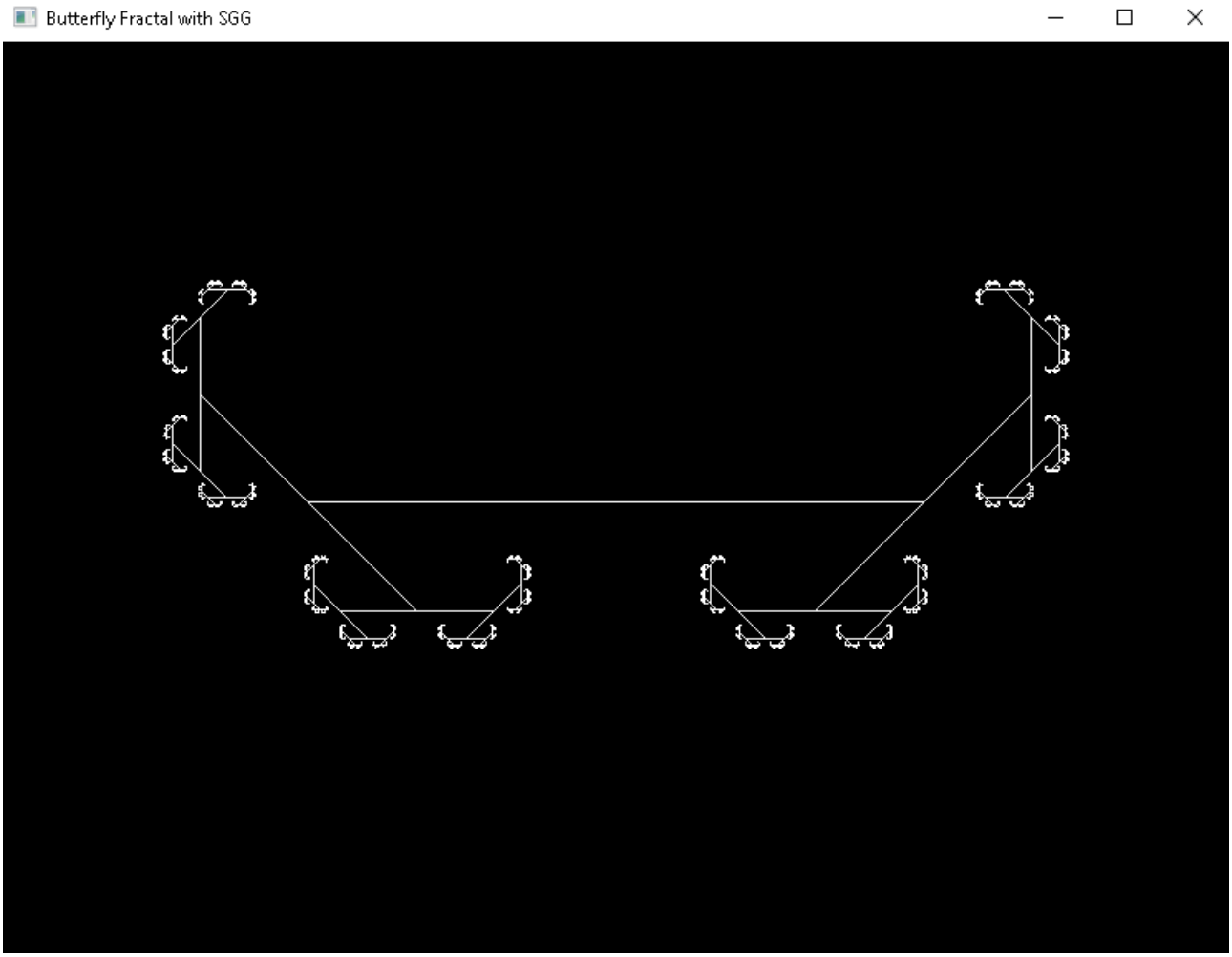
```
const float ANGLE_OFFSET = 45.0f;  // Γωνία περιστροφής σε κάθε στάδιο (μοίρες)

/**
 * Συνάρτηση για την αναδρομική σχεδίαση του Golden Dragon Fractal
 * @param x Συντεταγμένη x του σημείου εκκίνησης
 * @param y Συντεταγμένη y του σημείου εκκίνησης
 * @param length Μήκος του τρέχοντος τμήματος της γραμμής
 * @param angle Γωνία σχεδίασης σε μοίρες
 * @param depth Τρέχον βάθος αναδρομής
 *
 * Αυτή η συνάρτηση σχεδιάζει το fractal Golden Dragon χρησιμοποιώντας αναδρομή.
 * Σε κάθε στάδιο, υπολογίζει τις συντεταγμένες του επόμενου σημείου και σχεδιάζει
 * γραμμές με διαφορετικές γωνίες περιστροφής.
 */
void drawGoldenDragon(float x, float y, float length, float angle, int depth) {
    // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
    if (depth == 0) return;

    // Υπολογισμός τελικών συντεταγμένων για το τρέχον τμήμα
    float xEnd = x + length * cos(angle * M_PI / 180.0f);
    float yEnd = y + length * sin(angle * M_PI / 180.0f);

    // Ρύθμιση του πινέλου για τη σχεδίαση με σταδιακή αλλαγή χρώματος
    graphics::Brush brush;
    brush.fill_color[0] = 0.3f + depth * 0.04f; // Αυξάνει τον τόνο του κόκκινου όσο
αυξάνεται το βάθος
    brush.fill_color[1] = 0.2f;                 // Πράσινος τόνος
    brush.fill_color[2] = 0.5f + depth * 0.02f; // Αυξάνει τον τόνο του μπλε όσο
αυξάνεται το βάθος

    // Σχεδίαση γραμμής από το σημείο (x, y) στο (xEnd, yEnd)
    graphics::drawLine(x, y, xEnd, yEnd, brush);

    // Αναδρομική κλήση για σχεδίαση των επόμενων τμημάτων
    // Περιστροφή κατά -ANGLE_OFFSET για το πρώτο τμήμα και +ANGLE_OFFSET για το
δεύτερο
    drawGoldenDragon(xEnd, yEnd, length * SCALE_FACTOR, angle - ANGLE_OFFSET, depth -
1);
    drawGoldenDragon(xEnd, yEnd, length * SCALE_FACTOR, angle + ANGLE_OFFSET, depth -
1);
}

/**
 * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ
 * @details Καθορίζει το φόντο και καλεί τη συνάρτηση σχεδίασης του fractal.
 */
void draw() {
    // Ρύθμιση του φόντου σε μαύρο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Κόκκινο συστατικό
    bg.fill_color[1] = 0.0f; // Πράσινο συστατικό
    bg.fill_color[2] = 0.0f; // Μπλε συστατικό
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης για σχεδίαση του Golden Dragon Fractal από το αριστερό
κέντρο του παραθύρου
    drawGoldenDragon(WINDOW_WIDTH / 10, WINDOW_HEIGHT / 2, 200.0f, 0.0f, MAX_DEPTH);
}

int main() {
    // Δημιουργία παραθύρου με διαστάσεις WINDOW_WIDTH x WINDOW_HEIGHT
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Golden Dragon Fractal with
SGG");
```

```
    // Καθορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύριου βρόχου για την απόκριση στο παράθυρο
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawGoldenDragon**: Αναδρομική συνάρτηση που σχεδιάζει τμήματα του fractal με κλίμακα και περιστροφή. Χρησιμοποιεί την *SGG Library* για να σχεδιάσει γραμμές με διαφορετικά χρώματα ανάλογα με το βάθος της αναδρομής.
2. **draw**: Σχεδιάζει το fractal ξεκινώντας από το κέντρο του παραθύρου με προκαθορισμένο μήκος γραμμής και βάθος αναδρομής.

Butterfly Fractals



Το *Butterfly Fractal* είναι ένα γεωμετρικό fractal που δημιουργείται από την αναδρομική σχεδίαση κύκλων και γραμμών. Χαρακτηρίζεται από συμμετρικές γραμμές και μοτίβα που μοιάζουν με τα φτερά μιας πεταλούδας. Θα το σχεδιάσουμε χρησιμοποιώντας αναδρομή και την *SGG Library*.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

// Παράμετροι για το fractal
const int MAX_DEPTH = 20;          // Βάθος αναδρομής
const float SCALE_FACTOR = 0.5f;  // Παράγοντας κλίμακας για κάθε νέο τμήμα
const float ANGLE_OFFSET = 45.0f; // Γωνία περιστροφής για κάθε στάδιο

/**
 * Συνάρτηση για την αναδρομική σχεδίαση του Butterfly Fractal
```

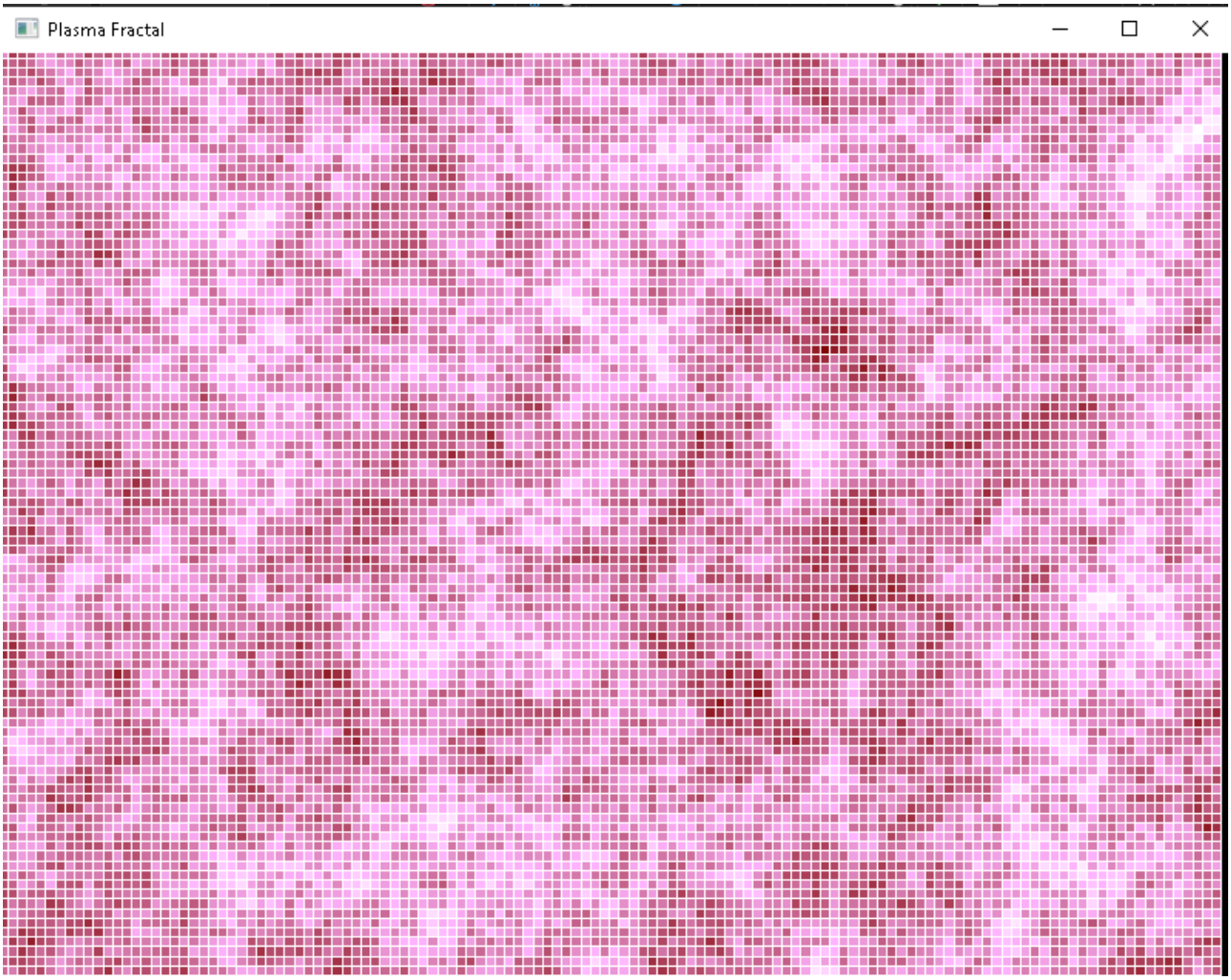
```
 * @param x Συντεταγμένη x του σημείου εκκίνησης
 * @param y Συντεταγμένη y του σημείου εκκίνησης
 * @param length Μήκος του τρέχοντος τμήματος
 * @param angle Γωνία περιστροφής σε μοίρες για το τρέχον τμήμα
 * @param depth Τρέχον βάθος αναδρομής
 *
 * Αυτή η συνάρτηση σχεδιάζει γραμμές που διακλαδίζονται αναδρομικά,
 * δημιουργώντας ένα σχήμα που μοιάζει με πεταλούδα.
 */
void drawButterflyFractal(float x, float y, float length, float angle, int depth) {
    // Όταν το βάθος φτάσει στο 0, σταματά η αναδρομή
    if (depth == 0) return;

    // Υπολογισμός των τελικών συντεταγμένων για τις δύο διακλαδώσεις
    float xEnd1 = x + length * cos(angle * M_PI / 180.0f);
    float yEnd1 = y + length * sin(angle * M_PI / 180.0f);
    float xEnd2 = x + length * cos((angle + 180) * M_PI / 180.0f);
    float yEnd2 = y + length * sin((angle + 180) * M_PI / 180.0f);

    // Ρύθμιση του πινέλου σχεδίασης με χρώμα που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.5f + depth * 0.05f;  // Αυξάνει τον κόκκινο τόνο για κάθε
επίπεδο
    brush.fill_color[1] = 0.3f;                  // Σταθερή απόχρωση πράσινου
    brush.fill_color[2] = 0.7f - depth * 0.05f;  // Μειώνει τον μπλε τόνο για κάθε
επίπεδο

    // Σχεδίαση των δύο γραμμών που αποτελούν τις διακλαδώσεις του fractal
    graphics::drawLine(x, y, xEnd1, yEnd1, brush);
    graphics::drawLine(x, y, xEnd2, yEnd2, brush);

    // Αναδρομική κλήση για τις δύο γραμμές του επόμενου επιπέδου
    // Το μήκος μειώνεται και η γωνία περιστρέφεται κατά ±ANGLE_OFFSET
    drawButterflyFractal(xEnd1, yEnd1, length * SCALE_FACTOR, angle - ANGLE_OFFSET,
depth - 1);
    drawButterflyFractal(xEnd2, yEnd2, length * SCALE_FACTOR, angle + ANGLE_OFFSET,
depth - 1);
}

/**
 * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ
 * @details Καθαρίζει το φόντο και καλεί τη συνάρτηση σχεδίασης του fractal.
 */
void draw() {
    // Ρύθμιση του φόντου σε μαύρο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;  // Κόκκινο συστατικό
    bg.fill_color[1] = 0.0f;  // Πράσινο συστατικό
    bg.fill_color[2] = 0.0f;  // Μπλε συστατικό
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης σχεδίασης του fractal από το κέντρο του παραθύρου
    drawButterflyFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200.0f, 0.0f,
MAX_DEPTH);
}

/**
 * Κύρια συνάρτηση
 * @details Δημιουργεί το παράθυρο και ορίζει τη συνάρτηση σχεδίασης
 */
int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Butterfly Fractal with SGG");
```

```
    // Ορισμός της συνάρτησης σχεδίασης που καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του κύριου βρόχου μηνυμάτων για τη σχεδίαση
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawButterflyFractal**: Η αναδρομική συνάρτηση σχεδίασης που δημιουργεί το *Butterfly Fractal*. Κάθε κλήση σχεδιάζει δύο γραμμές που διακλαδίζονται και δημιουργούν το επόμενο επίπεδο fractal.
2. **draw**: Καλείται σε κάθε καρέ και διασφαλίζει ότι το fractal σχεδιάζεται από το κέντρο του παραθύρου με προκαθορισμένο μέγεθος και γωνία.

Plasma Fractals



Το *Plasma Fractal* είναι ένα fractal που χρησιμοποιεί την τεχνική διαίρεσης και κατάκτησης για να δημιουργήσει ένα τυχαίο αλλά συνεχές μοτίβο χρωματισμού. Μπορούμε να το δημιουργήσουμε σε ένα πλέγμα χρησιμοποιώντας τη διαδικασία *Diamond-Square* για να υπολογίσουμε τις τιμές των σημείων.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <vector>

// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

// Διαστάσεις και παράμετροι πλέγματος
const int GRID_SIZE = 129;       // (2^n) + 1, απαραίτητο για Diamond-Square
float roughness = 0.5f;          // Παράγοντας τραχύτητας για τον έλεγχο του fractal

// Πλέγμα για την αποθήκευση τιμών υψομέτρου
```

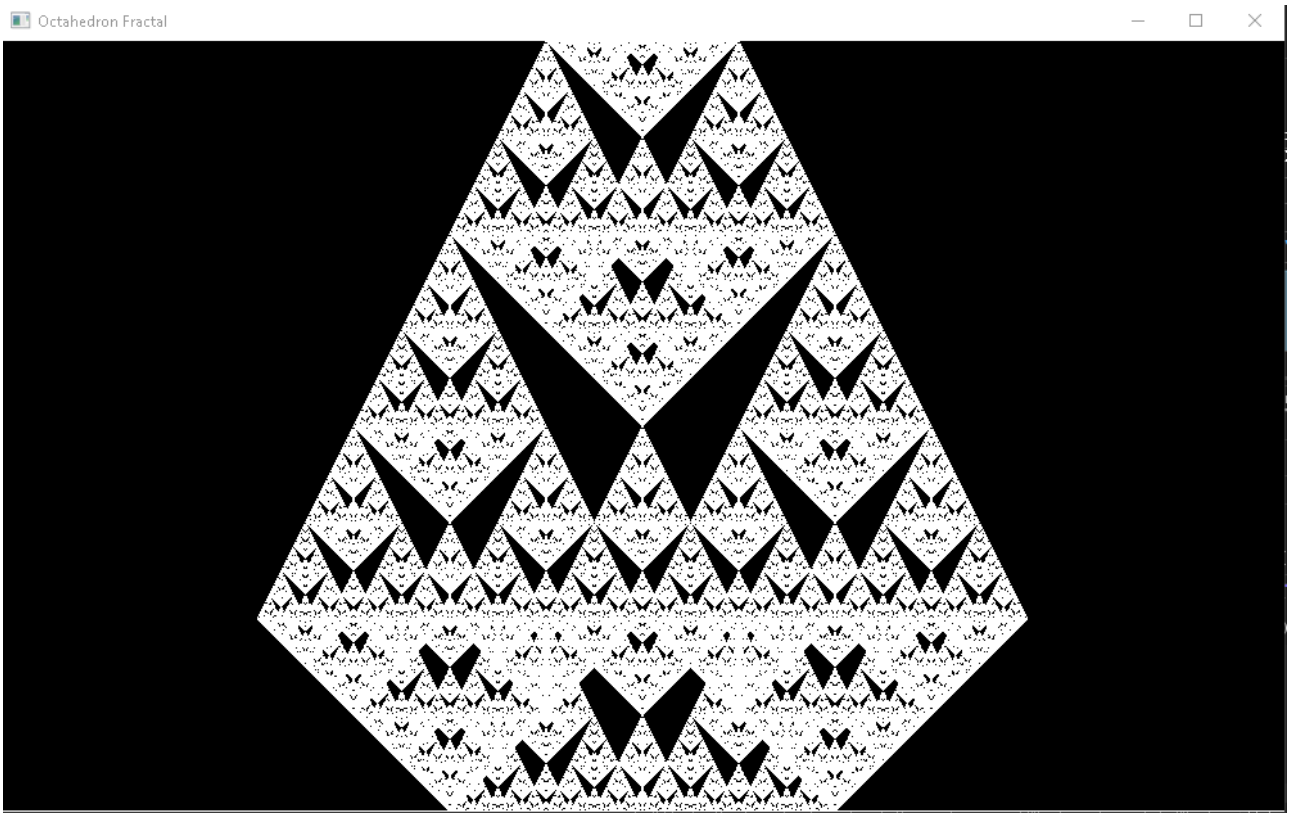
```
std::vector<std::vector<float>> grid(GRID_SIZE, std::vector<float>(GRID_SIZE));

/**
 * Αρχικοποίηση του πλέγματος με τυχαίες αρχικές τιμές στις γωνίες.
 * Οι τιμές αυτές είναι μεταξύ 0 και 1.
 */
void initializeGrid() {
    grid[0][0] = static_cast<float>(rand()) / RAND_MAX;
    grid[0][GRID_SIZE - 1] = static_cast<float>(rand()) / RAND_MAX;
    grid[GRID_SIZE - 1][0] = static_cast<float>(rand()) / RAND_MAX;
    grid[GRID_SIZE - 1][GRID_SIZE - 1] = static_cast<float>(rand()) / RAND_MAX;
}

/**
 * Υπολογίζει τη μέση τιμή τεσσάρων σημείων για χρήση στον αλγόριθμο Diamond-Square.
 * @param a, b, c, d: Τιμές των τεσσάρων σημείων γύρω από ένα σημείο πλέγματος.
 * @return Μέση τιμή των σημείων a, b, c, d.
 */
float average(float a, float b, float c, float d) {
    return (a + b + c + d) / 4.0f;
}

/**
 * Αλγόριθμος Diamond-Square που παράγει το fractal ύψος για το πλέγμα.
 * @param size: Τρέχον μέγεθος του τμήματος του πλέγματος που διαιρείται.
 *
 * Η συνάρτηση εκτελεί τον αλγόριθμο Diamond-Square. Σε κάθε βήμα, υπολογίζει νέα
σημεία
 * χρησιμοποιώντας τον παράγοντα τραχύτητας για την προσθήκη τυχαιότητας.
 */
void diamondSquare(int size) {
    int half = size / 2;
    if (half < 1) return;

    // Diamond step: Εύρεση μέσης τιμής και προσθήκη τυχαιότητας
    for (int y = half; y < GRID_SIZE - 1; y += size) {
        for (int x = half; x < GRID_SIZE - 1; x += size) {
            float avg = average(grid[x - half][y - half], grid[x + half][y - half],
                grid[x - half][y + half], grid[x + half][y + half]);
            grid[x][y] = avg + ((static_cast<float>(rand()) / RAND_MAX) * 2 - 1) *
roughness;
        }
    }

    // Square step: Υπολογισμός για κάθε κελί γύρω από τα διαμάντια
    for (int y = 0; y < GRID_SIZE; y += half) {
        for (int x = (y + half) % size; x < GRID_SIZE; x += size) {
            float avg = average(grid[(x - half + GRID_SIZE) % GRID_SIZE][y],
                grid[(x + half) % GRID_SIZE][y],
                grid[x][(y + half) % GRID_SIZE],
                grid[x][(y - half + GRID_SIZE) % GRID_SIZE]);
            grid[x][y] = avg + ((static_cast<float>(rand()) / RAND_MAX) * 2 - 1) *
roughness;
        }
    }

    // Αναδρομική κλήση για το επόμενο επίπεδο
    diamondSquare(size / 2);
}

/**
 * Σχεδιάζει το fractal χρωματίζοντας το πλέγμα σύμφωνα με τις τιμές υψομέτρου.
 * Το ύψος κάθε κελιού κανονικοποιείται για να παράγει χρώμα.
 */
void drawPlasmaFractal() {
```

```
    graphics::Brush brush;

    for (int y = 0; y < GRID_SIZE - 1; y++) {
        for (int x = 0; x < GRID_SIZE - 1; x++) {
            // Κανονικοποίηση της τιμής ύψους για χρωματική απεικόνιση
            float colorValue = (grid[x][y] + 1) / 2.0f;
            brush.fill_color[0] = colorValue * 0.5f + 0.5f; // Κόκκινη απόχρωση
            brush.fill_color[1] = colorValue * 0.7f;        // Πράσινη απόχρωση
            brush.fill_color[2] = colorValue * 1.0f;        // Μπλε απόχρωση
            float cellSize = static_cast<float>(WINDOW_WIDTH) / GRID_SIZE;
            graphics::drawRect(x * cellSize, y * cellSize, cellSize, cellSize, brush);
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη για να σχεδιάσει το fractal.
 */
void draw() {
    drawPlasmaFractal();
}

/**
 * Κύρια συνάρτηση
 * Δημιουργεί το παράθυρο, αρχικοποιεί το πλέγμα και ξεκινά τον αλγόριθμο Diamond-
Square.
 */
int main() {
    // Αρχικοποίηση της γεννήτριας τυχαίων αριθμών
    srand(static_cast<unsigned int>(time(0)));

    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Plasma Fractal");

    // Αρχικοποίηση πλέγματος και εκτέλεση αλγορίθμου Diamond-Square
    initializeGrid();
    diamondSquare(GRID_SIZE - 1);

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για τη σχεδίαση
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **initializeGrid**: Θέτει τις αρχικές τιμές των γωνιακών σημείων στο πλέγμα.
2. **diamondSquare**: Υλοποιεί τη διαδικασία Diamond-Square για να γεμίσει το πλέγμα με τιμές υψομέτρου.
3. **drawPlasmaFractal**: Μετατρέπει τις τιμές υψομέτρου σε χρώματα και σχεδιάζει το πλέγμα.
4. **draw**: Καλεί τη συνάρτηση σχεδίασης του fractal.

Octahedron Fractal



Το *Octahedron Fractal* είναι ένα fractal που βασίζεται σε επαναλαμβανόμενα μοτίβα οκταέδρων και σχηματίζεται μέσω υποδιαίρεσης. Για να σχεδιάσουμε ένα τέτοιο fractal, μπορούμε να ξεκινήσουμε με ένα βασικό οκτάεδρο και να συνεχίσουμε να προσθέτουμε μικρότερα οκτάεδρα στις πλευρές του σε κάθε επανάληψη, δημιουργώντας ένα τρισδιάστατο αποτέλεσμα.

Ωστόσο, δεδομένου ότι η SGG Library δεν υποστηρίζει απευθείας τρισδιάστατα σχήματα, θα προσεγγίσουμε το *Octahedron Fractal* με σχεδίαση των βασικών γεωμετρικών στοιχείων (τρίγωνα) σε προβολή και θα προσπαθήσουμε να δώσουμε αίσθηση βάθους.

Παρακάτω είναι ένα πρόγραμμα που προσομοιώνει το fractal σε δισδιάστατη μορφή, όπου θα σχεδιάσουμε μικρότερα τρίγωνα για να προσεγγίσουμε το αποτέλεσμα ενός οκταέδρου.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>

// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 1000;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 10;  // Μέγιστο βάθος αναδρομής για το fractal

/**
 * Συνάρτηση που σχεδιάζει ένα τρίγωνο
 * @param x1, y1: Συντεταγμένες της πρώτης κορυφής
 * @param x2, y2: Συντεταγμένες της δεύτερης κορυφής
 * @param x3, y3: Συντεταγμένες της τρίτης κορυφής
```

```
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Η συνάρτηση αυτή χρησιμοποιείται για τη σχεδίαση των πλευρών ενός τριγώνου
 * με τη βοήθεια των τριών γραμμών που συνδέουν τα σημεία (x1, y1), (x2, y2) και (x3,
y3).
 */
void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, const
graphics::Brush& brush) {
    graphics::drawLine(x1, y1, x2, y2, brush);
    graphics::drawLine(x2, y2, x3, y3, brush);
    graphics::drawLine(x3, y3, x1, y1, brush);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του fractal
 * @param x, y: Συντεταγμένες του κεντρικού σημείου του τριγώνου
 * @param size: Μέγεθος του τρέχοντος τριγώνου
 * @param depth: Βάθος της αναδρομής, που καθορίζει το επίπεδο λεπτομέρειας
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Η συνάρτηση καλείται αναδρομικά για να δημιουργήσει τέσσερα μικρότερα τρίγωνα,
 * που τοποθετούνται σε τέσσερις διαφορετικές θέσεις γύρω από το κεντρικό τρίγωνο.
 * Όταν φτάσει στο βάθος 0, σχεδιάζει το βασικό τρίγωνο.
 */
void drawOctahedronFractal(float x, float y, float size, int depth, const
graphics::Brush& brush) {
    if (depth == 0) {
        // Σχεδίαση ενός κεντρικού τρίγωνου για το αρχικό οκτάεδρο
        drawTriangle(x, y - size, x - size, y + size, x + size, y + size, brush);
        return;
    }

    // Υπολογισμός νέου μεγέθους για τα μικρότερα τρίγωνα
    float newSize = size / 2.0f;

    // Αναδρομική κλήση για τη σχεδίαση τεσσάρων μικρότερων τριγώνων
    drawOctahedronFractal(x, y - size, newSize, depth - 1, brush);        // Πάνω
τρίγωνο
    drawOctahedronFractal(x - size, y + size, newSize, depth - 1, brush); // Κάτω
αριστερό τρίγωνο
    drawOctahedronFractal(x + size, y + size, newSize, depth - 1, brush); // Κάτω
δεξιό τρίγωνο
    drawOctahedronFractal(x, y + size * 2, newSize, depth - 1, brush);    // Κεντρικό
κάτω τρίγωνο
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 *
 * Αρχικά καθαρίζει το παράθυρο σχεδίασης και στη συνέχεια καλεί τη συνάρτηση
 * `drawOctahedronFractal` για να σχεδιάσει το fractal στο κέντρο του παραθύρου.
 */
void draw() {
    // Ορισμός μαύρου φόντου για αντίθεση
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ορισμός του πινέλου για το fractal με απόχρωση μπλε-πράσινου
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.5f;
```

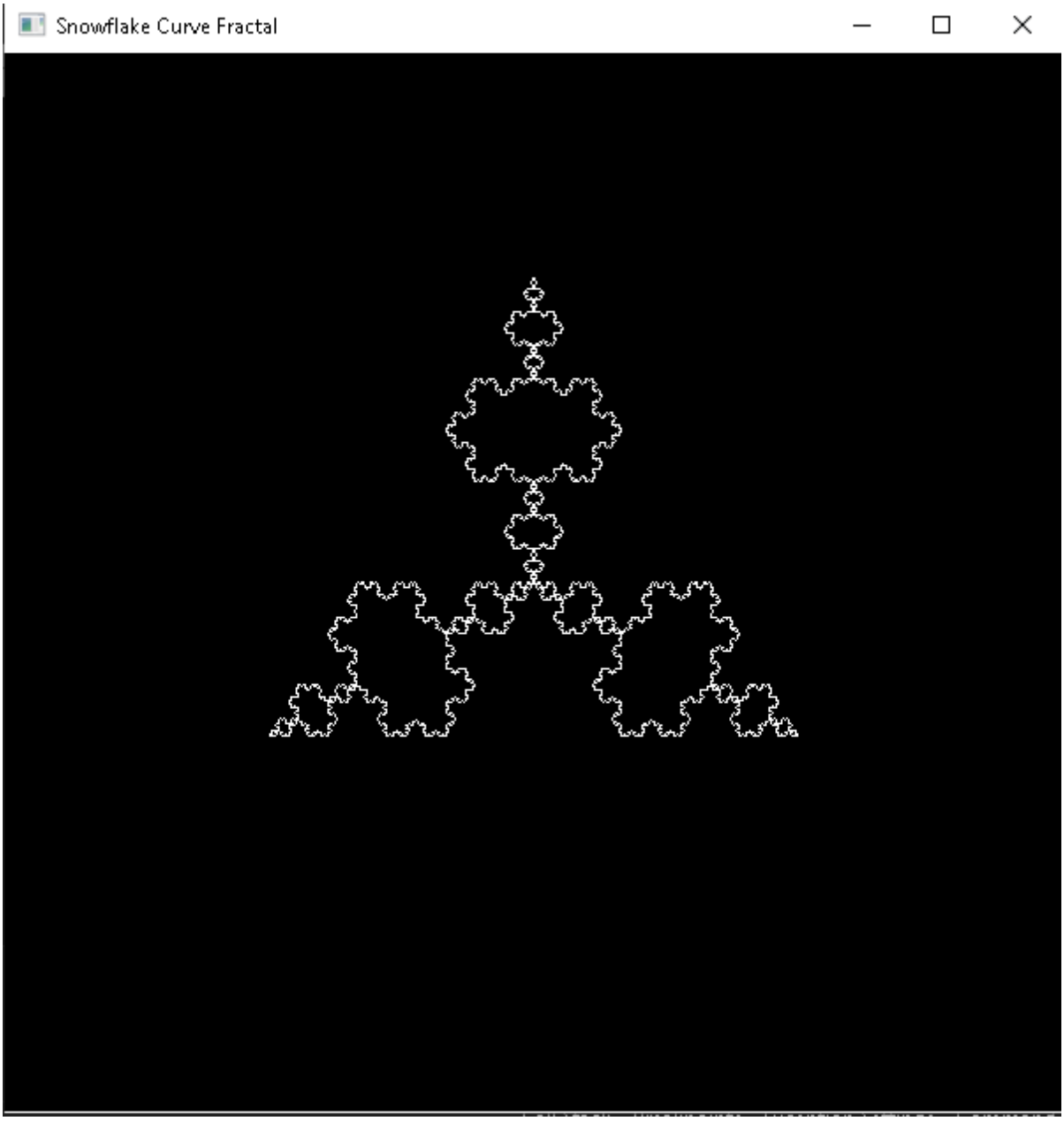
```
    brush.fill_color[2] = 1.0f;

    // Κλήση της συνάρτησης για σχεδίαση του fractal
    drawOctahedronFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 4, WINDOW_HEIGHT / 4,
MAX_DEPTH, brush);
}

/**
 * Κύρια συνάρτηση
 *
 * Αρχικοποιεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά
 * τον κύριο βρόχο μηνυμάτων για να διατηρήσει το παράθυρο ενεργό.
 */
int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Octahedron Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του loop μηνυμάτων για τη σχεδίαση
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawTriangle**: Αυτή η συνάρτηση σχεδιάζει ένα τρίγωνο με τις κορυφές που δίνονται από τις συντεταγμένες `(x1, y1)`, `(x2, y2)`, και `(x3, y3)`.
2. **drawOctahedronFractal**: Αυτή η αναδρομική συνάρτηση δημιουργεί το fractal χρησιμοποιώντας επαναλαμβανόμενες κλήσεις για να τοποθετήσει μικρότερα τρίγωνα σε κάθε επίπεδο του fractal.
3. **draw**: Καθαρίζει την οθόνη και καλεί τη συνάρτηση `drawOctahedronFractal` για να ξεκινήσει η σχεδίαση του fractal.
4. **main**: Δημιουργεί το παράθυρο και εκκινεί τον βρόχο μηνυμάτων για την προβολή του fractal.

Το παραπάνω πρόγραμμα δημιουργεί ένα απλοποιημένο *Octahedron Fractal* χρησιμοποιώντας αναδρομή, που μοιάζει με το αρχικό fractal οκταέδρου αλλά σε δισδιάστατη μορφή.

Snowflake Curve



Το *Snowflake Curve Fractal* ή *Koch Snowflake* είναι ένα από τα πιο κλασικά fractals που δημιουργούνται με αναδρομή. Ξεκινάμε με ένα ισόπλευρο τρίγωνο και σε κάθε επανάληψη υποδιαιρούμε κάθε πλευρά του τρίγωνου σε τμήματα, σχηματίζοντας νέα τρίγωνα πάνω στην καμπύλη.

Παρακάτω είναι ο κώδικας του προγράμματος σε SGG Library για τη σχεδίαση του *Koch Snowflake Fractal*.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>
```

```
#define M_PI 3.14159265358979323846

// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5;  // Μέγιστο βάθος αναδρομής για το fractal

/**
 * Συνάρτηση για σχεδίαση ενός τμήματος της καμπύλης Koch.
 * @param x1, y1: Συντεταγμένες αρχής της γραμμής
 * @param x2, y2: Συντεταγμένες τέλους της γραμμής
 * @param depth: Βάθος αναδρομής που καθορίζει το επίπεδο λεπτομέρειας
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Στην αναδρομή, το τμήμα διαιρείται σε τρία ίσα μέρη και σχηματίζεται
 * ισόπλευρο τρίγωνο με κορυφή εκτός της αρχικής ευθείας. Όταν depth = 0, σχεδιάζεται
ευθεία γραμμή.
 */
void drawKochSegment(float x1, float y1, float x2, float y2, int depth, const
graphics::Brush& brush) {
    if (depth == 0) {
        // Σχεδίαση βασικής γραμμής
        graphics::drawLine(x1, y1, x2, y2, brush);
    }
    else {
        // Υπολογισμός διαφοράς x και y μεταξύ των άκρων του τμήματος
        float dx = x2 - x1;
        float dy = y2 - y1;
        float dist = std::sqrt(dx * dx + dy * dy) / 3.0f;  // Απόσταση για κάθε τμήμα

        float angle = std::atan2(dy, dx);  // Γωνία του τμήματος

        // Υπολογισμός σημείων διαίρεσης
        float xA = x1 + dx / 3;
        float yA = y1 + dy / 3;

        float xB = x1 + 2 * dx / 3;
        float yB = y1 + 2 * dy / 3;

        // Υπολογισμός κορυφής του ισόπλευρου τριγώνου
        float xC = xA + dist * std::cos(angle - M_PI / 3);
        float yC = yA + dist * std::sin(angle - M_PI / 3);

        // Αναδρομή για σχεδίαση των τεσσάρων επιμέρους τμημάτων
        drawKochSegment(x1, y1, xA, yA, depth - 1, brush);
        drawKochSegment(xA, yA, xC, yC, depth - 1, brush);
        drawKochSegment(xC, yC, xB, yB, depth - 1, brush);
        drawKochSegment(xB, yB, x2, y2, depth - 1, brush);
    }
}

/**
 * Συνάρτηση που σχεδιάζει τη βασική χιονονιφάδα ως ισόπλευρο τρίγωνο.
 * @param x, y: Κέντρο του τριγώνου
 * @param sideLength: Μήκος κάθε πλευράς του τριγώνου
 * @param depth: Βάθος της καμπύλης Koch για κάθε πλευρά
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Υπολογίζει τις κορυφές του ισόπλευρου τριγώνου και καλεί την drawKochSegment
 * για κάθε πλευρά με τον ίδιο βαθμό αναδρομής.
 */
void drawSnowflake(float x, float y, float sideLength, int depth, const
graphics::Brush& brush) {
    // Υπολογισμός ύψους ισόπλευρου τριγώνου
    float height = sideLength * std::sqrt(3) / 2;
```

```
    // Υπολογισμός συντεταγμένων κορυφών του τριγώνου
    float x1 = x - sideLength / 2;
    float y1 = y + height / 3;

    float x2 = x + sideLength / 2;
    float y2 = y + height / 3;

    float x3 = x;
    float y3 = y - 2 * height / 3;

    // Σχεδίαση τριών πλευρών του τρίγωνου με χρήση της καμπύλης Koch
    drawKochSegment(x1, y1, x2, y2, depth, brush);
    drawKochSegment(x2, y2, x3, y3, depth, brush);
    drawKochSegment(x3, y3, x1, y1, depth, brush);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG
 *
 * Ρυθμίζει το φόντο και σχεδιάζει το fractal χιονονιφάδας
 * στο κέντρο του παραθύρου, καλώντας τη drawSnowflake.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ρύθμιση βούρτσας για τη χιονονιφάδα
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 1.0f;  // Μπλε χρώμα

    // Κλήση της συνάρτησης σχεδίασης του fractal στο κέντρο
    drawSnowflake(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 300, MAX_DEPTH, brush);
}

/**
 * Κύρια συνάρτηση
 *
 * Δημιουργεί το παράθυρο, ρυθμίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύριο βρόχο
 * για την αναπαράσταση της χιονονιφάδας Koch.
 */
int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Snowflake Curve Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του loop μηνυμάτων
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawKochSegment**: Η αναδρομική συνάρτηση διαίρεσης της κάθε πλευράς σε 4 τμήματα. Κάθε τμήμα αποτελείται από ένα νέο τρίγωνο που προστίθεται πάνω στη γραμμή.
2. **drawSnowflake**: Αυτή η συνάρτηση καλεί τη συνάρτηση `drawKochSegment` για τις τρεις πλευρές του βασικού ισόπλευρου τριγώνου για τη δημιουργία της χιονονιφάδας.
3. **draw**: Καθαρίζει την οθόνη και σχεδιάζει το fractal χρησιμοποιώντας τις προηγούμενες συναρτήσεις.
4. **main**: Ρυθμίζει το παράθυρο, θέτει τη συνάρτηση σχεδίασης και εκκινεί τον βρόχο μηνυμάτων για την προβολή του fractal.

Αυτό το πρόγραμμα δημιουργεί το κλασικό *Koch Snowflake Fractal* με την παραδοσιακή προσέγγιση τριγωνοποίησης.

L System fractal

Για να δημιουργήσουμε ένα *L-System Fractal* χρησιμοποιώντας την SGG Library, μπορούμε να γράψουμε ένα πρόγραμμα το οποίο θα ακολουθεί κανόνες *L-System* για να σχεδιάσει ένα fractal βασισμένο σε μια συμβολοσειρά εντολών. Ένα από τα πιο γνωστά παραδείγματα είναι το *Fractal Tree*, όπου χρησιμοποιούνται οι κανόνες αναδρομής και ανακατεύθυνσης του βραχίονα της χελώνας.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <string>
#include <cmath>
#include <stack>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;
const float ANGLE_INCREMENT = 25.0f;  // Βήμα περιστροφής για το L-System
const int MAX_DEPTH = 5;              // Βάθος επανάληψης του L-System
const float LENGTH = 10.0f;           // Μήκος γραμμής για κάθε κίνηση της χελώνας

/**
```

```
 * Κλάση Turtle για την απεικόνιση σχεδίων με βάση εντολές κινήσεων.
 * Αποθηκεύει την τρέχουσα θέση, γωνία και το χρώμα της γραμμής.
 */
class Turtle {
public:
    float x, y;
    float angle;
    bool penDown;
    graphics::Brush brush;

    /**
     * Κατασκευαστής Turtle που θέτει τις αρχικές συντεταγμένες.
     * Αρχική γωνία: -90 μοίρες (επάνω).
     */
    Turtle(float startX, float startY) : x(startX), y(startY), angle(-90),
penDown(true) {
        brush.fill_color[0] = 0.0f;   // Πράσινο χρώμα
        brush.fill_color[1] = 0.5f;
        brush.fill_color[2] = 0.0f;
    }

    /**
     * Κίνηση της χελώνας προς την κατεύθυνση της τρέχουσας γωνίας.
     * @param distance: Απόσταση κίνησης
     */
    void move(float distance) {
        float newX = x + distance * cos(angle * M_PI / 180.0f);
        float newY = y + distance * sin(angle * M_PI / 180.0f);
        if (penDown) {
            graphics::drawLine(x, y, newX, newY, brush);
        }
        x = newX;
        y = newY;
    }

    /**
     * Περιστροφή της χελώνας κατά μια γωνία.
     * @param degrees: Γωνία περιστροφής σε μοίρες
     */
    void rotate(float degrees) {
        angle += degrees;
    }

    /**
     * Αποθήκευση της τρέχουσας κατάστασης (θέση και γωνία) της χελώνας σε στοίβα.
     * @param stateStack: Η στοίβα που αποθηκεύει την κατάσταση της χελώνας
     */
    void pushState(std::stack<Turtle>& stateStack) {
        stateStack.push(*this);
    }

    /**
     * Επαναφορά της κατάστασης της χελώνας από τη στοίβα.
     * @param stateStack: Η στοίβα που περιέχει τις προηγούμενες καταστάσεις
     */
    void popState(std::stack<Turtle>& stateStack) {
        if (!stateStack.empty()) {
            *this = stateStack.top();
            stateStack.pop();
        }
    }
};

/**
 * Συνάρτηση για τη δημιουργία της συμβολοσειράς του L-System.
```

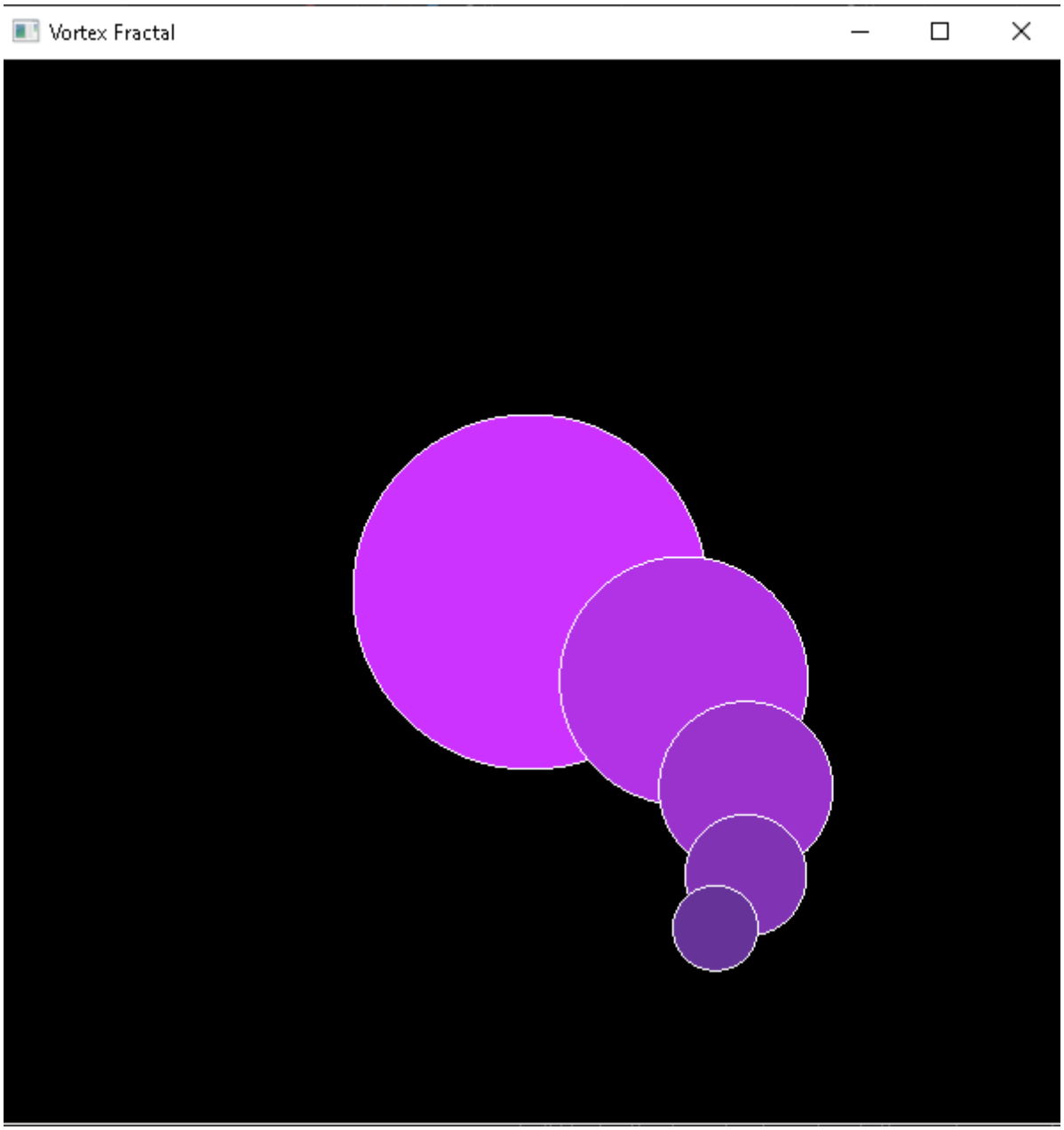
```
 * @param axiom: Το αρχικό στοιχείο της ακολουθίας
 * @param rule: Κανόνας παραγωγής για το L-System
 * @param depth: Βάθος επανάληψης του κανόνα παραγωγής
 * @return Η τελική συμβολοσειρά του L-System
 */
std::string generateLSystem(const std::string& axiom, const std::string& rule, int
depth) {
    std::string current = axiom;
    for (int i = 0; i < depth; ++i) {
        std::string next;
        for (char c : current) {
            if (c == 'F') {
                next += rule;  // Αντικατάσταση 'F' με τον κανόνα
            }
            else {
                next += c;     // Διατήρηση άλλων συμβόλων
            }
        }
        current = next;
    }
    return current;
}

/**
 * Συνάρτηση σχεδίασης του L-System με βάση την ακολουθία που παράγεται.
 * @param turtle: Η χελώνα που σχεδιάζει το fractal
 * @param sequence: Η ακολουθία εντολών που παράγεται από το L-System
 */
void drawLSystem(Turtle& turtle, const std::string& sequence) {
    std::stack<Turtle> stateStack;
    for (char command : sequence) {
        if (command == 'F') {
            turtle.move(LENGTH);  // Μετακίνηση της χελώνας
        }
        else if (command == '+') {
            turtle.rotate(ANGLE_INCREMENT);  // Περιστροφή δεξιά
        }
        else if (command == '-') {
            turtle.rotate(-ANGLE_INCREMENT); // Περιστροφή αριστερά
        }
        else if (command == '[') {
            turtle.pushState(stateStack);    // Αποθήκευση κατάστασης
        }
        else if (command == ']') {
            turtle.popState(stateStack);     // Επαναφορά κατάστασης
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ
 */
void draw() {
    // Καθαρισμός του παραθύρου
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ορισμός των εντολών και των κανόνων του L-System
    std::string axiom = "F";
    std::string rule = "F[+F]F[-F]F";  // Κανόνας παραγωγής για τη δομή του fractal
```

```
    Turtle turtle(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 50);  // Αρχική θέση της χελώνας
    turtle.brush.fill_color[0] = 0.0f;  // Χρώμα χελώνας
    turtle.brush.fill_color[1] = 0.3f;
    turtle.brush.fill_color[2] = 0.1f;

    // Δημιουργία ακολουθίας του L-System και σχεδίαση
    std::string lSystemSequence = generateLSystem(axiom, rule, MAX_DEPTH);
    drawLSystem(turtle, lSystemSequence);
}

/**
 * Κύρια συνάρτηση που αρχικοποιεί το παράθυρο και ξεκινά το πρόγραμμα
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "L-System Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **Turtle Class**: Η κλάση `Turtle` καθορίζει μια χελώνα που κινείται και περιστρέφεται στην οθόνη. Μπορεί να κρατήσει ή να επαναφέρει τη θέση και την κατεύθυνσή της, όπως απαιτείται από τα L-Systems.
2. **generateLSystem**: Αυτή η συνάρτηση δημιουργεί τη συμβολοσειρά του fractal L-System βάσει ενός αξιώματος και ενός κανόνα, για έναν προκαθορισμένο αριθμό επιπέδων.
3. **drawLSystem**: Αυτή η συνάρτηση χρησιμοποιεί τη συμβολοσειρά για να καθοδηγήσει τη χελώνα. Οι εντολές `F`, `+`, `-`, `[`, `]` καθορίζουν τις κινήσεις της χελώνας:
    - `F`: Κίνηση μπροστά.
    - `+`: Περιστροφή δεξιά.
    - `-`: Περιστροφή αριστερά.
    - `[`: Αποθήκευση της τρέχουσας κατάστασης.
    - `]`: Ανάκτηση της προηγούμενης αποθηκευμένης κατάστασης.
4. **draw**: Η βασική συνάρτηση σχεδίασης που παράγει το fractal στο κέντρο της οθόνης.
5. **main**: Δημιουργεί το παράθυρο και θέτει τη συνάρτηση σχεδίασης για το L-System Fractal.

Vortex fractal



Το *Vortex Fractal* είναι ένα εντυπωσιακό γεωμετρικό σχέδιο που δημιουργείται από περιστρεφόμενα σχήματα σε διαφορετικές κλίμακες και αποστάσεις από το κέντρο. Μπορούμε να το υλοποιήσουμε χρησιμοποιώντας την SGG Library με αναδρομική σχεδίαση, όπου κάθε επίπεδο περιστρέφεται και κλιμακώνεται γύρω από ένα κέντρο.

Ακολουθεί ο κώδικας σε SGG Library για τη σχεδίαση του *Vortex Fractal*.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <iostream>
#define M_PI 3.14159265358979323846
```

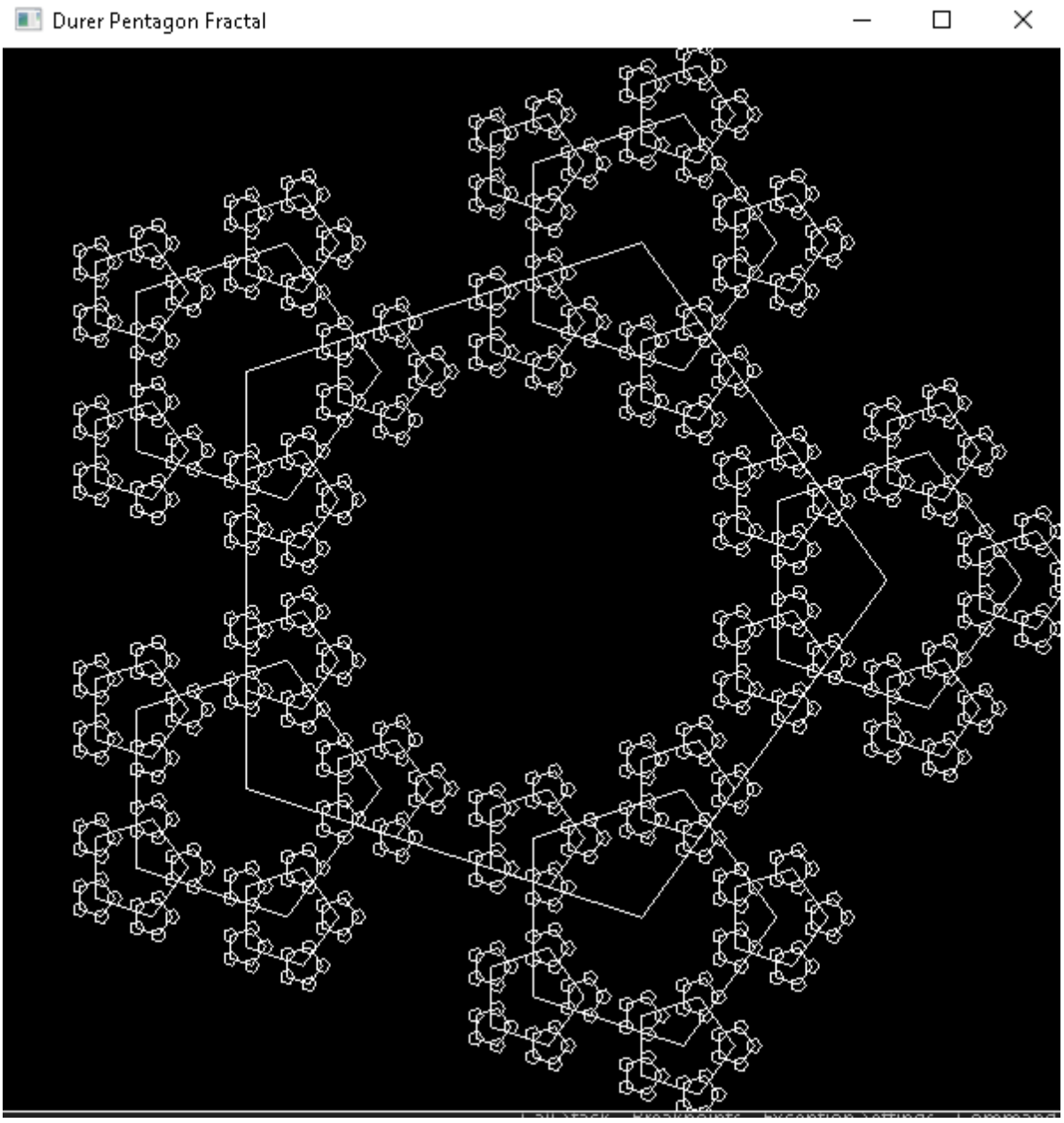
```
// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5;            // Μέγιστο βάθος αναδρομής
const float INITIAL_RADIUS = 100.0f; // Αρχική ακτίνα του πρώτου κύκλου
const float SCALE_FACTOR = 0.7f;     // Παράγοντας κλίμακας για κάθε νέο κύκλο
const float ANGLE_INCREMENT = 30.0f; // Γωνιακή περιστροφή για κάθε νέο κύκλο

/**
 * Συνάρτηση σχεδίασης κύκλου (βασικό στοιχείο του fractal).
 * @param x, y: Συντεταγμένες κέντρου του κύκλου
 * @param radius: Ακτίνα του κύκλου
 * @param brush: Χρώμα σχεδίασης του κύκλου
 *
 * Η συνάρτηση σχεδιάζει έναν κύκλο στο σημείο `(x, y)` με ακτίνα `radius` και χρώμα
που ορίζεται από το `brush`.
 */
void drawCircle(float x, float y, float radius, const graphics::Brush& brush) {
    graphics::drawDisk(x, y, radius, brush);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του Vortex Fractal.
 * @param x, y: Συντεταγμένες κέντρου του τρέχοντος κύκλου
 * @param radius: Ακτίνα του τρέχοντος κύκλου
 * @param depth: Τρέχον βάθος αναδρομής
 * @param angle: Γωνία περιστροφής για τον επόμενο κύκλο
 *
 * Η συνάρτηση σχεδιάζει έναν κύκλο και κατόπιν καλείται αναδρομικά για να
δημιουργήσει έναν μικρότερο κύκλο,
 * περιστρεφόμενο κατά `ANGLE_INCREMENT` μοίρες γύρω από τον προηγούμενο.
 */
void drawVortex(float x, float y, float radius, int depth, float angle) {
    if (depth == 0) return;  // Όριο βάθους αναδρομής

    // Ορισμός του χρώματος για τον τρέχοντα κύκλο
    graphics::Brush brush;
    brush.fill_color[0] = 0.3f + (depth * 0.1f); // Σταδιακή αλλαγή χρώματος
    brush.fill_color[1] = 0.2f;
    brush.fill_color[2] = 0.5f + (depth * 0.1f);
    drawCircle(x, y, radius, brush);

    // Υπολογισμός παραμέτρων για τον επόμενο κύκλο
    float newRadius = radius * SCALE_FACTOR;  // Μείωση ακτίνας για τον επόμενο κύκλο
    float newAngle = angle + ANGLE_INCREMENT; // Αύξηση γωνίας περιστροφής

    // Υπολογισμός συντεταγμένων του επόμενου κύκλου
    float newX = x + radius * cos(newAngle * M_PI / 180.0f);
    float newY = y + radius * sin(newAngle * M_PI / 180.0f);

    // Αναδρομική κλήση για σχεδίαση του επόμενου κύκλου
    drawVortex(newX, newY, newRadius, depth - 1, newAngle);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Αρχικά καθαρίζει το παράθυρο και στη συνέχεια καλεί τη `drawVortex` για να
σχεδιάσει το fractal στο κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
```

```
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κεντρική κλήση για σχεδίαση του vortex fractal
    drawVortex(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, INITIAL_RADIUS, MAX_DEPTH, 0.0f);
}

/**
 * Κύρια συνάρτηση του προγράμματος
 *
 * Αρχικοποιεί το παράθυρο, ρυθμίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύριο βρόχο
 * για τη διατήρηση της λειτουργίας του παραθύρου.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Vortex Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawCircle**: Μια βοηθητική συνάρτηση για σχεδίαση ενός κύκλου σε συγκεκριμένη θέση και ακτίνα.
2. **drawVortex**: Η αναδρομική συνάρτηση που δημιουργεί το *Vortex Fractal*:
    - Σχεδιάζει έναν κύκλο στη θέση `(x, y)` με την καθορισμένη ακτίνα και χρώμα.
    - Υπολογίζει τις συντεταγμένες για τον επόμενο κύκλο σε μικρότερη ακτίνα και μετατοπισμένο γωνιακά.
    - Καλεί τον εαυτό της με μειωμένο βάθος, ώστε να συνεχίσει το μοτίβο μέχρι να φτάσει στη βάση της αναδρομής.
3. **draw**: Η κύρια συνάρτηση σχεδίασης καθαρίζει το φόντο και καλεί την `drawVortex` για να σχεδιάσει το fractal στο κέντρο της οθόνης.
4. **main**: Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης, και ξεκινά το μήνυμα επανάληψης για την εμφάνιση του fractal.

Durer Pentagon Fractal



Το *Dürer Pentagon Fractal* είναι ένα fractal βασισμένο στο πεντάγωνο, το οποίο δημιουργείται διαιρώντας και κλιμακώνοντας το αρχικό πεντάγωνο. Μπορούμε να το σχεδιάσουμε αναδρομικά, όπου κάθε επίπεδο δημιουργεί μικρότερα πεντάγωνα προς το εσωτερικό του αρχικού πενταγώνου.

Παρακάτω είναι ο κώδικας για τη σχεδίαση του *Dürer Pentagon Fractal* χρησιμοποιώντας την SGG Library.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <cmath>
#include <vector>
#define M_PI 3.14159265358979323846
```

```
// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5;          // Μέγιστο βάθος αναδρομής
const float SCALE_FACTOR = 0.38f; // Συντελεστής κλίμακας για κάθε νέο πεντάγωνο

/**
 * Υπολογίζει τις κορυφές ενός πενταγώνου με κέντρο τις συντεταγμένες (x, y).
 * @param x, y: Κέντρο του πενταγώνου
 * @param radius: Ακτίνα από το κέντρο μέχρι τις κορυφές του πενταγώνου
 * @param rotation: Προαιρετική γωνία περιστροφής του πενταγώνου
 * @return: Διάνυσμα ζευγών συντεταγμένων που αντιπροσωπεύουν τις κορυφές του
πενταγώνου
 */
std::vector<std::pair<float, float>> calculatePentagonVertices(float x, float y, float
radius, float rotation = 0) {
    std::vector<std::pair<float, float>> vertices;
    for (int i = 0; i < 5; i++) {
        float angle = rotation + i * 2 * M_PI / 5; // Υπολογισμός γωνίας για κάθε
κορυφή
        float vx = x + radius * cos(angle);        // Συντεταγμένη x της κορυφής
        float vy = y + radius * sin(angle);        // Συντεταγμένη y της κορυφής
        vertices.push_back({ vx, vy });
    }
    return vertices;
}

/**
 * Σχεδιάζει ένα πεντάγωνο με βάση τις κορυφές του.
 * @param vertices: Διάνυσμα ζευγών συντεταγμένων για τις κορυφές του πενταγώνου
 * @param brush: Αντικείμενο graphics::Brush για τον καθορισμό χρώματος σχεδίασης
 *
 * Η συνάρτηση σχεδιάζει ένα πεντάγωνο συνδέοντας τις κορυφές μεταξύ τους.
 */
void drawPentagon(const std::vector<std::pair<float, float>>& vertices, const
graphics::Brush& brush) {
    for (size_t i = 0; i < vertices.size(); i++) {
        size_t next = (i + 1) % vertices.size();  // Υπολογισμός της επόμενης κορυφής
για σύνδεση
        graphics::drawLine(vertices[i].first, vertices[i].second,
vertices[next].first, vertices[next].second, brush);
    }
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του Dürer Pentagon Fractal.
 * @param x, y: Κέντρο του τρέχοντος πενταγώνου
 * @param radius: Ακτίνα του τρέχοντος πενταγώνου
 * @param depth: Τρέχον επίπεδο βάθους
 *
 * Η συνάρτηση σχεδιάζει ένα πεντάγωνο και αναδρομικά καλεί τον εαυτό της για να
δημιουργήσει μικρότερα πεντάγωνα σε κάθε κορυφή του.
 */
void drawDurerPentagonFractal(float x, float y, float radius, int depth) {
    if (depth == 0) return;  // Τερματίζουμε την αναδρομή όταν φτάσουμε στο μέγιστο
βάθος

    // Ρύθμιση του χρώματος ανάλογα με το βάθος για διαφοροποίηση των επιπέδων
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f + 0.1f * depth; // Μεταβολή κόκκινου χρώματος ανά βάθος
    brush.fill_color[1] = 0.4f + 0.1f * depth; // Μεταβολή πράσινου χρώματος
    brush.fill_color[2] = 0.6f - 0.1f * depth; // Μείωση μπλε χρώματος με το βάθος

    // Υπολογισμός και σχεδίαση του κεντρικού πενταγώνου
```

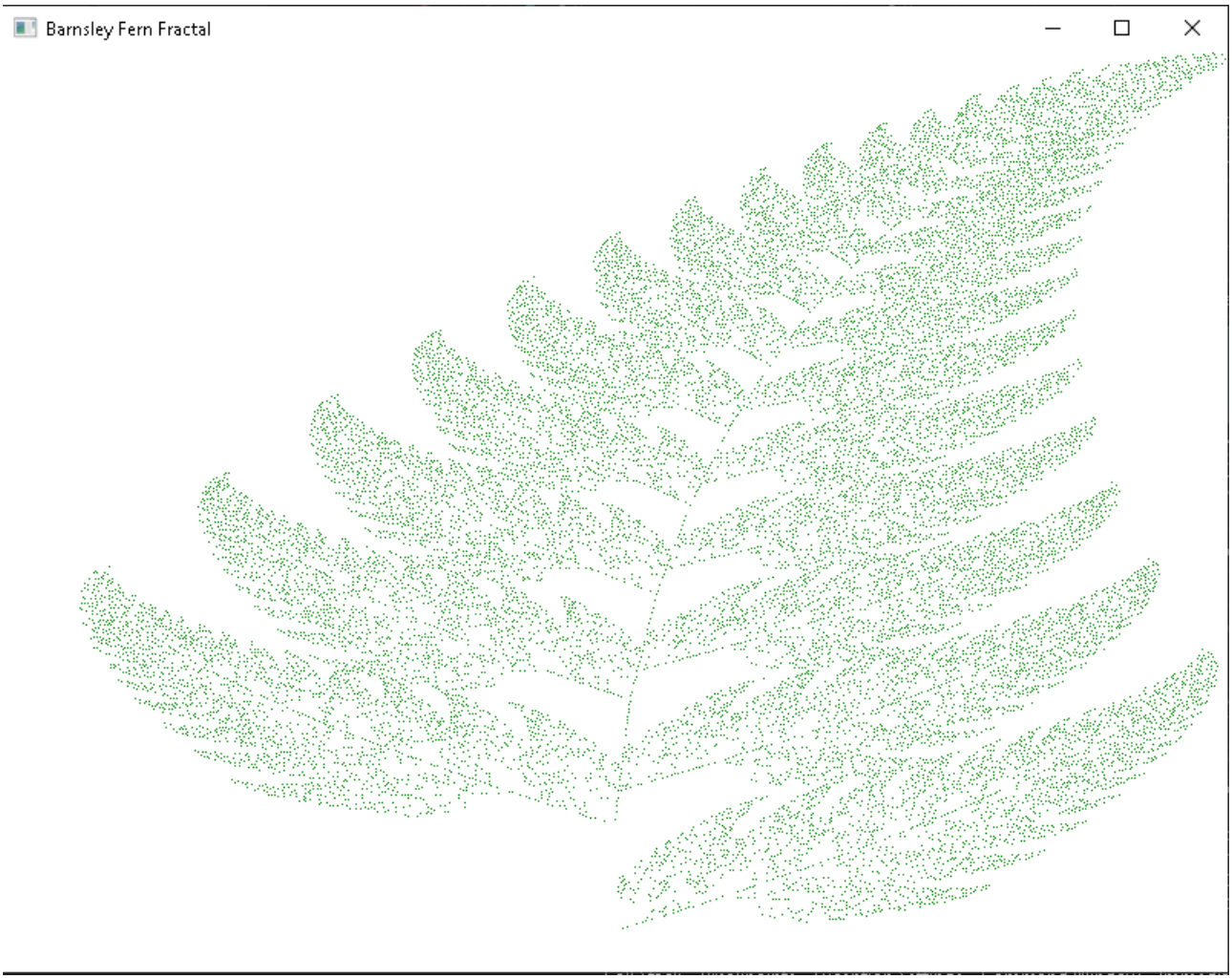
```
    auto vertices = calculatePentagonVertices(x, y, radius);
    drawPentagon(vertices, brush);

    // Αναδρομική κλήση για δημιουργία μικρότερων πενταγώνων σε κάθε κορυφή
    for (auto& vertex : vertices) {
        drawDurerPentagonFractal(vertex.first, vertex.second, radius * SCALE_FACTOR,
depth - 1);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Η συνάρτηση αρχικά καθαρίζει το παράθυρο και στη συνέχεια καλεί τη
drawDurerPentagonFractal
 * για να ξεκινήσει τη διαδικασία δημιουργίας του fractal από το κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κεντρική κλήση της συνάρτησης για τη σχεδίαση του fractal
    drawDurerPentagonFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200, MAX_DEPTH);
}

/**
 * Κύρια συνάρτηση του προγράμματος.
 *
 * Δημιουργεί το παράθυρο, ρυθμίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύριο βρόχο.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Durer Pentagon Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Κώδικα

1. **calculatePentagonVertices**: Υπολογίζει τις κορυφές ενός πενταγώνου με ακτίνα `radius` γύρω από το κέντρο `(x, y)`. Η γωνία `rotation` καθορίζει τη στροφή του πενταγώνου.
2. **drawPentagon**: Σχεδιάζει ένα πεντάγωνο χρησιμοποιώντας τις κορυφές από το `calculatePentagonVertices` και το χρώμα που ορίζεται στο `brush`.
3. **drawDurerPentagonFractal**: Αναδρομική συνάρτηση που σχεδιάζει το fractal. Σχεδιάζει το πεντάγωνο και καλεί τον εαυτό της για να σχεδιάσει μικρότερα πεντάγωνα σε κάθε κορυφή του.
4. **draw**: Καθαρίζει την οθόνη με λευκό φόντο και καλεί την κύρια συνάρτηση `drawDurerPentagonFractal`.

5. **main**: Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά το μήνυμα επανάληψης για να εμφανιστεί το fractal.

Barnsley Fern Fractal



Το Barnsley Fern είναι ένα διάσημο φράκταλ που μοιάζει με φύλλο φτέρης και δημιουργείται με χρήση αναδρομικών συναρτήσεων. Παρακάτω είναι ένα πρόγραμμα σε SGG Library που σχεδιάζει το *Barnsley Fern Fractal* χρησιμοποιώντας τις τέσσερις συναρτήσεις affine transformation που αποτελούν το σύνολο κανόνων του.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <random>

// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;
const int ITERATIONS = 100000;  // Αριθμός επαναλήψεων για σχεδίαση του φράκταλ

/**
 * Κλιμακώνει την τιμή x για να ταιριάζει με τις συντεταγμένες της οθόνης.
 * @param x: Η x συντεταγμένη στο χώρο του φράκταλ
 * @return: Η κλιμακωμένη x συντεταγμένη στο χώρο της οθόνης
 */
float scaleX(float x) {
    return (x + 2.5f) * (WINDOW_WIDTH / 5.0f);
```

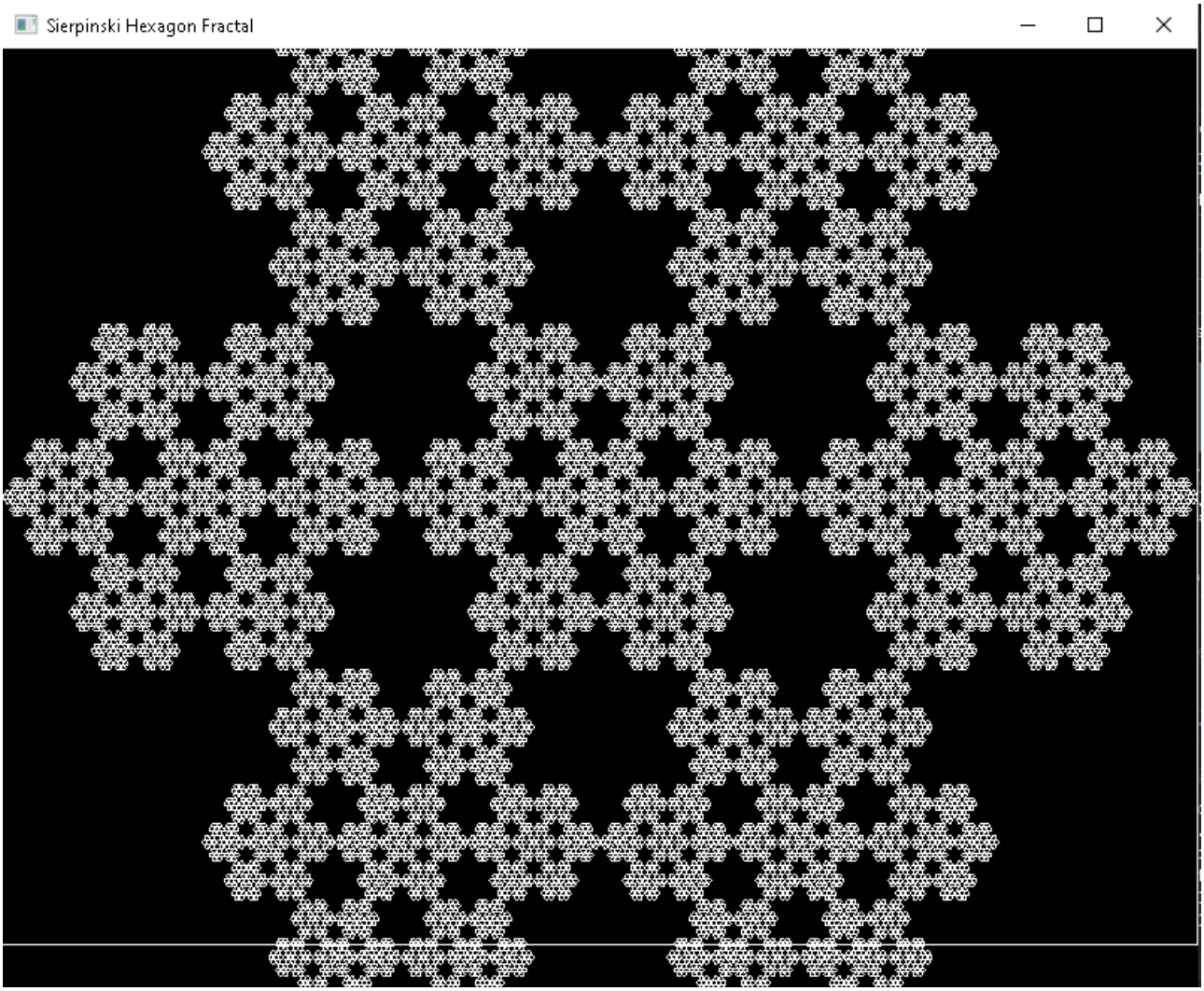
```
}

/**
 * Κλιμακώνει την τιμή y για να ταιριάζει με τις συντεταγμένες της οθόνης.
 * @param y: Η y συντεταγμένη στο χώρο του φράκταλ
 * @return: Η κλιμακωμένη y συντεταγμένη στο χώρο της οθόνης
 */
float scaleY(float y) {
    return WINDOW_HEIGHT - (y * (WINDOW_HEIGHT / 10.0f));
}

/**
 * Συνάρτηση σχεδίασης του φράκταλ Barnsley Fern.
 *
 * Χρησιμοποιεί μια σειρά πιθανοτικών μετασχηματισμών για την ενημέρωση των
συντεταγμένων x και y και σχεδιάζει
 * έναν μικρό κύκλο (pixel) στις αντίστοιχες κλιμακωμένες συντεταγμένες της οθόνης.
 */
void drawFern() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f; // Πράσινο χρώμα
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.0f;

    float x = 0.0f, y = 0.0f; // Αρχικές συντεταγμένες

    // Επαναλαμβανόμενος βρόχος για κάθε επανάληψη
    for (int i = 0; i < ITERATIONS; ++i) {
        float nextX, nextY;
        float r = static_cast<float>(rand()) / RAND_MAX; // Τυχαίος αριθμός μεταξύ 0
και 1

        // Επιλογή μετασχηματισμού με βάση την τιμή του r
        if (r < 0.01f) {
            // F1: Απλός κάθετος μετασχηματισμός (με μικρή πιθανότητα)
            nextX = 0.0f;
            nextY = 0.16f * y;
        }
        else if (r < 0.86f) {
            // F2: Κύριος μετασχηματισμός για τα φύλλα της φτέρης
            nextX = 0.85f * x + 0.04f * y;
            nextY = -0.04f * x + 0.85f * y + 1.6f;
        }
        else if (r < 0.93f) {
            // F3: Δημιουργία του αριστερού φύλλου
            nextX = 0.2f * x - 0.26f * y;
            nextY = 0.23f * x + 0.22f * y + 1.6f;
        }
        else {
            // F4: Δημιουργία του δεξιού φύλλου
            nextX = -0.15f * x + 0.28f * y;
            nextY = 0.26f * x + 0.24f * y + 0.44f;
        }

        // Ενημέρωση των συντεταγμένων
        x = nextX;
        y = nextY;

        // Κλιμάκωση των συντεταγμένων στο χώρο της οθόνης
        float px = scaleX(x);
        float py = scaleY(y);

        // Σχεδίαση του σημείου (pixel) στη θέση (px, py)
        graphics::drawDisk(px, py, 1, brush);
    }
```

```
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Η συνάρτηση καθαρίζει το παράθυρο με λευκό φόντο και κατόπιν καλεί τη συνάρτηση
drawFern()
 * για να σχεδιάσει το φράκταλ της φτέρης.
 */
void draw() {
    // Καθαρισμός παραθύρου με λευκό φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 1.0f;
    bg.fill_color[1] = 1.0f;
    bg.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης σχεδίασης του φράκταλ
    drawFern();
}

/**
 * Κύρια συνάρτηση του προγράμματος.
 *
 * Δημιουργεί το παράθυρο και ξεκινά τον κύριο βρόχο σχεδίασης για την αναπαράσταση
του Barnsley Fern.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Barnsley Fern Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Προγράμματος

1. **Συναρτήσεις scaleX και scaleY**: Αυτές οι συναρτήσεις κλιμακώνουν τις συντεταγμένες ώστε το fractal να τοποθετείται σωστά στο παράθυρο.
2. **drawFern**: Αυτή η συνάρτηση χρησιμοποιεί τις τέσσερις affine transformations του Barnsley Fern για να παράγει το fractal. Σε κάθε επανάληψη, επιλέγεται τυχαία ένας από τους τέσσερις κανόνες με βάση τις πιθανότητες που καθορίζονται από τον ίδιο τον Barnsley. Οι νέες συντεταγμένες υπολογίζονται για κάθε κανόνα και σχεδιάζονται ως μικροί κύκλοι (pixel).
3. **draw**: Καθαρίζει το παράθυρο με λευκό χρώμα και καλεί την `drawFern`.
4. **main**: Δημιουργεί το παράθυρο και ρυθμίζει τη βασική λειτουργία σχεδίασης και έναρξης του `Barnsley Fern Fractal`.

Sierpinski Hexagon Fractal



Το Sierpinski Hexagon είναι ένα fractal βασισμένο στο μοτίβο του εξαγώνου, το οποίο επαναλαμβάνεται με μικρότερα εξάγωνα σε κάθε βήμα. Ακολουθεί ένα ολοκληρωμένο πρόγραμμα που σχεδιάζει το *Sierpinski Hexagon Fractal* χρησιμοποιώντας τη SGG Library.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5; // Μέγιστο βάθος αναδρομής του fractal

/**
 * @brief Υπολογίζει τις συντεταγμένες των κορυφών ενός εξαγώνου με βάση το κέντρο και το μέγεθός του.
 * @param cx Συντεταγμένη x του κέντρου του εξαγώνου
 * @param cy Συντεταγμένη y του κέντρου του εξαγώνου
 * @param size Η απόσταση από το κέντρο προς τις κορυφές
```

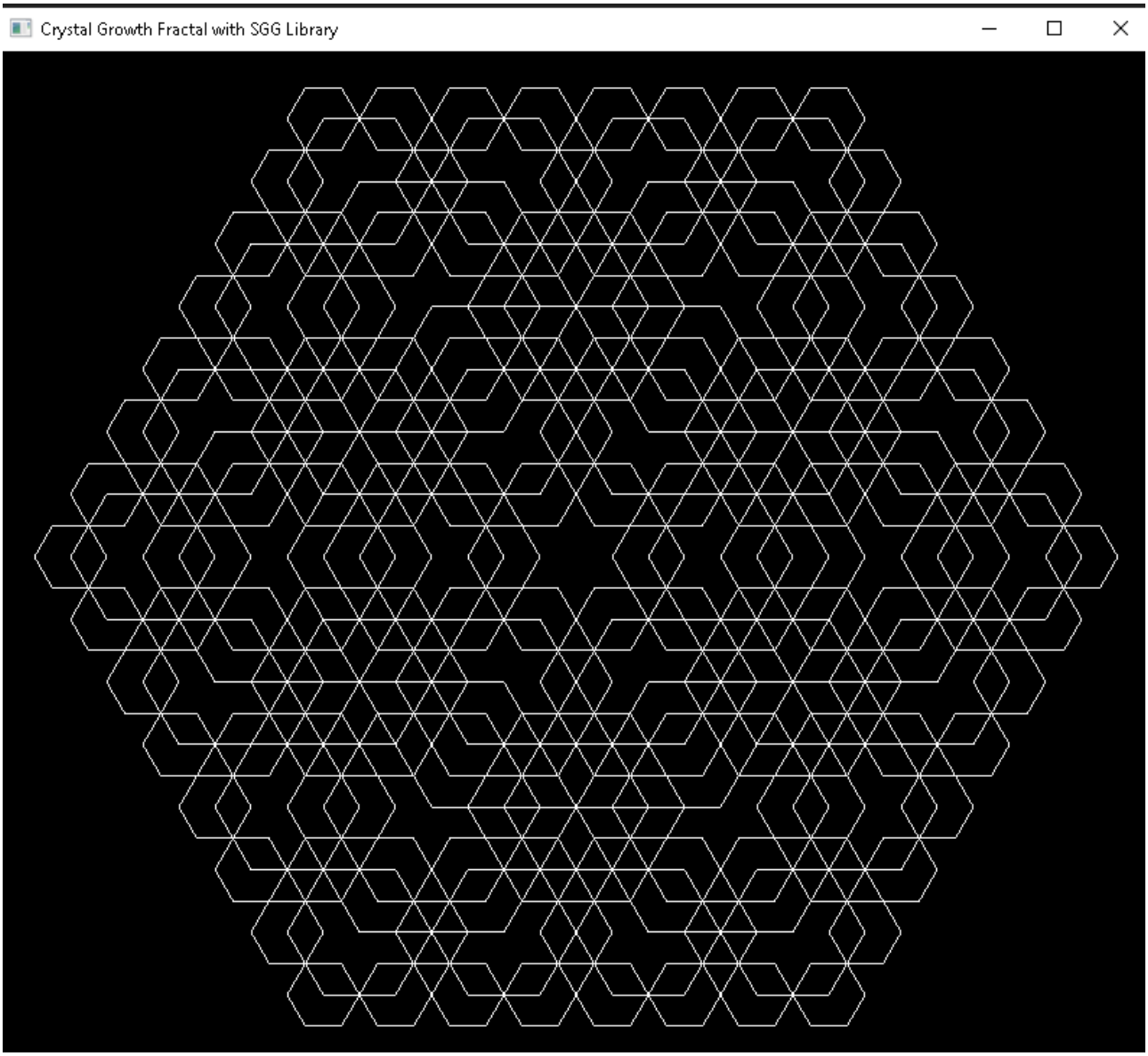
```
 * @return Ένας vector που περιέχει τα ζεύγη συντεταγμένων (x, y) των κορυφών του
εξαγώνου
 */
std::vector<std::pair<float, float>> getHexagonVertices(float cx, float cy, float
size) {
    std::vector<std::pair<float, float>> vertices;
    for (int i = 0; i < 6; ++i) {
        float angle = M_PI / 3 * i; // Κάθε γωνία του εξαγώνου είναι 60 μοίρες
        float x = cx + size * cos(angle);
        float y = cy + size * sin(angle);
        vertices.push_back({ x, y }); // Προσθήκη κορυφής στον vector
    }
    return vertices;
}

/**
 * @brief Σχεδιάζει ένα εξάγωνο συνδέοντας τις κορυφές του.
 * @param vertices Ένας vector που περιέχει τις κορυφές του εξαγώνου
 * @param brush Το αντικείμενο Brush που χρησιμοποιείται για τον καθορισμό του
χρώματος σχεδίασης
 */
void drawHexagon(const std::vector<std::pair<float, float>>& vertices, const
graphics::Brush& brush) {
    for (size_t i = 0; i < vertices.size(); ++i) {
        size_t next = (i + 1) % vertices.size(); // Υπολογισμός επόμενης κορυφής για
σύνδεση γραμμών
        graphics::drawLine(vertices[i].first, vertices[i].second,
vertices[next].first, vertices[next].second, brush);
    }
}

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το Sierpinski Hexagon fractal.
 * @param cx Συντεταγμένη x του κέντρου του εξαγώνου
 * @param cy Συντεταγμένη y του κέντρου του εξαγώνου
 * @param size Η απόσταση από το κέντρο προς τις κορυφές
 * @param depth Το τρέχον βάθος αναδρομής, που μειώνεται κατά 1 σε κάθε κλήση
 */
void drawSierpinskiHexagon(float cx, float cy, float size, int depth) {
    // Όταν το βάθος είναι 0, σχεδιάζεται το τρέχον εξάγωνο και η αναδρομή σταματά
    if (depth == 0) {
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f; // Σκούρο πράσινο
        brush.fill_color[1] = 0.6f;
        brush.fill_color[2] = 0.2f;
        auto vertices = getHexagonVertices(cx, cy, size); // Υπολογισμός κορυφών
εξαγώνου
        drawHexagon(vertices, brush); // Σχεδίαση εξαγώνου
        return;
    }

    float newSize = size / 3.0f; // Μείωση του μεγέθους κάθε επόμενου εξαγώνου στο 1/3
του αρχικού

    // Αναδρομική σχεδίαση των εξαγώνων στις έξι κορυφές γύρω από το κεντρικό
    for (int i = 0; i < 6; ++i) {
        float angle = M_PI / 3 * i;
        float nx = cx + 2 * newSize * cos(angle); // Υπολογισμός x για την κορυφή
        float ny = cy + 2 * newSize * sin(angle); // Υπολογισμός y για την κορυφή
        drawSierpinskiHexagon(nx, ny, newSize, depth - 1); // Αναδρομική κλήση για την
κορυφή
    }

    // Κλήση αναδρομής για το κεντρικό εξάγωνο στο παρόν επίπεδο
    drawSierpinskiHexagon(cx, cy, newSize, depth - 1);
```

```
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ από τη βιβλιοθήκη γραφικών.
 * Καθαρίζει την οθόνη και σχεδιάζει το Sierpinski Hexagon στο κέντρο του παραθύρου.
 */
void draw() {
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Μαύρο φόντο
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Σχεδίαση του Sierpinski Hexagon με κέντρο το κέντρο του παραθύρου
    drawSierpinskiHexagon(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 2,
MAX_DEPTH);
}

/**
 * @brief Σημείο εισόδου του προγράμματος.
 * Δημιουργεί το παράθυρο γραφικών, ρυθμίζει τη συνάρτηση σχεδίασης και ξεκινά τον
κύριο βρόχο.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Sierpinski Hexagon Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Εκκίνηση του κύριου βρόχου μηνυμάτων της βιβλιοθήκης γραφικών
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Προγράμματος

1. **getHexagonVertices**: Δημιουργεί ένα εξάγωνο με βάση το κέντρο `(cx, cy)` και το μέγεθος `size`, επιστρέφοντας τις κορυφές του εξαγώνου.
2. **drawHexagon**: Χρησιμοποιεί τις συντεταγμένες του εξαγώνου από τη `getHexagonVertices` για να σχεδιάσει τα πλευρικά τμήματα.
3. **drawSierpinskiHexagon**: Αναδρομική συνάρτηση που σχεδιάζει το fractal. Κάθε εξάγωνο διαιρείται σε 6 εξάγωνα και ένα κεντρικό μικρότερο, τα οποία σχεδιάζονται αναδρομικά.
4. **draw**: Καθαρίζει το φόντο και καλεί την `drawSierpinskiHexagon` για τη σχεδίαση του fractal.
5. **main**: Δημιουργεί το παράθυρο, ρυθμίζει τη βασική λειτουργία σχεδίασης και ξεκινά τον βρόχο μηνυμάτων.

Crystal Growth Fractal



Το *Crystal Growth Fractal* βασίζεται στη δημιουργία κρυστάλλινων σχημάτων σε μορφή fractal, με πολυγωνικές δομές που αυξάνονται γύρω από ένα κεντρικό σημείο. Παρακάτω δίνεται ένα ολοκληρωμένο πρόγραμμα για τη σχεδίαση του *Crystal Growth Fractal* χρησιμοποιώντας τη SGG Library.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 700;
const int MAX_DEPTH = 4;  // Μέγιστο βάθος αναδρομής για το fractal
```

```cpp
/**
 * @struct Point
 * @brief Απλή δομή που αναπαριστά ένα σημείο σε 2D επίπεδο.
 */
struct Point {
    float x, y;
};

/**
 * @brief Δημιουργεί τις κορυφές ενός κανονικού πολυγώνου με βάση το κέντρο, την
ακτίνα και τον αριθμό πλευρών.
 * @param cx Συντεταγμένη x του κέντρου του πολυγώνου
 * @param cy Συντεταγμένη y του κέντρου του πολυγώνου
 * @param radius Η απόσταση από το κέντρο προς τις κορυφές
 * @param sides Ο αριθμός των πλευρών του πολυγώνου
 * @return Ένας vector που περιέχει τα σημεία των κορυφών του πολυγώνου
 */
std::vector<Point> getPolygonVertices(float cx, float cy, float radius, int sides) {
    std::vector<Point> vertices;
    for (int i = 0; i < sides; ++i) {
        float angle = 2 * M_PI * i / sides; // Υπολογισμός γωνίας για κάθε κορυφή
        vertices.push_back({ cx + radius * cos(angle), cy + radius * sin(angle) });
    }
    return vertices;
}

/**
 * @brief Σχεδιάζει ένα πολύγωνο συνδέοντας διαδοχικά τις κορυφές που δίνονται.
 * @param vertices Ένας vector που περιέχει τα σημεία των κορυφών του πολυγώνου
 * @param brush Το αντικείμενο Brush που χρησιμοποιείται για τον καθορισμό του
χρώματος σχεδίασης
 */
void drawPolygon(const std::vector<Point>& vertices, const graphics::Brush& brush) {
    for (size_t i = 0; i < vertices.size(); ++i) {
        size_t next = (i + 1) % vertices.size(); // Υπολογισμός επόμενης κορυφής για
κλείσιμο πολυγώνου
        graphics::drawLine(vertices[i].x, vertices[i].y, vertices[next].x,
vertices[next].y, brush);
    }
}

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το Crystal Growth Fractal.
 * @param cx Συντεταγμένη x του κέντρου του πολυγώνου
 * @param cy Συντεταγμένη y του κέντρου του πολυγώνου
 * @param radius Η ακτίνα του πολυγώνου
 * @param sides Ο αριθμός των πλευρών του πολυγώνου
 * @param depth Το τρέχον βάθος αναδρομής, που μειώνεται κατά 1 σε κάθε κλήση
 */
void drawCrystalGrowth(float cx, float cy, float radius, int sides, int depth) {
    // Όταν το βάθος είναι 0, σταματάει η αναδρομή
    if (depth == 0) return;

    graphics::Brush brush;
    brush.fill_color[0] = 0.3f + 0.1f * depth; // Αυξάνεται η φωτεινότητα του χρώματος
με το βάθος
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.7f;

    // Υπολογισμός και σχεδίαση του πολυγώνου στο τρέχον επίπεδο
    auto vertices = getPolygonVertices(cx, cy, radius, sides);
    drawPolygon(vertices, brush);

    // Δημιουργία μικρότερων πολυγώνων στις κορυφές με αναδρομή
    float newRadius = radius * 0.5f; // Νέα ακτίνα για τα μικρότερα πολύγωνα
```

```
    for (const auto& vertex : vertices) {
        drawCrystalGrowth(vertex.x, vertex.y, newRadius, sides, depth - 1);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών σε κάθε καρέ.
 * Καθαρίζει την οθόνη και καλεί την αναδρομική συνάρτηση σχεδίασης του Crystal Growth
Fractal.
 */
void draw() {
    // Ρύθμιση μαύρου φόντου για αντίθεση με το fractal
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης για τη σχεδίαση του Crystal Growth Fractal στο κέντρο της
οθόνης
    drawCrystalGrowth(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 4, 6,
MAX_DEPTH);
}

/**
 * @brief Σημείο εισόδου του προγράμματος.
 * Δημιουργεί το παράθυρο γραφικών, ορίζει τη συνάρτηση σχεδίασης και εκκινεί τον
κύριο βρόχο.
 */
int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Crystal Growth Fractal with
SGG Library");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται από τη βιβλιοθήκη σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του κύριου βρόχου μηνυμάτων της βιβλιοθήκης
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Κώδικα

1. **getPolygonVertices**: Δημιουργεί έναν πολυγωνικό δακτύλιο γύρω από το κέντρο `(cx, cy)` με συγκεκριμένη ακτίνα και πλευρές.

2. **drawPolygon**: Σχεδιάζει το πολύγωνο χρησιμοποιώντας τις κορυφές που δημιουργούνται από τη `getPolygonVertices`.

3. **drawCrystalGrowth**: Αναδρομική συνάρτηση που δημιουργεί και σχεδιάζει το fractal. Για κάθε βάθος:

    - Σχεδιάζει ένα πολύγωνο με συγκεκριμένη ακτίνα.
    - Για κάθε κορυφή, καλεί αναδρομικά τη συνάρτηση με μικρότερη ακτίνα και μειωμένο βάθος.
4. **draw**: Καθαρίζει την οθόνη και ξεκινά τη σχεδίαση του fractal από το κέντρο του παραθύρου.

5. **main**: Δημιουργεί το παράθυρο, ρυθμίζει τη λειτουργία σχεδίασης και ξεκινά τη λειτουργία του παραθύρου.

Fractal Hands

Το *Fractal Hands* είναι ένα fractal σχέδιο που δημιουργείται συνήθως με μοτίβα επαναλαμβανόμενων δακτύλων ή σχημάτων χεριών, που επαναλαμβάνονται και περιστρέφονται γύρω από ένα κεντρικό σημείο για να δημιουργήσουν συμμετρική και σύνθετη γεωμετρία.

Παρακάτω είναι ένα παράδειγμα κώδικα σε SGG Library για τη δημιουργία του *Fractal Hands*. Το πρόγραμμα χρησιμοποιεί αναδρομή για να δημιουργήσει τα επαναλαμβανόμενα χέρια σε διαφορετικές θέσεις και με περιστροφές.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <iostream>

#define M_PI 3.14159265358979323846  // Ορισμός του π για ακρίβεια

// Σταθερές διαστάσεων παραθύρου και βάθους fractal
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 700;
const int MAX_DEPTH = 20;  // Μέγιστο βάθος αναδρομής για το fractal

/**
 * @brief Σχεδιάζει ένα "χέρι" με συγκεκριμένο μήκος και πάχος.
 * @param x Η αρχική συντεταγμένη x του χεριού.
 * @param y Η αρχική συντεταγμένη y του χεριού.
```

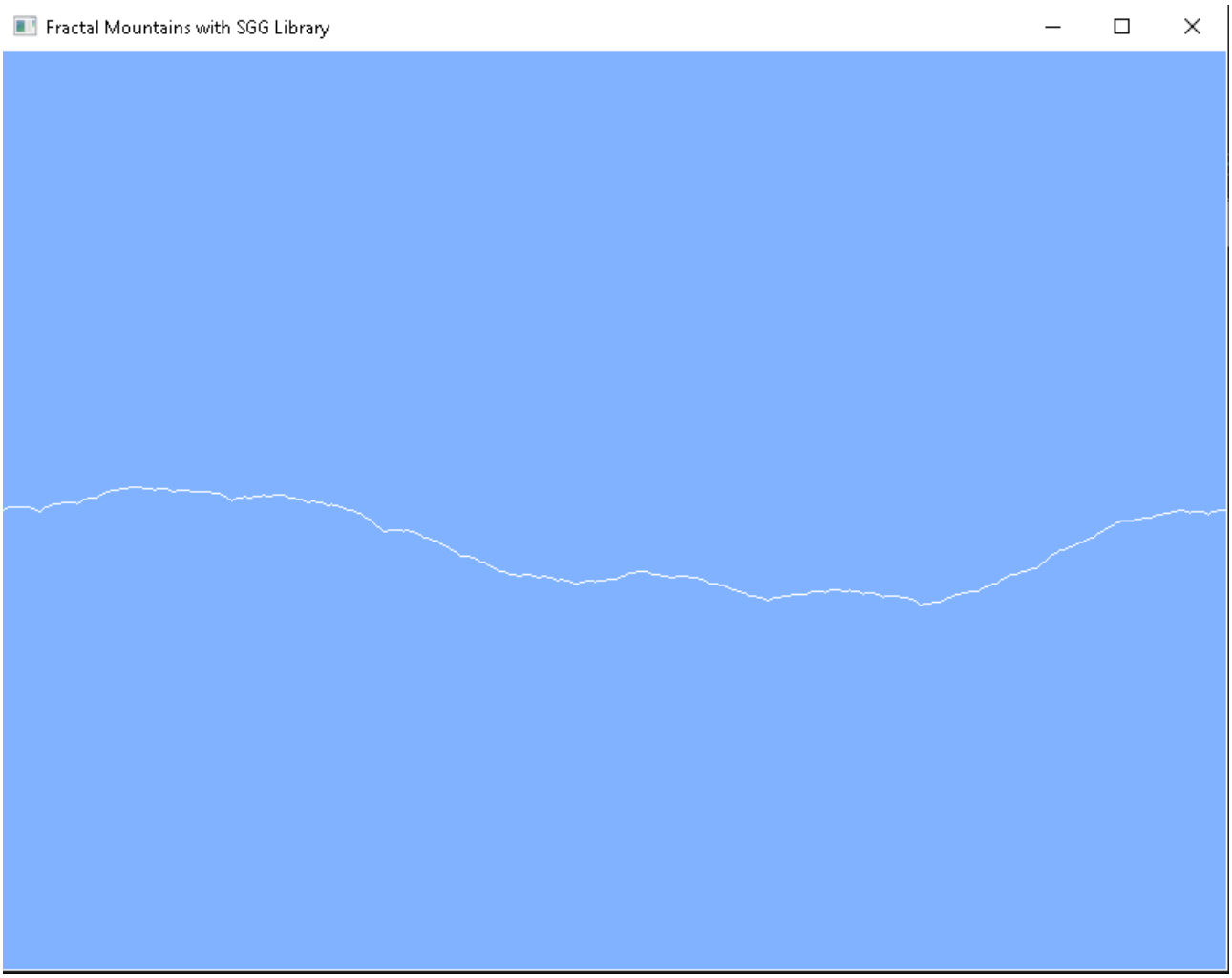
```
 * @param length Το μήκος του χεριού.
 * @param thickness Το πάχος του χεριού.
 * @param brush Το πινέλο (χρώμα) που θα χρησιμοποιηθεί για το χέρι.
 */
void drawHand(float x, float y, float length, float thickness, const graphics::Brush&
brush) {
    // Σχεδιάζουμε το κύριο "χέρι" ως γραμμή από το (x, y) μέχρι (x + length, y)
    graphics::drawLine(x, y, x + length, y, brush);

    // Δύο μικρότερες γραμμές (δάχτυλα) εκτείνονται από το τέλος του "χεριού" προς τα
πάνω και κάτω
    float fingerLength = length / 3;
    for (int i = -1; i <= 1; i += 2) {  // Δύο δάχτυλα: ένα πάνω και ένα κάτω
        float fingerX = x + length;      // Η αρχική x για το δάχτυλο
        float fingerY = y + i * thickness;  // Η αρχική y για το δάχτυλο (ανάλογα με
το i)
        graphics::drawLine(fingerX, fingerY, fingerX + fingerLength, fingerY, brush);
    }
}

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του fractal "Fractal Hands".
 *        Σε κάθε αναδρομική κλήση, δημιουργούνται δύο νέα "χέρια" που εκτείνονται
 *        από το τέλος του κύριου "χεριού".
 * @param x Η συντεταγμένη x του σημείου εκκίνησης για το χέρι.
 * @param y Η συντεταγμένη y του σημείου εκκίνησης για το χέρι.
 * @param length Το μήκος του χεριού.
 * @param thickness Το πάχος του χεριού.
 * @param depth Το τρέχον βάθος αναδρομής. Όταν φτάσει το μηδέν, η αναδρομή
τερματίζεται.
 * @param angle Η γωνία εκκίνησης για τη σχεδίαση των νέων "χεριών".
 */
void drawFractalHands(float x, float y, float length, float thickness, int depth,
float angle) {
    // Αν έχουμε φτάσει στο τέλος της αναδρομής, τερματίζουμε τη σχεδίαση
    if (depth == 0) return;

    // Ορισμός του πινέλου (χρώματος) για το τρέχον βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f + 0.2f * depth; // Προοδευτική αλλαγή κόκκινου
    brush.fill_color[1] = 0.5f;                // Σταθερό πράσινο
    brush.fill_color[2] = 0.7f;                // Σταθερό μπλε

    // Σχεδίαση του αρχικού χεριού
    drawHand(x, y, length, thickness, brush);

    // Υπολογισμός νέου μήκους και πάχους για τα επόμενα "χέρια"
    float newLength = length * 0.6f;
    float newThickness = thickness * 0.6f;

    // Δύο νέα "χέρια" εκτείνονται από το τέλος του τρέχοντος χεριού προς τα αριστερά
και δεξιά
    for (int i = -1; i <= 1; i += 2) {
        float newAngle = angle + i * 45;  // Γωνία περιστροφής: -45 ή +45 μοίρες
        float newX = x + length * cos(angle * M_PI / 180);  // Νέα x συντεταγμένη για
αναδρομή
        float newY = y + length * sin(angle * M_PI / 180);  // Νέα y συντεταγμένη για
αναδρομή

        // Αλλαγή προσανατολισμού και αναδρομική σχεδίαση του επόμενου χεριού
        graphics::setOrientation(newAngle);
        drawFractalHands(newX, newY, newLength, newThickness, depth - 1, newAngle);
    }

    // Επαναφορά της περιστροφής
```

```
    graphics::resetPose();
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
 *        Αυτή η συνάρτηση καθαρίζει τον καμβά και σχεδιάζει το fractal χεριών
 *        από το κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός καμβά με μαύρο χρώμα
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης για σχεδίαση του fractal hands ξεκινώντας από το κέντρο
του παραθύρου
    drawFractalHands(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 5, MAX_DEPTH, 0);
}

/**
 * @brief Η κύρια συνάρτηση του προγράμματος που αρχικοποιεί το παράθυρο
 *        και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση του προγράμματος.
 */
int main() {
    // Δημιουργία παραθύρου με διαστάσεις και τίτλο
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Fractal Hands with SGG
Library");

    // Καθορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων για να συνεχιστεί η ανανέωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Κώδικα

1. **drawHand**: Σχεδιάζει μια γραμμή για το "χέρι" και δύο μικρότερες γραμμές στο τέλος της, που αντιπροσωπεύουν "δάχτυλα."
2. **drawFractalHands**: Αναδρομική συνάρτηση για τη σχεδίαση του fractal. Κάθε χέρι δημιουργεί δύο νέα χέρια, το καθένα με μικρότερο μήκος και σε διαφορετική γωνία από το προηγούμενο.
3. **draw**: Καθαρίζει τον καμβά και ξεκινά τη σχεδίαση του fractal από το κέντρο του παραθύρου.
4. **main**: Δημιουργεί το παράθυρο, ρυθμίζει τη λειτουργία σχεδίασης και ξεκινά τη βρόχο του παραθύρου.

Fractal Mountains



Το fractal mountains είναι ένα δημοφιλές φράκταλ που δημιουργείται από την αναδρομική διαίρεση μιας γραμμής σε μικρότερες γραμμές, με τυχαίες παραμορφώσεις ύψους που προσομοιώνουν τις ανωμαλίες και τις κορυφογραμμές των βουνών. Αυτό το φράκταλ μπορεί να δημιουργηθεί χρησιμοποιώντας αναδρομικές συναρτήσεις.

Παρακάτω είναι ένα ολοκληρωμένο πρόγραμμα σε SGG Library που σχεδιάζει ένα "fractal mountains" με χρήση της τυχαιότητας και αναδρομής:

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <cstdlib>
#include <ctime>

// Σταθερές για τη διαμόρφωση παραθύρου και fractal παραμέτρων
const int WINDOW_WIDTH = 800;           // Πλάτος παραθύρου
const int WINDOW_HEIGHT = 600;          // Ύψος παραθύρου
const int MAX_DEPTH = 8;                // Το μέγιστο βάθος αναδρομής για το fractal
const float DISPLACEMENT = 80.0f;       // Αρχική τυχαία μετατόπιση του ύψους για τα
βουνά
```

```
/**
 * @brief Σχεδιάζει ένα τμήμα του fractal βουνού.
 *        Κάθε τμήμα αναλύεται σε μικρότερα τμήματα με τυχαία μετατόπιση ύψους στο
μέσο σημείο.
 * @param x1 Η αρχική συντεταγμένη x του τμήματος.
 * @param y1 Η αρχική συντεταγμένη y του τμήματος.
 * @param x2 Η τελική συντεταγμένη x του τμήματος.
 * @param y2 Η τελική συντεταγμένη y του τμήματος.
 * @param depth Το τρέχον βάθος αναδρομής.
 * @param displacement Η τυχαία μετατόπιση ύψους για το τρέχον επίπεδο βάθους.
 */
void drawFractalMountain(float x1, float y1, float x2, float y2, int depth, float
displacement) {
    if (depth == 0) {
        // Όταν το βάθος είναι 0, σχεδιάζουμε μια γραμμή που συνδέει τα δύο σημεία
(x1, y1) και (x2, y2)
        graphics::Brush brush;
        brush.fill_color[0] = 0.4f;  // Χρώμα για το βουνό (σκούρο καφέ-γκρι)
        brush.fill_color[1] = 0.3f;
        brush.fill_color[2] = 0.3f;
        graphics::drawLine(x1, y1, x2, y2, brush);  // Σχεδίαση γραμμής
    }
    else {
        // Υπολογισμός του μέσου σημείου με τυχαία μετατόπιση ύψους
        float midX = (x1 + x2) / 2;   // Μέσο σημείο στην x
        float midY = (y1 + y2) / 2 + (rand() % int(displacement * 2)) - displacement;
// Μέσο σημείο στην y με τυχαία μετατόπιση

        // Αναδρομική κλήση για τα δύο νέα τμήματα με μειωμένη μετατόπιση
        drawFractalMountain(x1, y1, midX, midY, depth - 1, displacement / 2);
        drawFractalMountain(midX, midY, x2, y2, depth - 1, displacement / 2);
    }
}

/**
 * @brief Η κύρια συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
 *        Σχεδιάζει το φόντο του ουρανού και στη συνέχεια το fractal βουνό.
 */
void draw() {
    // Καθαρισμός φόντου (ουρανός)
    graphics::Brush skyBrush;
    skyBrush.fill_color[0] = 0.5f;  // Ανοιχτό γαλάζιο χρώμα
    skyBrush.fill_color[1] = 0.7f;
    skyBrush.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, skyBrush);

    // Αρχικές συντεταγμένες για τη γραμμή του βουνού, από την αριστερή προς την δεξιά
άκρη του παραθύρου
    float x1 = 0;
    float y1 = WINDOW_HEIGHT / 2;  // Αρχικό ύψος του βουνού στο κέντρο του παραθύρου
    float x2 = WINDOW_WIDTH;
    float y2 = WINDOW_HEIGHT / 2;

    // Κλήση της αναδρομικής συνάρτησης για σχεδίαση του fractal βουνού
    drawFractalMountain(x1, y1, x2, y2, MAX_DEPTH, DISPLACEMENT);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *        Αρχικοποιεί το παράθυρο και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Αρχικοποίηση της γεννήτριας τυχαίων αριθμών
```

```
    srand(static_cast<unsigned>(time(0)));

    // Δημιουργία παραθύρου με τίτλο
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Fractal Mountains with SGG
Library");

    // Καθορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων
    graphics::startMessageLoop();

    return 0;
}
```

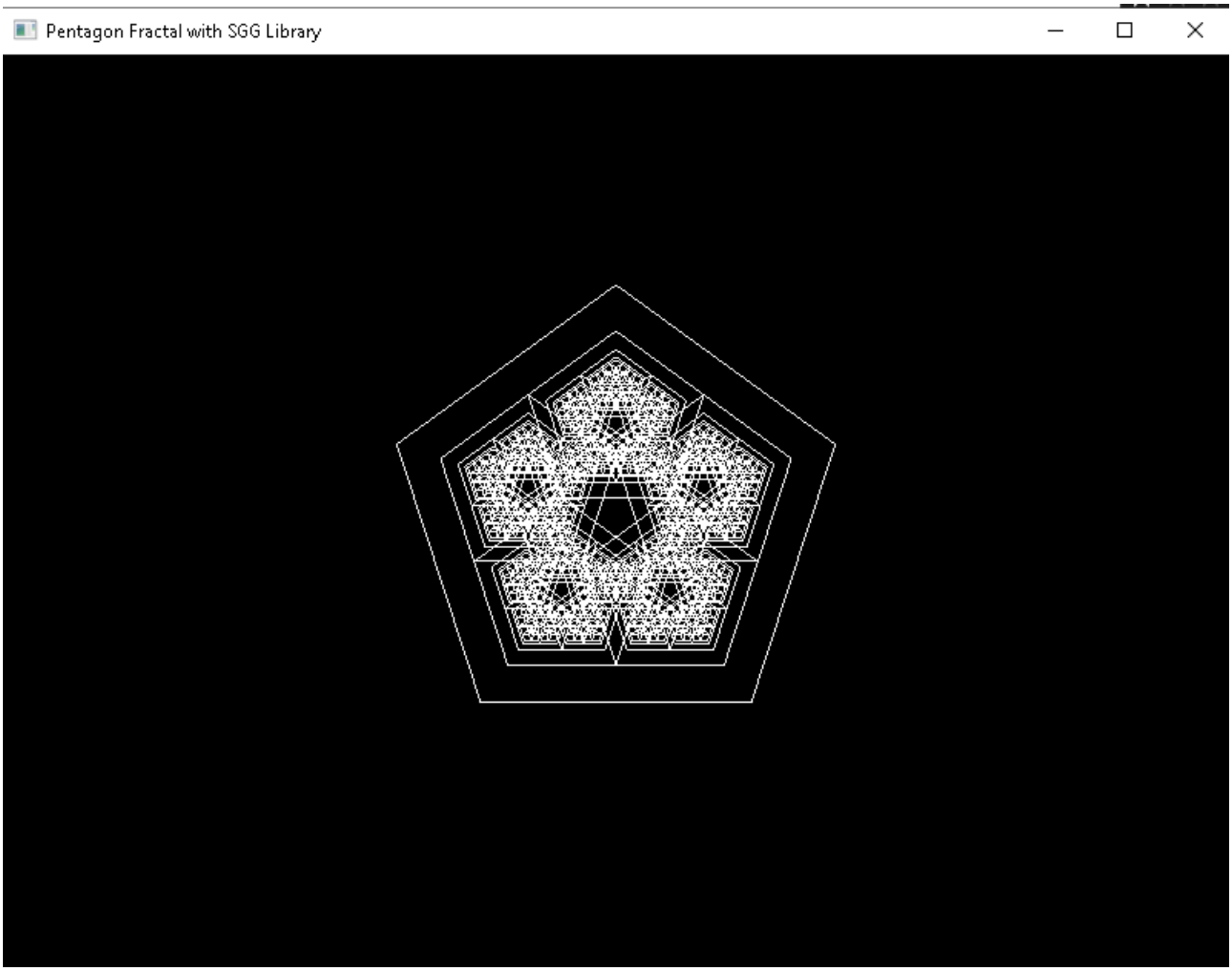
Περιγραφή Κώδικα

1. **drawFractalMountain**: Η συνάρτηση που δημιουργεί τα βουνά χρησιμοποιώντας αναδρομή. Σχεδιάζει μια γραμμή μεταξύ δύο σημείων, η οποία χωρίζεται σε δύο μικρότερες γραμμές με τυχαία μετατόπιση στο μέσο της απόστασης, δημιουργώντας την ψευδαίσθηση των βουνών.
2. **draw**: Ορίζει το φόντο του ουρανού και ξεκινά τη σχεδίαση των βουνών από την αριστερή άκρη του παραθύρου.
3. **main**: Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και εκκινεί τη βρόχο μηνυμάτων.

Σημειώσεις

- Το `MAX_DEPTH` καθορίζει την αναδρομική πολυπλοκότητα. Μεγαλύτερες τιμές αυξάνουν την πολυπλοκότητα του βουνού, ενώ μικρότερες τιμές κάνουν το σχέδιο πιο απλό.
- Το `DISPLACEMENT` καθορίζει το ύψος των κορυφογραμμών.

Pentagon Fractal



Το pentagon fractal είναι ένα ενδιαφέρον φράκταλ που δημιουργείται με την τοποθέτηση μικρότερων πενταγώνων στα άκρα ενός μεγαλύτερου πενταγώνου και την αναδρομή αυτής της διαδικασίας για κάθε νέο πεντάγωνο. Παρακάτω παρατίθεται ο κώδικας για την υλοποίηση του fractal pentagon σε SGG Library.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

// Σταθερές παραμέτρων παραθύρου και fractal
const int WINDOW_WIDTH = 800;      // Πλάτος παραθύρου
const int WINDOW_HEIGHT = 600;     // Ύψος παραθύρου
const int MAX_DEPTH = 5;           // Μέγιστο βάθος αναδρομής του fractal

/**
 * @brief Σχεδιάζει ένα πεντάγωνο κεντραρισμένο στις συντεταγμένες (x, y).
 * @param x Η συντεταγμένη x του κέντρου του πενταγώνου.
 * @param y Η συντεταγμένη y του κέντρου του πενταγώνου.
 * @param size Το μέγεθος (μήκος πλευράς) του πενταγώνου.
 * @param brush Το χρώμα της γραμμής για τη σχεδίαση.
 */
```

```
void drawPentagon(float x, float y, float size, graphics::Brush& brush) {
    // Το πεντάγωνο έχει γωνίες κάθε 72 μοίρες (2π/5 ακτίνια)
    float angleIncrement = 2 * M_PI / 5;
    float angle = M_PI / 2;  // Ξεκινάμε από την κορυφή

    // Αρχικές συντεταγμένες για την πρώτη κορυφή
    float prevX = x + size * cos(angle);
    float prevY = y - size * sin(angle);

    // Σχεδίαση των πλευρών του πενταγώνου
    for (int i = 1; i <= 5; i++) {
        angle += angleIncrement;
        float newX = x + size * cos(angle);
        float newY = y - size * sin(angle);
        graphics::drawLine(prevX, prevY, newX, newY, brush);
        prevX = newX;
        prevY = newY;
    }
}

/**
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του fractal pentagon.
 * @param x Η συντεταγμένη x του κέντρου του πενταγώνου.
 * @param y Η συντεταγμένη y του κέντρου του πενταγώνου.
 * @param size Το μέγεθος (μήκος πλευράς) του πενταγώνου.
 * @param depth Το τρέχον βάθος αναδρομής (μειώνεται σε κάθε κλήση).
 */
void drawFractalPentagon(float x, float y, float size, int depth) {
    if (depth == 0) return;  // Βάση αναδρομής: όταν depth = 0, τερματίζουμε την
αναδρομή

    // Ορισμός χρώματος που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f * depth;  // Ρύθμιση χρώματος ανά επίπεδο βάθους
    brush.fill_color[1] = 0.2f * depth;
    brush.fill_color[2] = 0.3f * depth;

    // Σχεδίαση του κεντρικού πενταγώνου
    drawPentagon(x, y, size, brush);

    // Τοποθέτηση και αναδρομική κλήση για μικρότερα πεντάγωνα στις γωνίες
    float angleIncrement = 2 * M_PI / 5;
    float angle = M_PI / 2;

    for (int i = 0; i < 5; i++) {
        float newX = x + size * cos(angle) / 2.5;
        float newY = y - size * sin(angle) / 2.5;

        // Κλήση για το νέο πεντάγωνο με μειωμένο μέγεθος και βάθος
        drawFractalPentagon(newX, newY, size / 2.5, depth - 1);
        angle += angleIncrement;
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
 *        Σχεδιάζει το φόντο και το fractal pentagon στο κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός φόντου με μαύρο χρώμα
    graphics::Brush bgBrush;
    bgBrush.fill_color[0] = 0.0f;
    bgBrush.fill_color[1] = 0.0f;
    bgBrush.fill_color[2] = 0.0f;
```

```
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bgBrush);

    // Κλήση της συνάρτησης σχεδίασης fractal pentagon
    drawFractalPentagon(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, MAX_DEPTH);
}

/**
 * @brief Η κύρια συνάρτηση του προγράμματος.
 *        Αρχικοποιεί το παράθυρο και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου και εκκίνηση
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Pentagon Fractal with SGG
Library");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου σχεδίασης
    graphics::startMessageLoop();

    return 0;
}
```

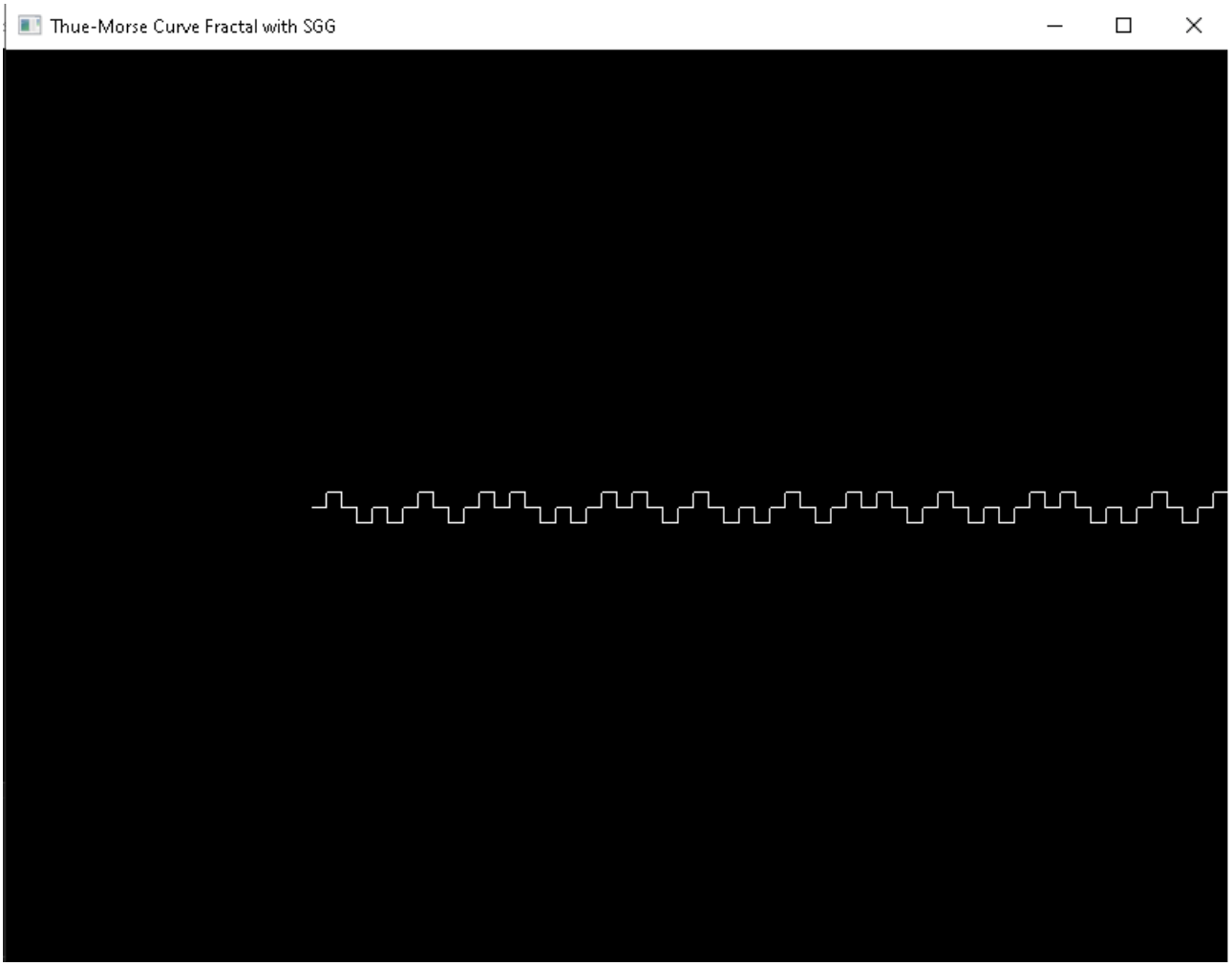
Επεξήγηση Κώδικα

1. **drawPentagon**: Σχεδιάζει ένα κανονικό πεντάγωνο με κέντρο `(x, y)` και πλευρά `size` χρησιμοποιώντας γραμμές για να συνδέσει τις κορυφές του.
2. **drawFractalPentagon**: Αναδρομική συνάρτηση που καλεί τη `drawPentagon` για να σχεδιάσει το αρχικό πεντάγωνο και, στη συνέχεια, υπολογίζει τις θέσεις για πέντε μικρότερα πεντάγωνα γύρω από αυτό, μειώνοντας το `depth` μέχρι να φτάσει στο μηδέν.
3. **draw**: Σχεδιάζει το υπόβαθρο και ξεκινά το fractal pentagon από το κέντρο του παραθύρου.
4. **main**: Ορίζει τις διαστάσεις παραθύρου, τη συνάρτηση σχεδίασης και ξεκινά τη βρόχο μηνυμάτων.

Σημειώσεις

- **MAX_DEPTH**: Το βάθος της αναδρομής επηρεάζει την πολυπλοκότητα του φράκταλ. Υψηλότερες τιμές θα αυξήσουν τη λεπτομέρεια και την πυκνότητα του fractal.
- **Χρώματα**: Οι αποχρώσεις στο `brush` αλλάζουν με το βάθος για να δημιουργήσουν οπτική αντίθεση στα επίπεδα του fractal.

Thue-Morse Curve



Η Thue-Morse καμπύλη είναι ένα φράκταλ που βασίζεται στην ακολουθία Thue-Morse, μια ακολουθία δυαδικών ψηφίων που δεν περιέχει επαναλήψεις και είναι κατάλληλη για δημιουργία καμπυλών με ενδιαφέρουσες συμμετρίες και σχήματα. Σε αυτήν την υλοποίηση, η ακολουθία Thue-Morse θα χρησιμοποιηθεί για να ελέγξει την περιστροφή και την κατεύθυνση σε κάθε βήμα της σχεδίασης.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <vector>
#define M_PI 3.14159265358979323846
// Σταθερές παραμέτρων παραθύρου και βάθους
const int WINDOW_WIDTH = 800;     // Πλάτος παραθύρου
const int WINDOW_HEIGHT = 600;    // Ύψος παραθύρου
const int DEPTH = 15;             // Βάθος της ακολουθίας Thue-Morse

/**
 * @brief Γεννάει την ακολουθία Thue-Morse μέχρι το δεδομένο βάθος.
 * @param depth Το βάθος για την ακολουθία Thue-Morse.
 * @return Επιστρέφει έναν vector<int> με την ακολουθία Thue-Morse.
 */
std::vector<int> generateThueMorseSequence(int depth) {
```

```
    std::vector<int> sequence = { 0 };  // Αρχική ακολουθία με το πρώτο στοιχείο
    for (int i = 0; i < depth; i++) {
        // Δημιουργία αντιγράφου της ακολουθίας και αντιστροφή ψηφίων
        std::vector<int> temp = sequence;
        for (int j = 0; j < temp.size(); j++) {
            temp[j] = 1 - temp[j];  // Αντιστροφή του ψηφίου (0 -> 1, 1 -> 0)
        }
        sequence.insert(sequence.end(), temp.begin(), temp.end());  // Συνένωση
ακολουθιών
    }
    return sequence;
}

/**
 * @brief Σχεδιάζει την καμπύλη Thue-Morse βασισμένη στην ακολουθία περιστροφών που
δημιουργείται.
 * @param startX Αρχική συντεταγμένη x.
 * @param startY Αρχική συντεταγμένη y.
 * @param length Το μήκος κάθε γραμμής.
 * @param angle Η αρχική γωνία σχεδίασης.
 * @param sequence Η ακολουθία Thue-Morse που καθορίζει τις περιστροφές.
 */
void drawThueMorseCurve(float startX, float startY, float length, float angle, const
std::vector<int>& sequence) {
    // Θέση και γωνία για τη σχεδίαση
    float x = startX;
    float y = startY;
    float currentAngle = angle;

    // Ορισμός χρώματος γραμμής
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f; // Μαύρο χρώμα
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f;

    // Σχεδίαση της καμπύλης σύμφωνα με την ακολουθία
    for (int i = 0; i < sequence.size(); i++) {
        // Υπολογισμός των συντεταγμένων του επόμενου σημείου
        float newX = x + length * cos(currentAngle * M_PI / 180.0f);
        float newY = y + length * sin(currentAngle * M_PI / 180.0f);
        graphics::drawLine(x, y, newX, newY, brush);  // Σχεδίαση γραμμής από το (x,
y) στο (newX, newY)

        // Αναβάθμιση θέσης για το επόμενο σημείο
        x = newX;
        y = newY;

        // Καθορισμός περιστροφής σύμφωνα με την ακολουθία
        if (sequence[i] == 1) {
            currentAngle += 90;  // Περιστροφή δεξιά κατά 90 μοίρες
        }
        else {
            currentAngle -= 90;  // Περιστροφή αριστερά κατά 90 μοίρες
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
 *        Σχεδιάζει το φόντο και την καμπύλη Thue-Morse.
 */
void draw() {
    // Καθαρισμός φόντου
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Λευκό φόντο
```

```
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Δημιουργία ακολουθίας Thue-Morse
    std::vector<int> thueMorseSequence = generateThueMorseSequence(DEPTH);

    // Σχεδίαση της καμπύλης από την αριστερή πλευρά του παραθύρου
    drawThueMorseCurve(WINDOW_WIDTH / 4, WINDOW_HEIGHT / 2, 10, 0, thueMorseSequence);
}

/**
 * @brief Η κύρια συνάρτηση του προγράμματος.
 *        Αρχικοποιεί το παράθυρο και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου και εκκίνηση
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Thue-Morse Curve Fractal with
SGG");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου σχεδίασης
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **generateThueMorseSequence**: Αυτή η συνάρτηση δημιουργεί την ακολουθία Thue-Morse αναδρομικά. Ξεκινά με το `{0}`, και κάθε επανάληψη προσθέτει αντίστροφες τιμές στην ακολουθία.

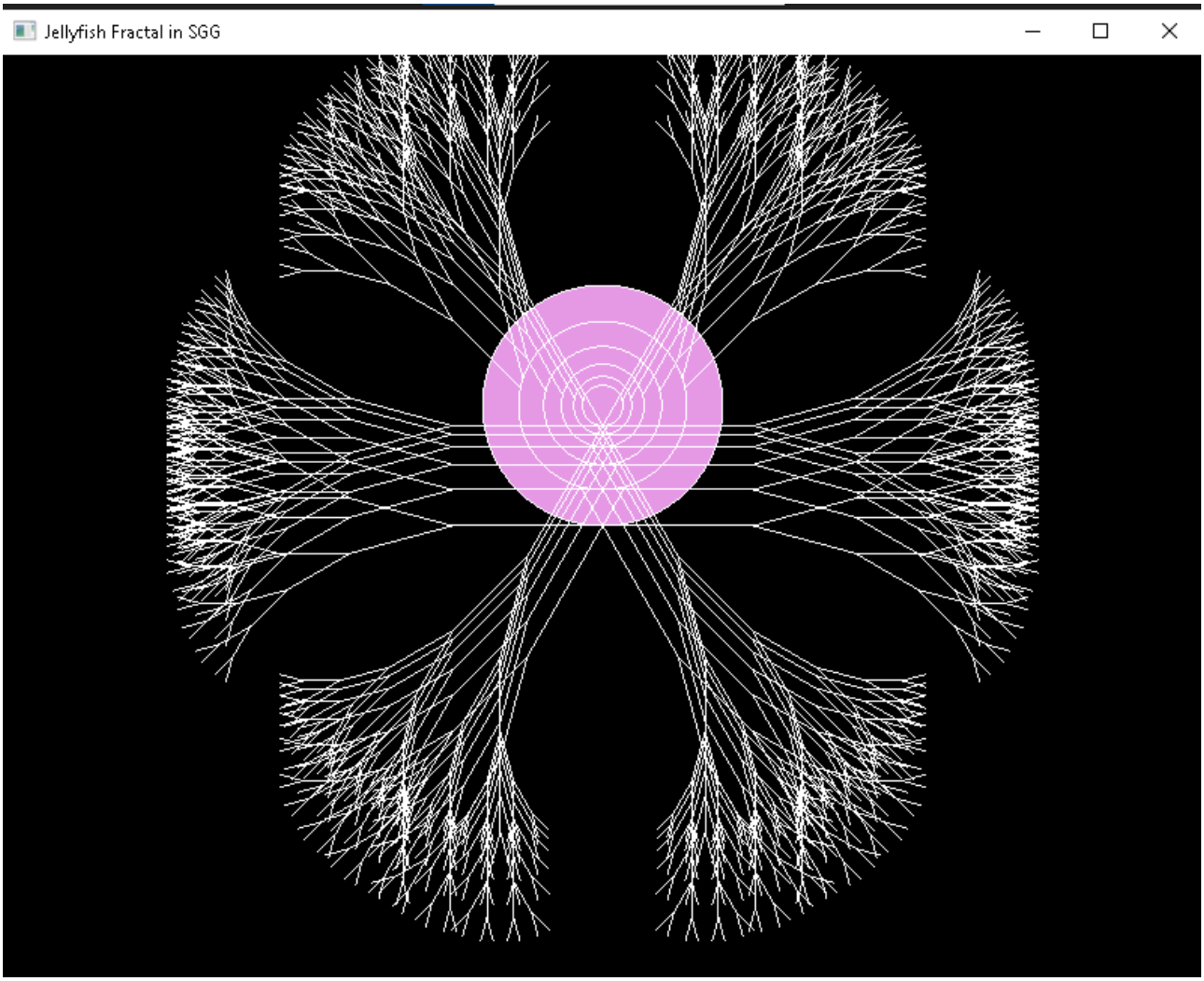
2. **drawThueMorseCurve**: Αναλαμβάνει τη σχεδίαση της καμπύλης με βάση την ακολουθία Thue-Morse. Για κάθε στοιχείο στην ακολουθία, σχεδιάζει μια γραμμή μήκους `length` και κατευθύνεται αριστερά ή δεξιά ανάλογα με το τρέχον στοιχείο.

3. **draw**: Καθαρίζει την οθόνη, δημιουργεί την ακολουθία Thue-Morse, και καλεί τη συνάρτηση `drawThueMorseCurve` για να σχεδιάσει την καμπύλη στο κέντρο του παραθύρου.

Προσαρμογές

- **DEPTH**: Μεγαλύτερη τιμή για την DEPTH θα προσθέσει λεπτομέρεια και πολυπλοκότητα στο φράκταλ.
- **length**: Μειώνοντας το `length`, αυξάνεται η πυκνότητα των γραμμών, επιτρέποντας την προβολή περισσότερων επιπέδων του fractal στο ίδιο παράθυρο.

Jellyfish Fractal



Το **Jellyfish fractal** μπορεί να παραχθεί συνδυάζοντας καμπύλες και επαναληπτικές "κλωστές" που μοιάζουν με τα πλοκάμια μιας μέδουσας. Παρακάτω είναι ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη SGG για να δημιουργήσει το Jellyfish fractal:

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>

#define M_PI 3.14159265358979323846

// Ρυθμίσεις για το παράθυρο
const int WINDOW_WIDTH = 800;      // Πλάτος παραθύρου
const int WINDOW_HEIGHT = 700;     // Ύψος παραθύρου

// Παράμετροι για το fractal jellyfish
const float INITIAL_RADIUS = 80.0f;    // Αρχική ακτίνα του σώματος της μέδουσας
const float RADIUS_DECAY = 0.7f;       // Συντελεστής μείωσης ακτίνας για κάθε
επανάληψη
const float TENTACLE_LENGTH = 100.0f;  // Αρχικό μήκος πλοκαμιών
```

```
const float TENTACLE_DECAY = 0.7f;     // Συντελεστής μείωσης μήκους για κάθε τμήμα
του πλοκαμιού
const int NUM_TENTACLES = 6;           // Αριθμός πλοκαμιών
const int MAX_DEPTH = 6;               // Μέγιστο βάθος αναδρομής

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει ένα τμήμα από το πλοκάμι της μέδουσας.
 * @param x Η αρχική συντεταγμένη x.
 * @param y Η αρχική συντεταγμένη y.
 * @param length Το μήκος του πλοκαμιού.
 * @param angle Η γωνία περιστροφής του πλοκαμιού.
 * @param depth Το υπόλοιπο βάθος αναδρομής.
 */
void drawTentacle(float x, float y, float length, float angle, int depth) {
    if (depth == 0) return; // Βάση της αναδρομής

    // Υπολογισμός των συντεταγμένων του τέλους της γραμμής
    float endX = x + length * cos(angle * M_PI / 180.0f);
    float endY = y + length * sin(angle * M_PI / 180.0f);

    // Σχεδίαση της γραμμής με χρώμα που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.4f + 0.1f * depth;  // Ροζ/μωβ απόχρωση
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.8f - 0.1f * depth;
    graphics::drawLine(x, y, endX, endY, brush);

    // Αναδρομικές κλήσεις για τα επόμενα τμήματα του πλοκαμιού
    drawTentacle(endX, endY, length * TENTACLE_DECAY, angle + 15.0f, depth - 1);
    drawTentacle(endX, endY, length * TENTACLE_DECAY, angle - 15.0f, depth - 1);
}

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το σώμα της μέδουσας και τα πλοκάμια.
 * @param x Η αρχική συντεταγμένη x του κέντρου του σώματος.
 * @param y Η αρχική συντεταγμένη y του κέντρου του σώματος.
 * @param radius Η ακτίνα του σώματος.
 * @param depth Το υπόλοιπο βάθος αναδρομής.
 */
void drawJellyfish(float x, float y, float radius, int depth) {
    if (depth == 0) return; // Βάση της αναδρομής

    // Σχεδίαση του σώματος της μέδουσας ως δίσκος
    graphics::Brush brush;
    brush.fill_color[0] = 0.9f;  // Απαλό ροζ-μωβ χρώμα
    brush.fill_color[1] = 0.6f;
    brush.fill_color[2] = 0.9f;
    graphics::drawDisk(x, y, radius, brush);

    // Σχεδίαση των πλοκαμιών γύρω από το σώμα
    for (int i = 0; i < NUM_TENTACLES; ++i) {
        float angle = 360.0f / NUM_TENTACLES * i;  // Κατανομή πλοκαμιών γύρω από τον
κύκλο
        drawTentacle(x, y + radius, TENTACLE_LENGTH, angle, MAX_DEPTH);
    }

    // Αναδρομή για τη σχεδίαση του επόμενου μικρότερου σώματος της μέδουσας
    drawJellyfish(x, y, radius * RADIUS_DECAY, depth - 1);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ και καθορίζει το φόντο.
 */
void draw() {
    // Καθαρισμός παραθύρου με μαύρο φόντο
```

```
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;  // Μαύρο φόντο
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κεντρική κλήση για το fractal jellyfish
    drawJellyfish(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 3, INITIAL_RADIUS, MAX_DEPTH);
}

/**
 * @brief Κύρια συνάρτηση που εκκινεί το πρόγραμμα.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Jellyfish Fractal in SGG");

    graphics::setDrawFunction(draw);

    graphics::startMessageLoop();

    return 0;
}
```

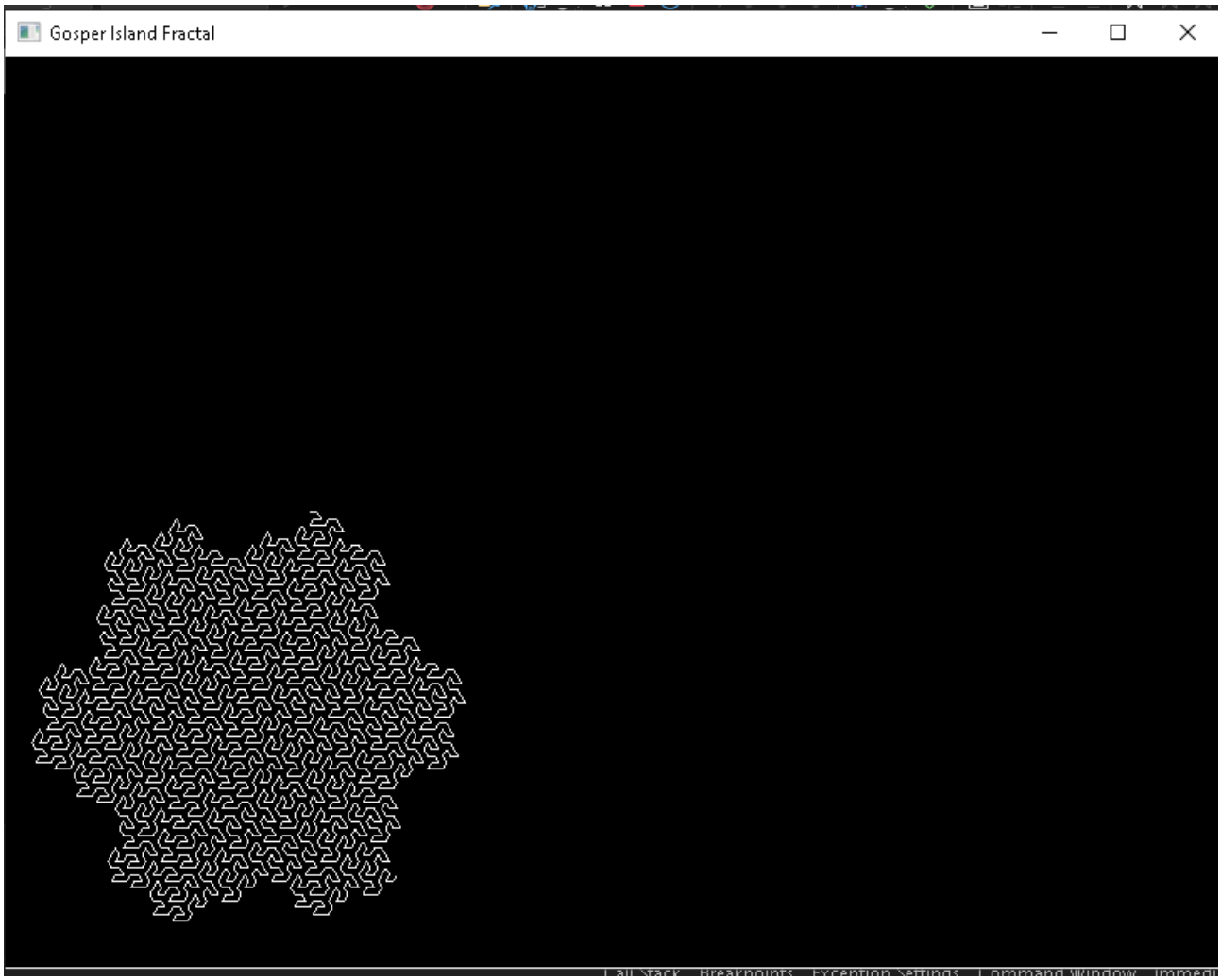
Επεξήγηση του Κώδικα

1. **drawTentacle**: Η συνάρτηση αυτή σχεδιάζει κάθε "πλοκάμι" της μέδουσας. Ξεκινά από την αρχή του πλοκαμιού και, μέσω αναδρομής, δημιουργεί διαδοχικές μικρότερες γραμμές που καμπυλώνουν το πλοκάμι.
2. **drawJellyfish**: Η κύρια συνάρτηση που σχεδιάζει το σώμα της μέδουσας και καλεί τη συνάρτηση `drawTentacle` για να προσθέσει πλοκάμια γύρω από το σώμα. Η αναδρομή δημιουργεί διαδοχικά μικρότερους δίσκους για το σώμα της μέδουσας, δίνοντας το φαινόμενο πολλαπλών στρωμάτων.
3. **draw**: Σχεδιάζει το jellyfish fractal στο κέντρο του παραθύρου με μαύρο φόντο.

Σημειώσεις

- Οι παράμετροι `RADIUS_DECAY` και `TENTACLE_DECAY` μπορούν να τροποποιηθούν για να δημιουργήσουν διάφορες μορφές και "πυκνότητες" πλοκαμιών.
- Το `MAX_DEPTH` καθορίζει το επίπεδο λεπτομέρειας, με υψηλότερες τιμές να προσθέτουν περισσότερα επίπεδα.

Gosper Island Fractal



Για να δημιουργήσουμε το **Gosper Island fractal** (ή αλλιώς Gosper Curve ή Dragon Curve), θα χρησιμοποιήσουμε μια γραμμή Lindenmayer (L-system) με τους κανόνες παραγωγής που ορίζουν τη γεωμετρική του ανάπτυξη. Το fractal αυτό είναι ενδιαφέρον γιατί αναπτύσσεται με έναν αυτο-ομοιόμορφο τρόπο, δημιουργώντας πολύπλοκα μοτίβα με βάση απλές οδηγίες.

Ακολουθούν τα βήματα και το πλήρες πρόγραμμα σε C++ για τη σχεδίαση του Gosper Island fractal, χρησιμοποιώντας τη βιβλιοθήκη **SGG**.

Περιγραφή του Gosper Island Fractal

Η Gosper Curve είναι ένα fractal που σχεδιάζεται με τη χρήση ενός L-system με τους εξής κανόνες:

- **Αρχικός κανόνας (Axiom)**: `A`
- **Κανόνες παραγωγής**:
  - `A -> A-B--B+A++AA+B-`
  - `B -> +A-BB--B-A++A+B`
- **Γωνία**: 60 μοίρες

Αυτές οι οδηγίες καθοδηγούν τη χελώνα να σχεδιάσει γραμμές, με το fractal να μεγαλώνει σε κάθε επανάληψη σύμφωνα με τους κανόνες.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <string>
#include <cmath>

// Σταθερά για τον υπολογισμό του π
#define M_PI 3.14159265358979323846

// Καθορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

// Γωνία περιστροφής σε ακτίνια (60 μοίρες)
const float ANGLE = 60.0f * M_PI / 180.0f;

/**
 * @brief Δομή για την "χελώνα", η οποία διατηρεί την τρέχουσα θέση και προσανατολισμό για τη σχεδίαση.
 */
struct Turtle {
    float x, y;          ///< Θέση x, y της χελώνας
    float angle;         ///< Γωνία προσανατολισμού
    bool penDown;        ///< Κατάσταση στυλό (ενεργό ή όχι)
    graphics::Brush brush;

    /**
     * @brief Κατασκευαστής για την αρχικοποίηση της χελώνας με θέση και χρώμα.
     * @param startX Αρχική θέση x.
     * @param startY Αρχική θέση y.
     */
    Turtle(float startX, float startY) : x(startX), y(startY), angle(0), penDown(true) {
        brush.fill_color[0] = 0.0f; // Μαύρο χρώμα για τη σχεδίαση
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;
    }

    /**
     * @brief Μετακίνηση της χελώνας προς τα εμπρός, αφήνοντας γραμμή αν το στυλό είναι κάτω.
     * @param distance Απόσταση μετακίνησης.
     */
    void forward(float distance) {
        float newX = x + distance * cos(angle);
        float newY = y + distance * sin(angle);
        if (penDown) {
            graphics::drawLine(x, y, newX, newY, brush);
        }
        x = newX;
        y = newY;
    }

    /**
     * @brief Στροφή της χελώνας προς τα αριστερά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnLeft(float radians) {
        angle -= radians;
```

```
    }

    /**
     * @brief Στροφή της χελώνας προς τα δεξιά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnRight(float radians) {
        angle += radians;
    }
};

/**
 * @brief Συνάρτηση για τη δημιουργία του L-System για την καμπύλη Gosper.
 * @param iterations Αριθμός επαναλήψεων για την επέκταση της ακολουθίας.
 * @return Η ακολουθία εντολών που δημιουργήθηκε για την καμπύλη Gosper.
 */
std::string generateGosperCurve(int iterations) {
    std::string result = "A"; // Αρχικό αξίωμα της ακολουθίας
    for (int i = 0; i < iterations; ++i) {
        std::string next;
        for (char c : result) {
            if (c == 'A') {
                next += "A-B--B+A++AA+B-"; // Κανόνας για το "A"
            }
            else if (c == 'B') {
                next += "+A-BB--B-A++A+B"; // Κανόνας για το "B"
            }
            else {
                next += c; // Διατήρηση του χαρακτήρα αν δεν είναι 'A' ή 'B'
            }
        }
        result = next;
    }
    return result;
}

/**
 * @brief Σχεδίαση της καμπύλης Gosper με χρήση της δομής χελώνας και εντολών L-
System.
 * @param turtle Αντικείμενο χελώνας για τη σχεδίαση.
 * @param instructions Η ακολουθία εντολών που θα ακολουθήσει η χελώνα.
 * @param length Μήκος κάθε βήματος της χελώνας.
 */
void drawGosperCurve(Turtle& turtle, const std::string& instructions, float length) {
    for (char command : instructions) {
        if (command == 'A' || command == 'B') {
            turtle.forward(length); // Μετακίνηση προς τα εμπρός
        }
        else if (command == '+') {
            turtle.turnLeft(ANGLE); // Στροφή προς τα αριστερά
        }
        else if (command == '-') {
            turtle.turnRight(ANGLE); // Στροφή προς τα δεξιά
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που εκτελείται σε κάθε καρέ.
 */
void draw() {
    // Καθαρισμός παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
```

```
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Αρχικοποίηση της χελώνας και των οδηγιών για την καμπύλη Gosper
    Turtle turtle(WINDOW_WIDTH / 4, WINDOW_HEIGHT / 2);  // Αρχική θέση της χελώνας
    std::string instructions = generateGosperCurve(4);  // Επίπεδο επανάληψης της
καμπύλης
    drawGosperCurve(turtle, instructions, 5.0f); // Μήκος κάθε βήματος
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος για τη δημιουργία και εκτέλεση της
εφαρμογής.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Gosper Island Fractal");

    graphics::setDrawFunction(draw);

    graphics::startMessageLoop();

    return 0;
}
```

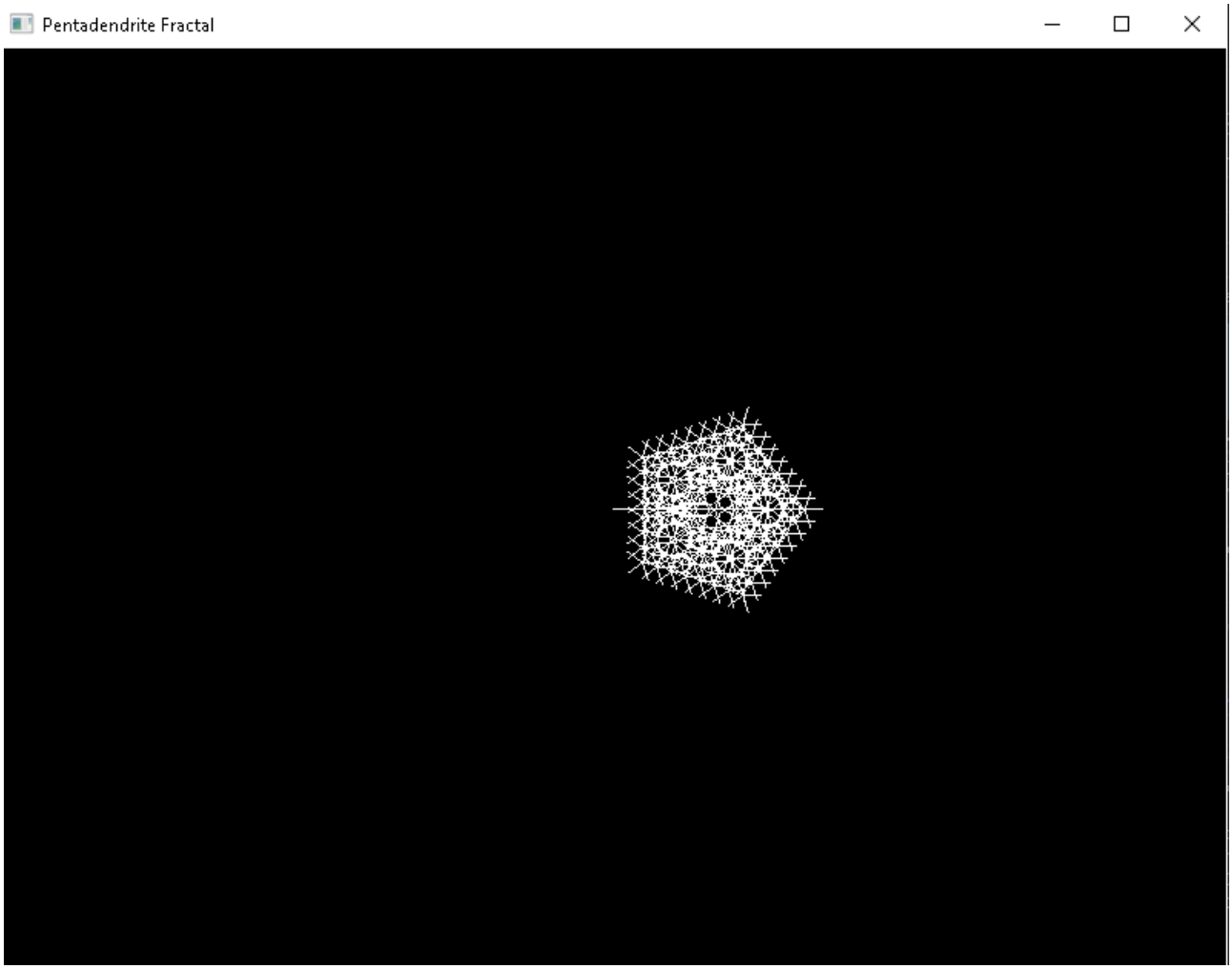
Επεξήγηση Κώδικα

1. **Δημιουργία Παραθύρου**: Ορίζει το μέγεθος του παραθύρου σε 800x800 pixels.
2. **Χελώνα**: Χρησιμοποιούμε την Turtle για να σχεδιάζουμε γραμμές, περιστροφές και κατευθύνσεις.
3. **L-System**: Η συνάρτηση `generateGosperCurve` δημιουργεί τις οδηγίες για το fractal ανάλογα με τον αριθμό των επαναλήψεων.
4. **Σχεδίαση Fractal**: Η `drawGosperCurve` χρησιμοποιεί τις οδηγίες της καμπύλης για να κινήσει τη χελώνα και να δημιουργήσει το σχέδιο.
5. **Κλήση Σχεδίασης και Εκτέλεσης**: Η `draw` σχεδιάζει το fractal καλώντας τις συναρτήσεις του L-system και τη χελώνα, και η `main` εκκινεί το πρόγραμμα.

Σχόλια

- **Επαναληψιμότητα και Απόδοση**: Για μεγαλύτερο βάθος (π.χ., `iterations=5`), το fractal γίνεται πιο λεπτομερές αλλά αυξάνεται και η πολυπλοκότητα, απαιτώντας περισσότερο χρόνο σχεδίασης.
- **Αλλαγές στο Μήκος και τη Γωνία**: Μπορείτε να πειραματιστείτε με το `ANGLE` και το μήκος για να προσαρμόσετε το fractal σε διαφορετικά μεγέθη ή μοτίβα.

Pentadendrite Fractal



Το συγκεκριμένο fractal χαρακτηρίζεται από πέντε συμμετρικά υποκαταστήματα που επαναλαμβάνονται σε κάθε επίπεδο αναδρομής, θυμίζοντας ένα είδος αστεριού ή δενδρικής δομής.

Περιγραφή του Pentadendrite Fractal

Το Pentadendrite είναι ένα γεωμετρικό fractal που χρησιμοποιεί έναν κανόνα αναδρομής για να προσθέτει πέντε μικρότερα αντίγραφα ενός κλάδου γύρω από ένα κεντρικό σημείο. Κάθε κλάδος έχει την ίδια μορφή με το αρχικό μοτίβο και αναπτύσσεται με προκαθορισμένη γωνία, προσθέτοντας ένα εντυπωσιακό αποτέλεσμα συμμετρίας.

Κώδικας για τη Σχεδίαση του Pentadendrite Fractal

Ακολουθεί το πλήρες πρόγραμμα σε C++ που χρησιμοποιεί τη βιβλιοθήκη SGG για τη σχεδίαση του Pentadendrite fractal:

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <cmath>
```

```
// Σταθερά για τον υπολογισμό του π
#define M_PI 3.14159265358979323846

// Ρυθμίσεις διαστάσεων παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

// Γωνία περιστροφής για κάθε νέο κλάδο (72 μοίρες)
const float ANGLE = 72.0f * M_PI / 180.0f;

/**
 * @brief Δομή για την "χελώνα", που διατηρεί την τρέχουσα θέση και γωνία της για τη
σχεδίαση.
 */
struct Turtle {
    float x, y;          ///< Συντεταγμένες της θέσης της χελώνας
    float angle;         ///< Γωνία προσανατολισμού
    graphics::Brush brush;

    /**
     * @brief Κατασκευαστής για την αρχικοποίηση της χελώνας με θέση και χρώμα.
     * @param startX Αρχική θέση x.
     * @param startY Αρχική θέση y.
     */
    Turtle(float startX, float startY) : x(startX), y(startY), angle(0) {
        brush.fill_color[0] = 0.0f; // Μαύρο χρώμα για τη σχεδίαση
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;
    }

    /**
     * @brief Προχωράει τη χελώνα προς τα εμπρός αφήνοντας μια γραμμή.
     * @param distance Απόσταση μετακίνησης.
     */
    void forward(float distance) {
        float newX = x + distance * cos(angle);
        float newY = y + distance * sin(angle);
        graphics::drawLine(x, y, newX, newY, brush);
        x = newX;
        y = newY;
    }

    /**
     * @brief Περιστροφή της χελώνας προς τα αριστερά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnLeft(float radians) {
        angle -= radians;
    }

    /**
     * @brief Περιστροφή της χελώνας προς τα δεξιά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnRight(float radians) {
        angle += radians;
    }
};

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του fractal Pentadendrite.
 * @param turtle Η χελώνα που καθοδηγεί τη σχεδίαση.
 * @param length Μήκος τρέχοντος κλάδου.
 * @param depth Βάθος αναδρομής.
 */
```

```
void drawPentadendrite(Turtle& turtle, float length, int depth) {
    if (depth == 0) {
        // Τελικό σημείο αναδρομής: απλή μετακίνηση προς τα εμπρός
        turtle.forward(length);
        return;
    }

    // Σχεδίαση του αρχικού τμήματος του κεντρικού κλάδου
    turtle.forward(length / 3);

    // Σχεδίαση πέντε υποκαταστημάτων γύρω από το τμήμα του κεντρικού κλαδιού
    for (int i = 0; i < 5; ++i) {
        // Αποθήκευση της τρέχουσας θέσης και γωνίας για επαναφορά μετά τον κλάδο
        float oldX = turtle.x;
        float oldY = turtle.y;
        float oldAngle = turtle.angle;

        // Περιστροφή της χελώνας για τη σχεδίαση του επόμενου κλάδου
        turtle.turnLeft(ANGLE * i);

        // Αναδρομή για σχεδίαση του υποκαταστήματος
        drawPentadendrite(turtle, length / 2, depth - 1);

        // Επαναφορά της θέσης και γωνίας της χελώνας
        turtle.x = oldX;
        turtle.y = oldY;
        turtle.angle = oldAngle;
    }

    // Σχεδίαση του τελικού τμήματος του κεντρικού κλαδιού
    turtle.forward(length / 3);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από την SGG Library σε κάθε καρέ.
 */
void draw() {
    // Καθαρισμός παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Αρχικοποίηση της χελώνας στο κέντρο του παραθύρου
    Turtle turtle(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2);

    // Σχεδίαση του Pentadendrite fractal με αρχικό μήκος κλαδιού και βάθος
    drawPentadendrite(turtle, 200.0f, 4);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Pentadendrite Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

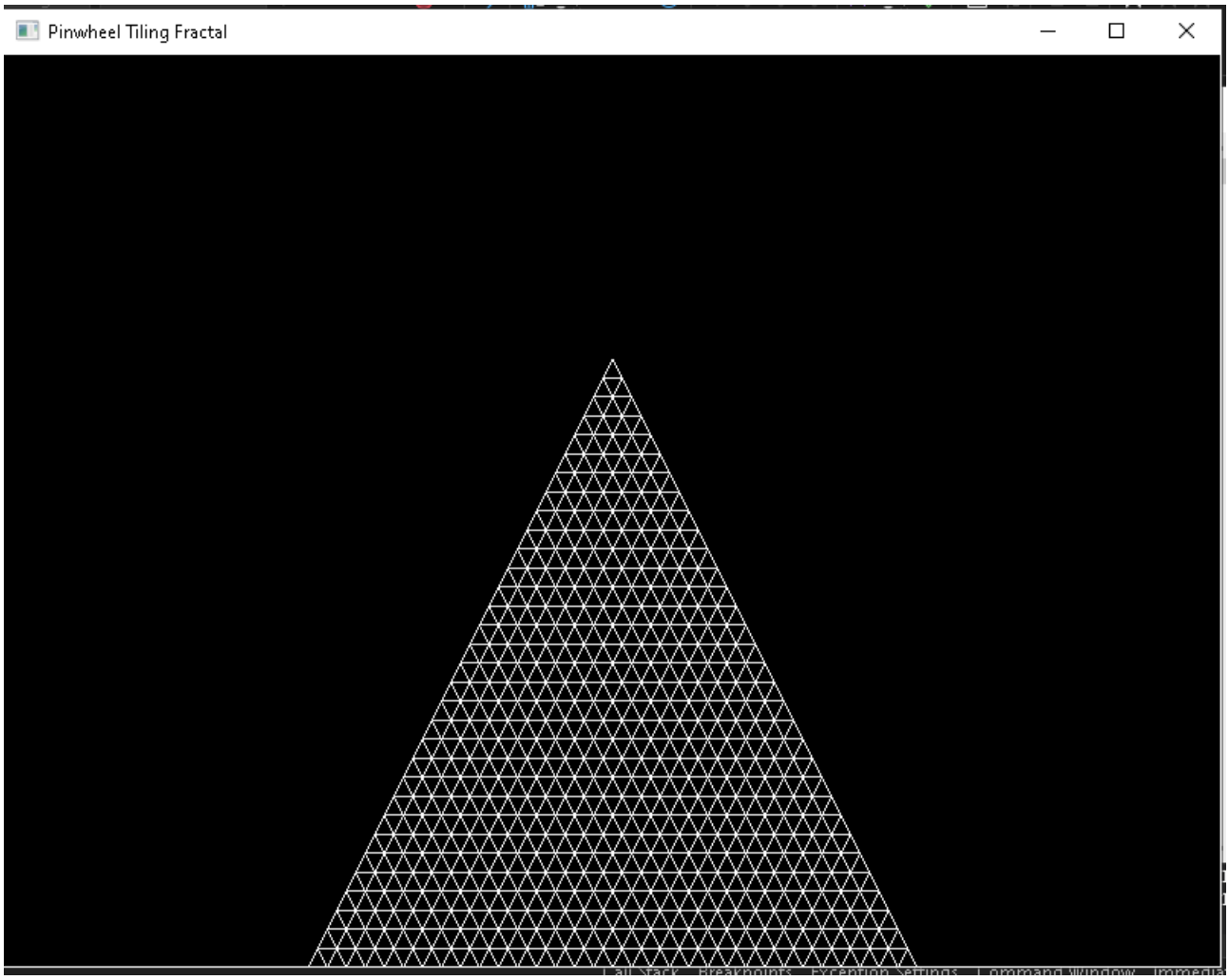
Επεξήγηση Κώδικα

1. **Δημιουργία Παραθύρου**: Ορίζει το μέγεθος του παραθύρου σε 800x800 pixels.
2. **Χελώνα**: Η δομή Turtle είναι υπεύθυνη για την κίνηση της χελώνας στη σχεδίαση γραμμών και για τις περιστροφές.
3. **Συνάρτηση Αναδρομής** `drawPentadendrite`: Αυτή η συνάρτηση καλείται αναδρομικά για κάθε επίπεδο του fractal:
    - Κάθε κλάδος χωρίζεται σε μικρότερα τμήματα.
    - Σε κάθε βήμα, προστίθενται πέντε υποκαταστήματα που περιστρέφονται σε προκαθορισμένη γωνία (72 μοίρες) γύρω από το αρχικό σημείο.
4. **Σχεδίαση Fractal**: Η `draw` ορίζει το φόντο και καλεί τη συνάρτηση για τη σχεδίαση του fractal, η οποία αρχίζει από το κέντρο της οθόνης.
5. **Κύρια Συνάρτηση**: Η `main` δημιουργεί το παράθυρο, θέτει τη συνάρτηση σχεδίασης και ξεκινά τη λούπα μηνυμάτων.

Σχόλια

- **Προσαρμογή Βάθους**: Μπορείτε να αυξήσετε το βάθος της αναδρομής (π.χ., `depth = 5`) για λεπτομερέστερο fractal, όμως αυτό αυξάνει την πολυπλοκότητα.
- **Αλλαγή Μήκους**: Πειραματιστείτε με το μήκος κάθε κλάδου για διαφορετικά μεγέθη fractals.

Pinwheel Tiling Fractal



Το **Pinwheel Tiling fractal** είναι ένα fractal που σχηματίζει ένα συμμετρικό μοτίβο χρησιμοποιώντας τρίγωνα με σταδιακή περιστροφή και τοποθέτηση σε επαναλαμβανόμενα μοτίβα. Η σχεδίασή του απαιτεί τον κατακερματισμό ενός τριγώνου σε μικρότερα τρίγωνα, τα οποία τοποθετούνται με περιστροφή για να δημιουργήσουν έναν πλακώδη σχηματισμό.

Ακολουθεί το πρόγραμμα για τη σχεδίαση του **Pinwheel Tiling fractal** σε C++ χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <cmath>

// Ρυθμίσεις παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;
const float PI = 3.14159265358979323846;

/**
 * @brief Δομή που αντιπροσωπεύει ένα τρίγωνο.
 */
struct Triangle {
    float x1, y1;  ///< Συντεταγμένες της πρώτης κορυφής
```

```
    float x2, y2;  ///< Συντεταγμένες της δεύτερης κορυφής
    float x3, y3;  ///< Συντεταγμένες της τρίτης κορυφής
};

/**
 * @brief Σχεδιάζει ένα τρίγωνο χρησιμοποιώντας την βιβλιοθήκη γραφικών SGG.
 * @param triangle Το τρίγωνο που θα σχεδιαστεί.
 * @param brush Το πινέλο για τη ρύθμιση του χρώματος της γραμμής.
 */
void drawTriangle(const Triangle& triangle, const graphics::Brush& brush) {
    // Σχεδίαση των πλευρών του τριγώνου ως γραμμές
    graphics::drawLine(triangle.x1, triangle.y1, triangle.x2, triangle.y2, brush);
    graphics::drawLine(triangle.x2, triangle.y2, triangle.x3, triangle.y3, brush);
    graphics::drawLine(triangle.x3, triangle.y3, triangle.x1, triangle.y1, brush);
}

/**
 * @brief Αναδρομική συνάρτηση για τη διαίρεση ενός τριγώνου σε μικρότερα τρίγωνα,
δημιουργώντας το fractal "Pinwheel".
 * @param t Το τρίγωνο που θα διαιρεθεί.
 * @param depth Το τρέχον βάθος αναδρομής. Όσο μεγαλύτερο το βάθος, τόσο περισσότερη
λεπτομέρεια.
 * @param brush Το πινέλο που καθορίζει το χρώμα των γραμμών.
 */
void pinwheelDivide(Triangle t, int depth, const graphics::Brush& brush) {
    if (depth == 0) {
        // Βάση της αναδρομής: σχεδιάζουμε το τρίγωνο
        drawTriangle(t, brush);
        return;
    }

    // Υπολογισμός μέσων σημείων για κάθε πλευρά του τριγώνου
    float mx1 = (t.x1 + t.x2) / 2;
    float my1 = (t.y1 + t.y2) / 2;
    float mx2 = (t.x2 + t.x3) / 2;
    float my2 = (t.y2 + t.y3) / 2;
    float mx3 = (t.x3 + t.x1) / 2;
    float my3 = (t.y3 + t.y1) / 2;

    // Δημιουργία νέων μικρότερων τριγώνων
    Triangle t1 = { t.x1, t.y1, mx1, my1, mx3, my3 };
    Triangle t2 = { mx1, my1, t.x2, t.y2, mx2, my2 };
    Triangle t3 = { mx3, my3, mx2, my2, t.x3, t.y3 };
    Triangle t4 = { mx1, my1, mx2, my2, mx3, my3 };

    // Αναδρομή για κάθε νέο τρίγωνο με μειωμένο βάθος
    pinwheelDivide(t1, depth - 1, brush);
    pinwheelDivide(t2, depth - 1, brush);
    pinwheelDivide(t3, depth - 1, brush);
    pinwheelDivide(t4, depth - 1, brush);
}

/**
 * @brief Σχεδιάζει το βασικό τριγωνικό μοτίβο και ξεκινά την αναδρομική διαίρεση για
το Pinwheel fractal.
 */
void drawPinwheelFractal() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f;  // Κόκκινη απόχρωση
    brush.fill_color[1] = 0.5f;  // Πράσινη απόχρωση
    brush.fill_color[2] = 0.7f;  // Μπλε απόχρωση

    // Ορισμός του αρχικού τριγώνου για το fractal
    Triangle baseTriangle = { 400, 200, 600, 600, 200, 600 };
```

```
    // Εκκίνηση αναδρομικής διαίρεσης
    pinwheelDivide(baseTriangle, 5, brush);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για να σχεδιάσει το
fractal στο παράθυρο.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Σχεδίαση του fractal "Pinwheel"
    drawPinwheelFractal();
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά το βρόχο
σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Pinwheel Tiling Fractal");

    graphics::setDrawFunction(draw);
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση του Κώδικα

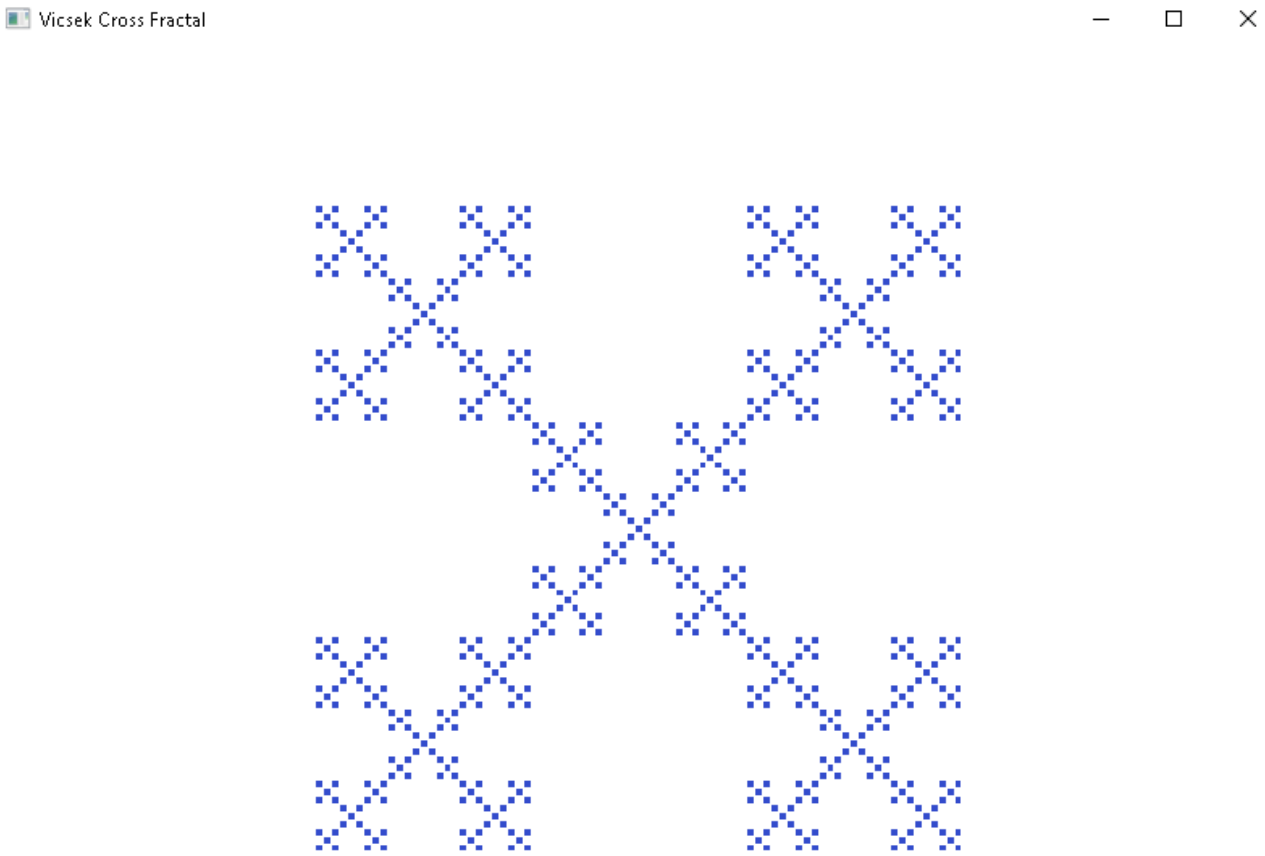
1. **Δομή** `Triangle`: Καθορίζει ένα τρίγωνο με τρία σημεία.
2. **Συνάρτηση** `drawTriangle`: Σχεδιάζει ένα τρίγωνο χρησιμοποιώντας τις κορυφές που του έχουν οριστεί.
3. **Συνάρτηση** `pinwheelDivide`: Είναι η βασική συνάρτηση για το Pinwheel fractal. Διαιρεί το αρχικό τρίγωνο σε τέσσερα μικρότερα τρίγωνα. Αυτή η συνάρτηση καλείται αναδρομικά για κάθε τρίγωνο μέχρι το καθορισμένο βάθος.
4. `drawPinwheelFractal`: Ξεκινάει με ένα αρχικό τρίγωνο στη μέση του παραθύρου και καλεί την αναδρομική συνάρτηση διαχωρισμού.
5. `draw`: Ορίζει το φόντο και καλεί τη συνάρτηση σχεδίασης του fractal.

Σχόλια

- **Προσαρμογή Βάθους**: Το βάθος της αναδρομής καθορίζει τη λεπτομέρεια του fractal. Μεγαλύτερο βάθος οδηγεί σε περισσότερα τρίγωνα.
- **Αλλαγή Χρωμάτων**: Μπορείτε να προσαρμόσετε τα χρώματα του `brush` για διαφορετικά οπτικά αποτελέσματα.

Αυτό το πρόγραμμα σχεδιάζει το Pinwheel fractal χωρίζοντας το αρχικό τρίγωνο σε μικρότερα και επαναλαμβάνοντας τη διαδικασία για κάθε υποτρίγωνο.

Vicsek Cross Fractal



Το **Vicsek Cross Fractal** είναι ένα fractal που δημιουργείται με την υποδιαίρεση ενός τετραγώνου σε πέντε μικρότερα τετράγωνα, διατηρώντας ένα μοτίβο διασταυρούμενων τετραγώνων. Αυτό το fractal μπορεί να σχεδιαστεί με τη χρήση αναδρομής.

Ακολουθεί ο κώδικας για τη σχεδίαση του **Vicsek Cross Fractal** σε C++ χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>

// Ρυθμίσεις παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Σχεδιάζει ένα τετράγωνο με κέντρο τις συντεταγμένες (x, y) και συγκεκριμένη πλευρά.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param side Το μήκος της πλευράς του τετραγώνου.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα του τετραγώνου.
 */
```

```
void drawSquare(float x, float y, float side, const graphics::Brush& brush) {
    graphics::drawRect(x, y, side, side, brush);
}

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Vicsek Cross Fractal.
 *        Σε κάθε επίπεδο αναδρομής, το τετράγωνο χωρίζεται σε πέντε μικρότερα
 *        τετράγωνα στη μορφή σταυρού.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param side Το μήκος της πλευράς του τετραγώνου.
 * @param depth Το τρέχον βάθος της αναδρομής. Αν το depth είναι 0, σταματά η
αναδρομή.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα των τετραγώνων.
 */
void drawVicsekCross(float x, float y, float side, int depth, const graphics::Brush&
brush) {
    // Βάση της αναδρομής: αν το βάθος είναι 0, σχεδιάζουμε το τετράγωνο και
επιστρέφουμε
    if (depth == 0) {
        drawSquare(x, y, side, brush);
        return;
    }

    // Υπολογισμός νέου μήκους πλευράς για τα μικρότερα τετράγωνα
    float newSide = side / 3.0f;

    // Αναδρομική κλήση για τα πέντε τετράγωνα στη μορφή του σταυρού
    drawVicsekCross(x, y, newSide, depth - 1, brush);                           //
Κεντρικό τετράγωνο
    drawVicsekCross(x - newSide, y - newSide, newSide, depth - 1, brush);     // Άνω
αριστερό τετράγωνο
    drawVicsekCross(x + newSide, y - newSide, newSide, depth - 1, brush);     // Άνω
δεξί τετράγωνο
    drawVicsekCross(x - newSide, y + newSide, newSide, depth - 1, brush);     // Κάτω
αριστερό τετράγωνο
    drawVicsekCross(x + newSide, y + newSide, newSide, depth - 1, brush);     // Κάτω
δεξί τετράγωνο
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για τη δημιουργία του
παραθύρου.
 *        Καθαρίζει την οθόνη και καλεί τη συνάρτηση για τη σχεδίαση του Vicsek Cross
Fractal.
 */
void draw() {
    // Καθαρισμός του παραθύρου με άσπρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 1.0f;  // Άσπρο φόντο
    bg.fill_color[1] = 1.0f;
    bg.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ρύθμιση πινέλου για το fractal
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;  // Μπλε απόχρωση
    brush.fill_color[1] = 0.3f;
    brush.fill_color[2] = 0.8f;

    // Κεντρική κλήση της συνάρτησης αναδρομής για το Vicsek Cross Fractal
    drawVicsekCross(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 400, 4, brush);
}
```

```
/**
 * @brief Κύρια συνάρτηση του προγράμματος που δημιουργεί το παράθυρο, ορίζει τη
συνάρτηση σχεδίασης και ξεκινά το πρόγραμμα.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Vicsek Cross Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για την ανανέωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση του Κώδικα

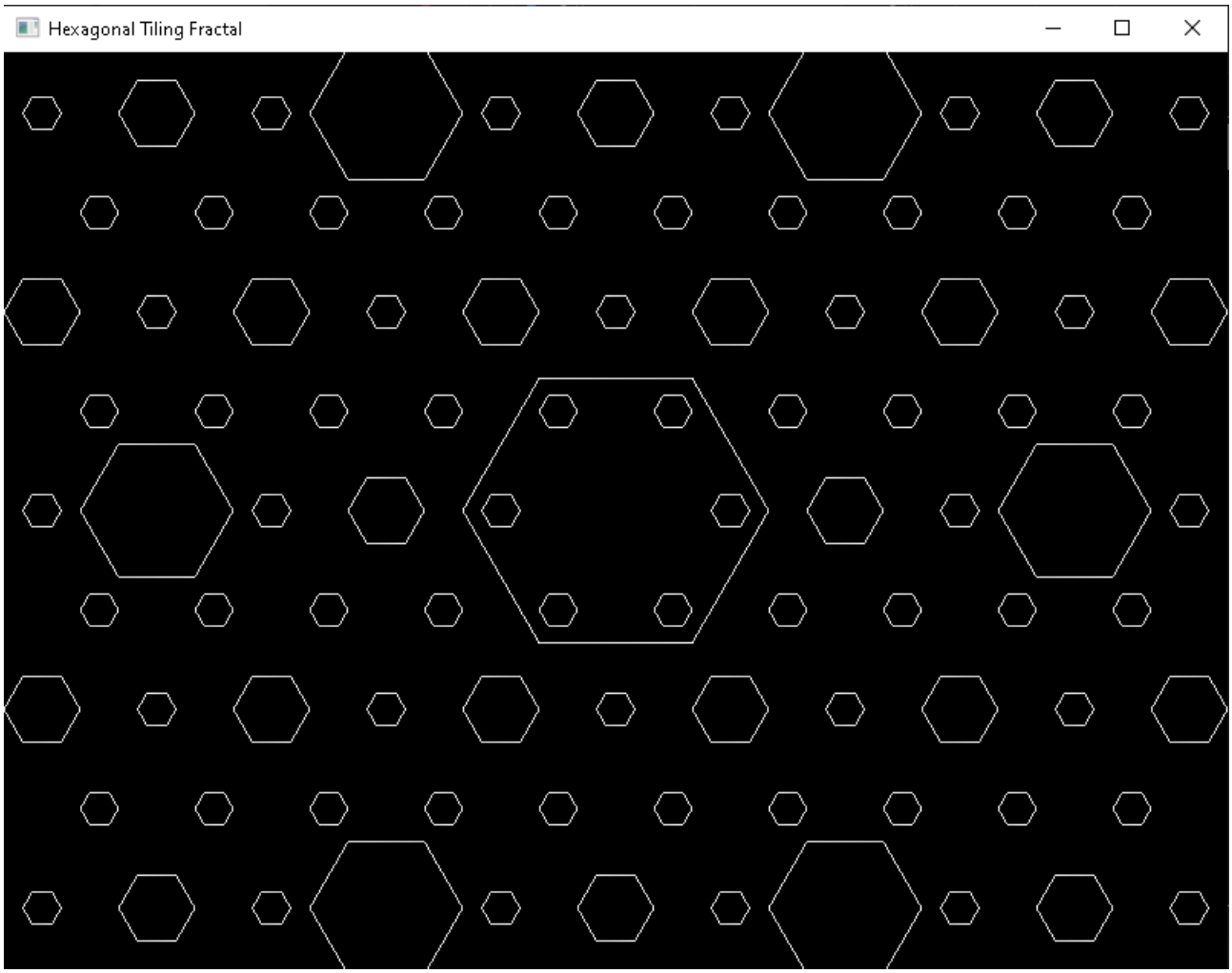
1. **Συνάρτηση** `drawSquare`: Σχεδιάζει ένα τετράγωνο σε μια δεδομένη θέση `(x, y)` με πλευρά `side`.
2. **Συνάρτηση** `drawVicsekCross`: Η βασική συνάρτηση για το fractal. Χωρίζει το αρχικό τετράγωνο σε πέντε μικρότερα τετράγωνα σε μορφή σταυρού και ανακαλείται αναδρομικά για κάθε μικρότερο τετράγωνο.
3. **Συνάρτηση** `draw`: Ορίζει το λευκό φόντο και καλεί την κεντρική συνάρτηση του fractal για να σχεδιάσει το μοτίβο στο κέντρο του παραθύρου.
4. `main`: Ορίζει το παράθυρο και καλεί τη συνάρτηση σχεδίασης με τη βιβλιοθήκη SGG.

Σχόλια

- **Βάθος Αναδρομής**: Το βάθος ελέγχει τον αριθμό των επαναλήψεων και τη λεπτομέρεια του fractal. Μεγαλύτερο βάθος δημιουργεί περισσότερο περίπλοκα μοτίβα.
- **Αλλαγή Χρωμάτων**: Το χρώμα του fractal μπορεί να αλλάξει ρυθμίζοντας το `brush`.

Αυτό το πρόγραμμα σχεδιάζει το **Vicsek Cross Fractal** διαιρώντας το αρχικό τετράγωνο σε μικρότερα τετράγωνα σε κάθε βήμα, δημιουργώντας το χαρακτηριστικό μοτίβο σταυρού.

Hexagon Tiling Fractal



Το *Hexagonal Tiling Fractal* είναι ένα fractal που βασίζεται σε εξαγωνική συμμετρία. Η βασική δομή περιλαμβάνει την τοποθέτηση ενός εξαγώνου στη μέση, και στη συνέχεια την αναδρομική προσθήκη εξαγώνων γύρω από το αρχικό εξάγωνο, δημιουργώντας έναν μοτίβο που επεκτείνεται σε διάφορα επίπεδα. Θα φτιάξουμε το πρόγραμμα χρησιμοποιώντας αναδρομή και τη βιβλιοθήκη SGG για σχεδίαση.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

#define M_PI 3.14159265358979323846

// Διαστάσεις παραθύρου
const float WINDOW_WIDTH = 800.0f;
const float WINDOW_HEIGHT = 600.0f;

/**
 * @brief Δομή Hexagon που αναπαριστά ένα εξάγωνο και περιέχει συναρτήσεις για τον υπολογισμό των κορυφών του και τη σχεδίασή του.
 */
struct Hexagon {
    float x, y;  // Συντεταγμένες του κέντρου του εξαγώνου
```

```
    float size;  // Μέγεθος της πλευράς του εξαγώνου

    /**
     * @brief Σχεδιάζει το εξάγωνο συνδέοντας τις γειτονικές κορυφές.
     * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα και τις γραμμές του
εξαγώνου.
     */
    void drawHexagon(const graphics::Brush& brush) const {
        // Επαναληπτική διαδικασία για τις 6 πλευρές του εξαγώνου
        for (int i = 0; i < 6; ++i) {
            // Υπολογισμός γωνιών για τα δύο άκρα κάθε πλευράς
            float angle1 = M_PI / 3 * i;
            float angle2 = M_PI / 3 * (i + 1);
            float x1 = x + size * cos(angle1);
            float y1 = y + size * sin(angle1);
            float x2 = x + size * cos(angle2);
            float y2 = y + size * sin(angle2);

            // Σχεδίαση πλευράς του εξαγώνου
            graphics::drawLine(x1, y1, x2, y2, brush);
        }
    }
};

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Hexagonal Tiling Fractal.
 *        Δημιουργεί ένα εξάγωνο στο κέντρο και στη συνέχεια καλεί την ίδια τη
συνάρτηση για τα 6 γειτονικά εξάγωνα.
 * @param x Η συντεταγμένη x του κέντρου του εξαγώνου.
 * @param y Η συντεταγμένη y του κέντρου του εξαγώνου.
 * @param size Το μέγεθος της πλευράς του εξαγώνου.
 * @param depth Το βάθος αναδρομής. Αν το depth είναι 0, η συνάρτηση σταματά.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα και τις γραμμές του
εξαγώνου.
 */
void drawHexagonalTilingFractal(float x, float y, float size, int depth, const
graphics::Brush& brush) {
    // Βάση αναδρομής: αν το βάθος είναι 0, δεν σχεδιάζουμε περαιτέρω εξάγωνα
    if (depth == 0) return;

    // Δημιουργία του κεντρικού εξαγώνου
    Hexagon hex = { x, y, size };
    hex.drawHexagon(brush);

    // Παράγοντες για τον υπολογισμό των συντεταγμένων των 6 γειτονικών εξαγώνων
    const float dx[] = { 1.5f, 0.75f, -0.75f, -1.5f, -0.75f, 0.75f };
    const float dy[] = { 0.0f, 1.3f, 1.3f, 0.0f, -1.3f, -1.3f };

    // Αναδρομική κλήση για τα 6 γειτονικά εξάγωνα
    for (int i = 0; i < 6; ++i) {
        // Υπολογισμός νέων συντεταγμένων για τα γειτονικά εξάγωνα
        float newX = x + dx[i] * size * 2;
        float newY = y + dy[i] * size * 2;

        // Κλήση αναδρομής για το γειτονικό εξάγωνο
        drawHexagonalTilingFractal(newX, newY, size / 2, depth - 1, brush);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών SGG. Σχεδιάζει
το fractal των εξαγώνων.
 */
void draw() {
    // Ρύθμιση του πινέλου σχεδίασης
```

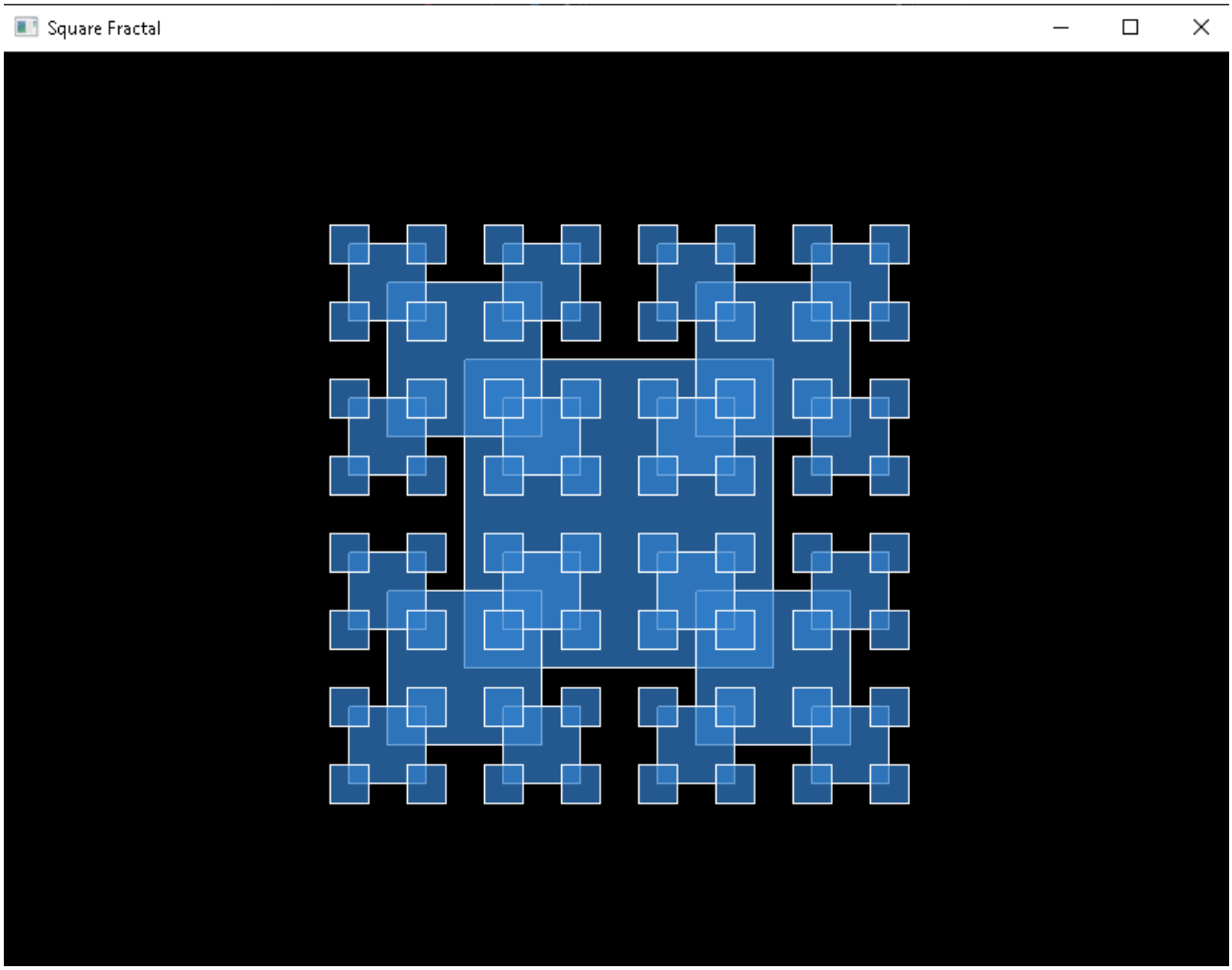
```
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;  // Απόχρωση μπλε-πράσινου για το fractal
    brush.fill_color[1] = 0.6f;
    brush.fill_color[2] = 0.8f;
    brush.outline_opacity = 1.0f;

    // Κλήση της αναδρομικής συνάρτησης για τη σχεδίαση του Hexagonal Tiling Fractal
    drawHexagonalTilingFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 4, brush);
}

/**
 * @brief Η κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον βρόχο ανανέωσης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου και καθορισμός του τίτλου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Hexagonal Tiling Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων για ανανέωση της οθόνης
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα:

1. **Δομή Hexagon**: Ορίζει ένα εξάγωνο με τις κεντρικές του συντεταγμένες και το μέγεθος πλευράς.
2. **Συνάρτηση drawHexagon**: Σχεδιάζει το εξάγωνο με χρήση της `graphics::drawLine` για να ενώσει τις κορυφές.
3. **drawHexagonalTilingFractal**: Αναδρομική συνάρτηση που σχεδιάζει εξάγωνα με μείωση του μεγέθους για κάθε επίπεδο και τοποθετώντας έξι γειτονικά εξάγωνα γύρω από το κεντρικό.
4. **draw**: Καλεί την αναδρομική συνάρτηση και αρχικοποιεί το πρώτο εξάγωνο στο κέντρο του παραθύρου.
5. **main**: Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά το κύκλο μηνυμάτων για τη διατήρηση του παραθύρου.

Αυτό το πρόγραμμα δημιουργεί ένα Hexagonal Tiling Fractal με επαναλαμβανόμενο μοτίβο εξαγώνων που μειώνονται σε μέγεθος σε κάθε επίπεδο.

Square Fractal



Το *Square Fractal* είναι ένα κλασικό fractal που ξεκινά με ένα μεγάλο τετράγωνο στο κέντρο και, σε κάθε βήμα αναδρομής, προσθέτει μικρότερα τετράγωνα στις τέσσερις γωνίες του. Καθώς η αναδρομή προχωρά, το fractal γίνεται πιο σύνθετο και γεμίζει σταδιακά τον χώρο.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Διαστάσεις παραθύρου
const float WINDOW_WIDTH = 800.0f;
const float WINDOW_HEIGHT = 600.0f;

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός fractal με τετράγωνα.
 *        Σχεδιάζει ένα κεντρικό τετράγωνο και στη συνέχεια δημιουργεί μικρότερα τετράγωνα στις γωνίες του.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param size Το μέγεθος της πλευράς του τετραγώνου.
 * @param depth Το βάθος αναδρομής. Η συνάρτηση σταματά αν το βάθος φτάσει στο 0.
 * @param brush Το πινέλο που χρησιμοποιείται για τον χρωματισμό των τετραγώνων.
 */
```

```
void drawSquareFractal(float x, float y, float size, int depth, const graphics::Brush&
brush) {
    // Βάση αναδρομής: όταν το βάθος είναι 0, σταματάμε τη σχεδίαση
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του νέου μεγέθους για τα επόμενα τετράγωνα
    float newSize = size / 2;

    // Αναδρομική σχεδίαση τετραγώνων στις τέσσερις γωνίες του τρέχοντος τετραγώνου
    drawSquareFractal(x - newSize, y - newSize, newSize, depth - 1, brush); // Πάνω
αριστερά γωνία
    drawSquareFractal(x + newSize, y - newSize, newSize, depth - 1, brush); // Πάνω
δεξιά γωνία
    drawSquareFractal(x - newSize, y + newSize, newSize, depth - 1, brush); // Κάτω
αριστερά γωνία
    drawSquareFractal(x + newSize, y + newSize, newSize, depth - 1, brush); // Κάτω
δεξιά γωνία
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ της
οθόνης.
 *        Καθορίζει το πινέλο και ξεκινά τη σχεδίαση του fractal από το κέντρο της
οθόνης.
 */
void draw() {
    // Ρύθμιση του πινέλου σχεδίασης
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;  // Απόχρωση μπλε-πράσινου για το fractal
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.8f;
    brush.fill_opacity = 0.7f;   // Ημιδιαφανές χρώμα

    // Κλήση της συνάρτησης drawSquareFractal από το κέντρο του παραθύρου
    drawSquareFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200, 4, brush);
}

/**
 * @brief Η κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον
βρόχο ανανέωσης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου και καθορισμός του τίτλου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Square Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων για ανανέωση της οθόνης
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα:

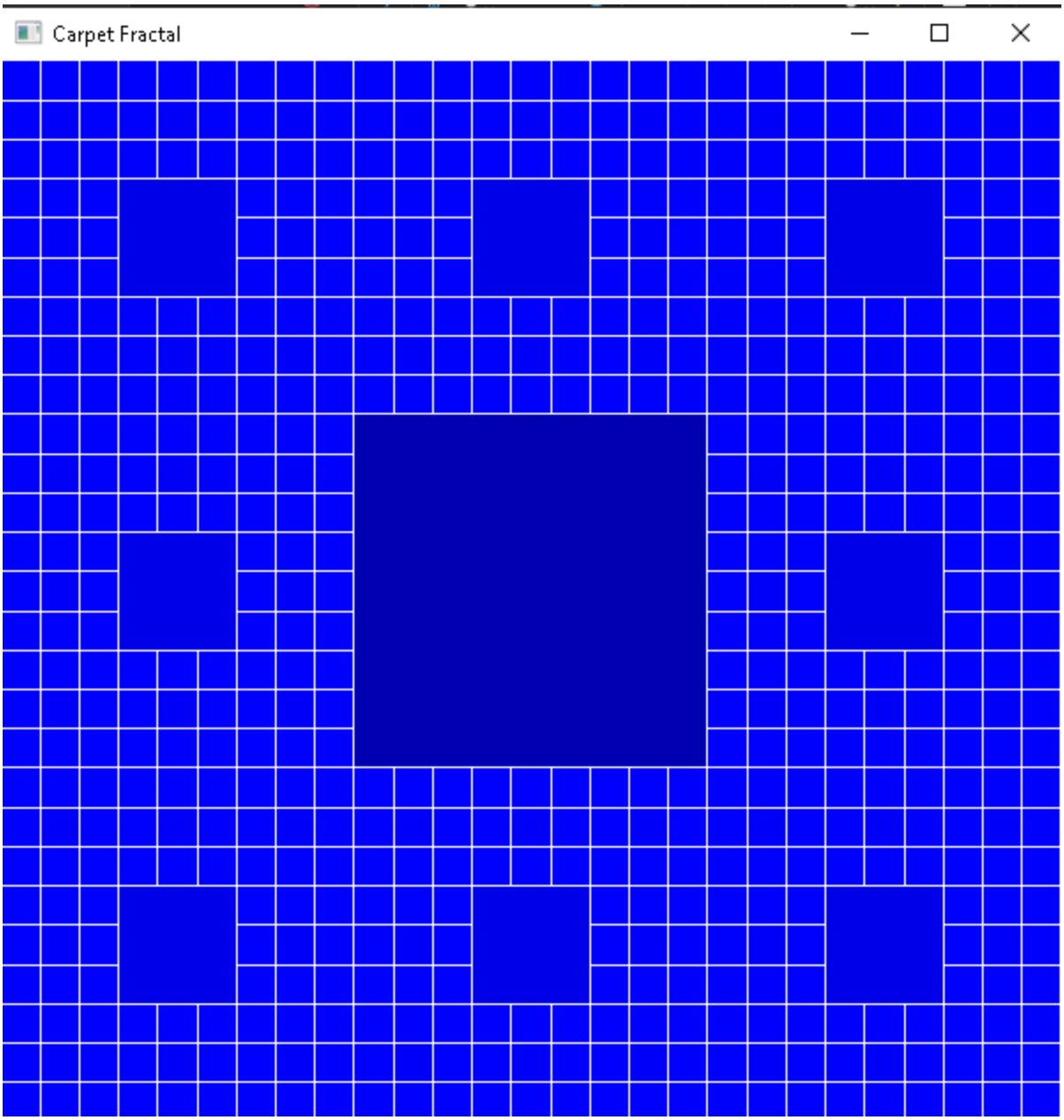
1. **drawSquareFractal**:

- Αυτή η αναδρομική συνάρτηση σχεδιάζει ένα κεντρικό τετράγωνο και στη συνέχεια σχεδιάζει τέσσερα μικρότερα τετράγωνα στις γωνίες του.
- Το `depth` καθορίζει τα επίπεδα αναδρομής, και κάθε μείωση του `depth` δημιουργεί μικρότερα τετράγωνα έως ότου ο `depth` φτάσει στο 0.

2. **draw**:
    - Χρησιμοποιεί τη συνάρτηση `drawSquareFractal` για να ξεκινήσει το fractal από το κέντρο της οθόνης με συγκεκριμένο μέγεθος και βάθος.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων για να κρατήσει ανοιχτό το παράθυρο.

Αυτό το πρόγραμμα θα δημιουργήσει ένα square fractal, το οποίο γίνεται πιο πυκνό καθώς μειώνεται το βάθος, δημιουργώντας εντυπωσιακά επαναλαμβανόμενα μοτίβα από τετράγωνα.

Carpet Fractal



Το *Carpet Fractal* είναι γνωστό και ως *Sierpinski Carpet* και δημιουργείται αφαιρώντας ένα κεντρικό τετράγωνο από ένα μεγαλύτερο τετράγωνο. Στη συνέχεια, η διαδικασία επαναλαμβάνεται αναδρομικά για κάθε υποτετράγωνο. Κάθε επόμενο επίπεδο αναδρομής διαιρεί το τετράγωνο σε 9 ίσα τμήματα, αφαιρώντας το κεντρικό τετράγωνο, δημιουργώντας έτσι ένα μοναδικό fractal μοτίβο.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600.0f;
const float WINDOW_HEIGHT = 600.0f;

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός Carpet Fractal.
```

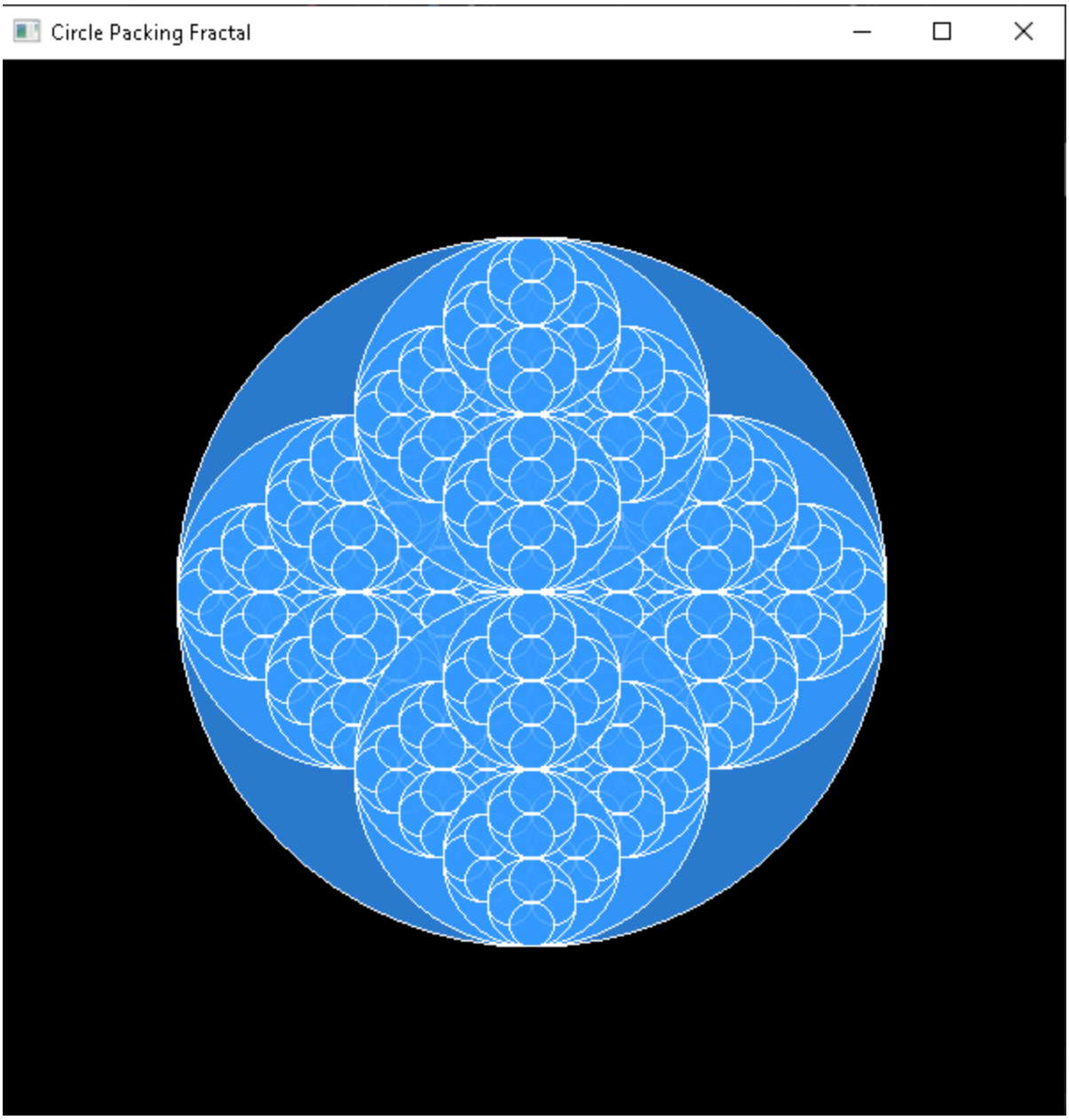
```
 *        Η συνάρτηση δημιουργεί ένα κεντρικό τετράγωνο και στη συνέχεια
 *        το διαιρεί σε μικρότερα τετράγωνα γύρω από το κέντρο του.
 * @param x Η συντεταγμένη x του κέντρου του τρέχοντος τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τρέχοντος τετραγώνου.
 * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου.
 * @param depth Το βάθος αναδρομής. Αν το βάθος φτάσει στο 0, σταματάμε τη διαδικασία.
 * @param brush Το πινέλο που χρησιμοποιείται για τη σχεδίαση των τετραγώνων.
 */
void drawCarpetFractal(float x, float y, float size, int depth, const graphics::Brush&
brush) {
    // Βάση αναδρομής: αν το βάθος είναι 0, τερματίζουμε την αναδρομή
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου για το τρέχον επίπεδο
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του νέου μεγέθους για τα υποτετράγωνα στο επόμενο επίπεδο αναδρομής
    float newSize = size / 3;

    // Αναδρομική κλήση για κάθε μία από τις 8 θέσεις γύρω από το κεντρικό τετράγωνο
    for (int dx = -1; dx <= 1; ++dx) {
        for (int dy = -1; dy <= 1; ++dy) {
            // Παράκαμψη του κεντρικού τετραγώνου για να αφήσουμε κενό το κέντρο
            if (dx == 0 && dy == 0) continue;

            // Αναδρομική κλήση για σχεδίαση υποτετραγώνου στη θέση (dx, dy)
            drawCarpetFractal(x + dx * newSize, y + dy * newSize, newSize, depth - 1,
brush);
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ της
οθόνης.
 *        Καθορίζει το χρώμα του fractal και ξεκινά τη σχεδίαση του Carpet Fractal από
το κέντρο του παραθύρου.
 */
void draw() {
    // Ρύθμιση πινέλου για το μπλε χρώμα του fractal
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 1.0f;  // Μπλε χρώμα
    brush.fill_opacity = 0.7f;   // Ημιδιαφανές πινέλο

    // Έναρξη της αναδρομικής σχεδίασης του fractal από το κέντρο του παραθύρου
    drawCarpetFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 600, 4, brush);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον
βρόχο ανανέωσης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Carpet Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Carpet Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();
```

```
    return 0;
}
```

εξήγηση Κώδικα

1. **drawCarpetFractal**:
    - Αυτή η συνάρτηση δημιουργεί ένα fractal μοτίβο από τετράγωνα.
    - Ξεκινάει με το αρχικό τετράγωνο και αναδρομικά διαιρεί κάθε τετράγωνο σε 9 τμήματα, αφαιρώντας το κεντρικό υποτετράγωνο και συνεχίζοντας στα άλλα 8.
    - Το `depth` καθορίζει τον αριθμό των επιπέδων αναδρομής, προσδιορίζοντας πόσο "βαθύ" είναι το fractal.
2. **draw**:
    - Ορίζει το χρώμα του fractal σε μπλε και καλεί τη συνάρτηση `drawCarpetFractal` από το κέντρο του παραθύρου.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Circle Packing Fractal



Το *Circle Packing Fractal* δημιουργείται τοποθετώντας κύκλους μέσα σε έναν μεγαλύτερο κύκλο και, στη συνέχεια, γεμίζοντας τα κενά με μικρότερους κύκλους, επαναλαμβάνοντας τη διαδικασία αναδρομικά. Αυτό το fractal παράγει μια εντυπωσιακή εικόνα με ομόκεντρους και μικρότερους κύκλους που γεμίζουν το χώρο.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;

/**
```

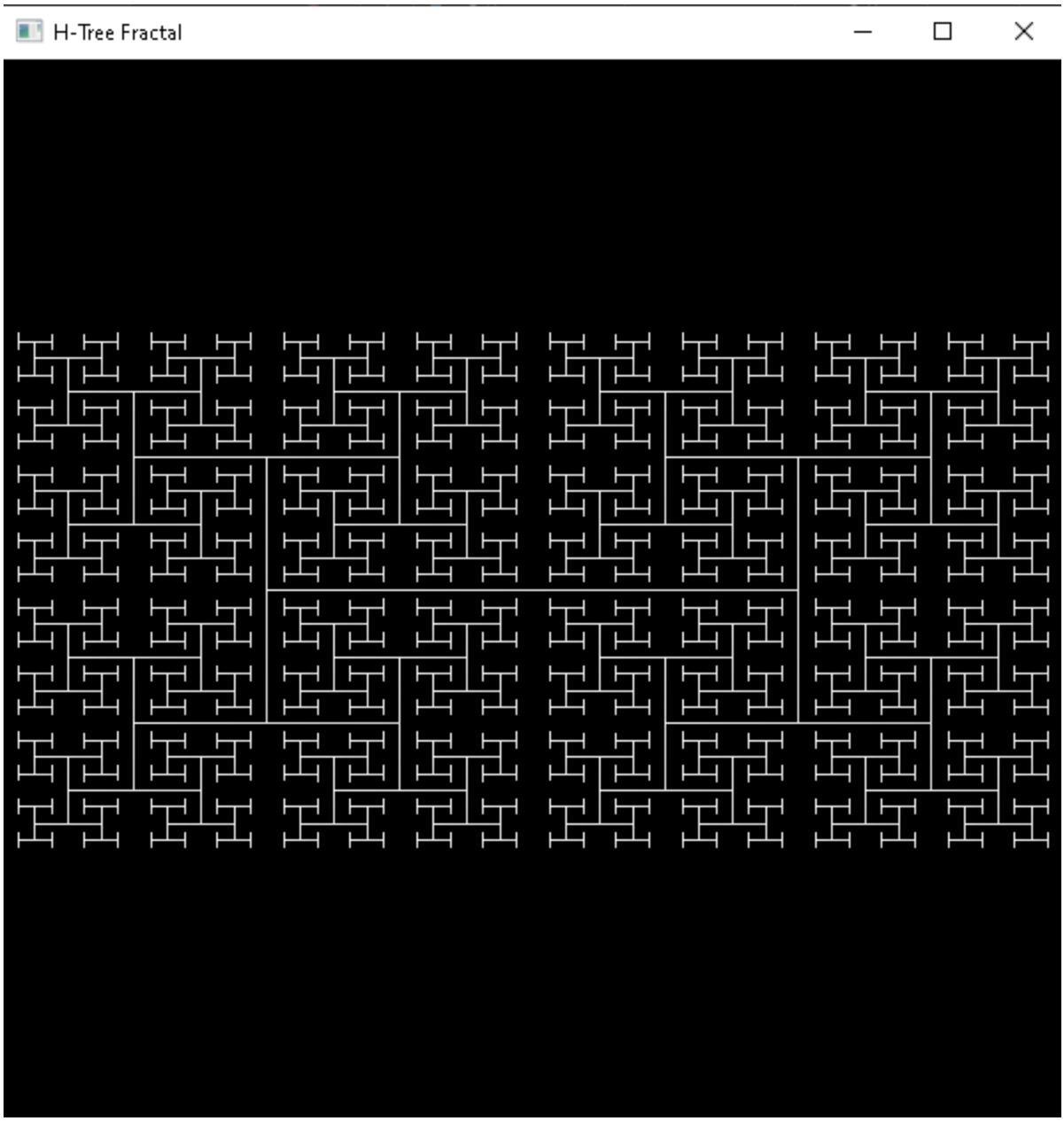
```
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του Circle Packing Fractal.
 *        Σχεδιάζει έναν κεντρικό κύκλο και μικρότερους κύκλους γύρω από αυτόν σε
καθορισμένες θέσεις.
 * @param x Η x συντεταγμένη του κέντρου του τρέχοντος κύκλου.
 * @param y Η y συντεταγμένη του κέντρου του τρέχοντος κύκλου.
 * @param radius Η ακτίνα του τρέχοντος κύκλου.
 * @param depth Το βάθος της αναδρομής. Αν είναι 0 ή η ακτίνα είναι πολύ μικρή, η
αναδρομή σταματά.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ του κύκλου.
 */
void drawCirclePackingFractal(float x, float y, float radius, int depth, const
graphics::Brush& brush) {
    // Βάση αναδρομής: Αν το βάθος είναι 0 ή η ακτίνα είναι πολύ μικρή, σταματάμε
    if (depth == 0 || radius < 1) return;

    // Σχεδίαση του κύριου κύκλου για το τρέχον επίπεδο
    graphics::drawDisk(x, y, radius, brush);

    // Μείωση ακτίνας για τους υποκύκλους στο επόμενο επίπεδο
    float newRadius = radius / 2;

    // Αναδρομικές κλήσεις για να δημιουργήσουμε υποκύκλους γύρω από τον κεντρικό
κύκλο
    drawCirclePackingFractal(x + newRadius, y, newRadius, depth - 1, brush);  // Δεξιά
    drawCirclePackingFractal(x - newRadius, y, newRadius, depth - 1, brush);  //
Αριστερά
    drawCirclePackingFractal(x, y + newRadius, newRadius, depth - 1, brush);  // Κάτω
    drawCirclePackingFractal(x, y - newRadius, newRadius, depth - 1, brush);  // Πάνω
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ της
οθόνης.
 *        Καθορίζει το χρώμα του fractal και καλεί τη συνάρτηση
`drawCirclePackingFractal` από το κέντρο του παραθύρου.
 */
void draw() {
    // Ρύθμιση πινέλου με ανοιχτό μπλε χρώμα και ημιδιαφανές αποτέλεσμα
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;
    brush.fill_color[1] = 0.6f;
    brush.fill_color[2] = 1.0f;  // Ανοιχτό μπλε χρώμα
    brush.fill_opacity = 0.8f;

    // Ξεκινάμε την αναδρομική σχεδίαση του fractal από το κέντρο του παραθύρου με
αρχική ακτίνα 200
    drawCirclePackingFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200, 5, brush);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον
βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με διαστάσεις 600x600 και τίτλο "Circle Packing Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Circle Packing Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για τη συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
```

```
}
```

Επεξήγηση Κώδικα

1. **drawCirclePackingFractal**:
    - Αυτή η συνάρτηση σχεδιάζει έναν κύκλο και κατόπιν δημιουργεί τέσσερις μικρότερους κύκλους (σε κάθετες και οριζόντιες κατευθύνσεις) γύρω από τον κεντρικό.
    - Ορίζει την ακτίνα των υποκύκλων στο μισό της ακτίνας του γονικού κύκλου.
    - Επαναλαμβάνει τη διαδικασία αναδρομικά, μειώνοντας κάθε φορά το `depth`, το οποίο καθορίζει το πόσα επίπεδα κύκλων θα δημιουργηθούν.
2. **draw**:
    - Ορίζει το χρώμα του fractal σε ανοιχτό μπλε και ξεκινά τη διαδικασία σχεδίασης από το κέντρο του παραθύρου με ακτίνα 200.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

H-Tree Fractal



Το *H-Tree Fractal* δημιουργείται ξεκινώντας από ένα σχήμα "Η" και επαναλαμβάνοντας τη διαδικασία αναδρομικά, τοποθετώντας μικρότερα "Η" σε κάθε άκρο του προηγούμενου "Η". Το αποτέλεσμα είναι ένα φράκταλ που θυμίζει διακλαδώσεις, με συμμετρικά τοποθετημένες γραμμές.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του H-Tree Fractal.
```

```
 *        Σχεδιάζει ένα "H" και στη συνέχεια μικρότερα "H" στα 4 άκρα του τρέχοντος
σχήματος.
 * @param x Η x συντεταγμένη του κέντρου του τρέχοντος "H".
 * @param y Η y συντεταγμένη του κέντρου του τρέχοντος "H".
 * @param length Το μήκος των γραμμών του "H".
 * @param depth Το βάθος της αναδρομής. Όταν φτάνει το 0, η αναδρομή σταματά.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ του "H".
 */
void drawHTree(float x, float y, float length, int depth, const graphics::Brush&
brush) {
    // Βάση αναδρομής: Αν το βάθος είναι 0 ή το μήκος των γραμμών είναι πολύ μικρό,
σταματάμε
    if (depth == 0 || length < 2) return;

    // Υπολογισμός των μισών και του ενός τετάρτου του μήκους των γραμμών
    float halfLength = length / 2;
    float quarterLength = length / 4;

    // Σχεδίαση των δύο κάθετων γραμμών του "H"
    graphics::drawLine(x - halfLength, y - quarterLength, x - halfLength, y +
quarterLength, brush);
    graphics::drawLine(x + halfLength, y - quarterLength, x + halfLength, y +
quarterLength, brush);

    // Σχεδίαση της οριζόντιας γραμμής του "H"
    graphics::drawLine(x - halfLength, y, x + halfLength, y, brush);

    // Αναδρομική κλήση για να σχεδιάσουμε μικρότερα "H" στα 4 άκρα του τρέχοντος "H"
    drawHTree(x - halfLength, y - quarterLength, length / 2, depth - 1, brush); //
Αριστερά πάνω
    drawHTree(x + halfLength, y - quarterLength, length / 2, depth - 1, brush); //
Δεξιά πάνω
    drawHTree(x - halfLength, y + quarterLength, length / 2, depth - 1, brush); //
Αριστερά κάτω
    drawHTree(x + halfLength, y + quarterLength, length / 2, depth - 1, brush); //
Δεξιά κάτω
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ της
οθόνης.
 *        Καθορίζει το χρώμα του fractal και καλεί τη συνάρτηση `drawHTree` από το
κέντρο του παραθύρου.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f;  // Μαύρο χρώμα για το fractal
    brush.outline_opacity = 1.0f;

    // Εκκίνηση της σχεδίασης του H-Tree από το κέντρο του παραθύρου με αρχικό μήκος
300
    drawHTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 300, 5, brush);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον
βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με διαστάσεις 600x600 και τίτλο "H-Tree Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "H-Tree Fractal");
```

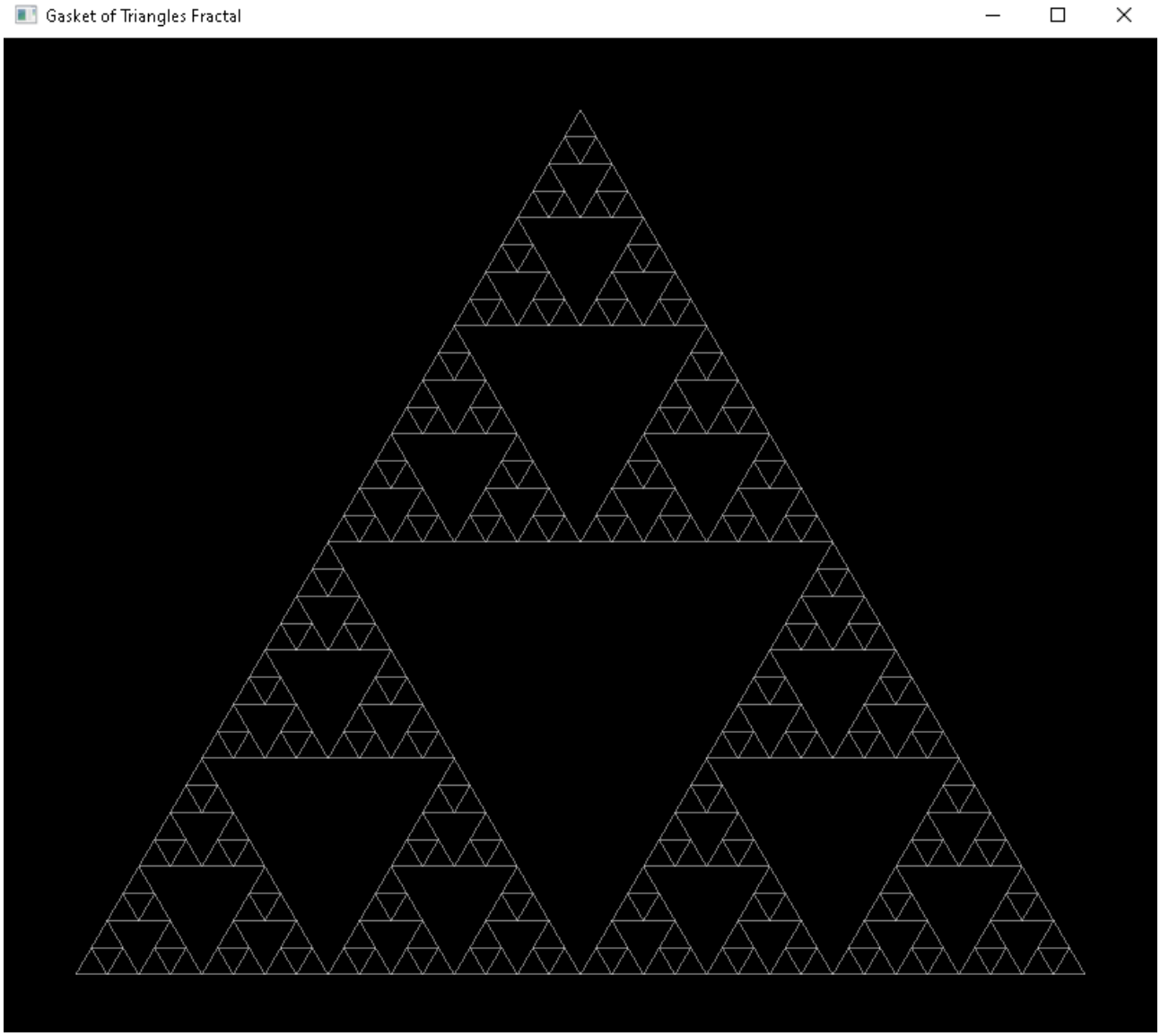
```
    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για τη συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawHTree**:
    - Αυτή η συνάρτηση σχεδιάζει το σχήμα "H" με βάση το κεντρικό σημείο `(x, y)` και το μήκος των γραμμών `length`.
    - Υπολογίζει τα τέσσερα άκρα του "H" για να τοποθετήσει τα επόμενα μικρότερα "H".
    - Καλεί αναδρομικά τη συνάρτηση για κάθε ένα από τα τέσσερα άκρα με το μισό μήκος γραμμής.
2. **draw**:
    - Ορίζει το χρώμα του fractal και ξεκινά την αναδρομική διαδικασία σχεδίασης από το κέντρο του παραθύρου.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων της βιβλιοθήκης SGG.

Με αυτό το πρόγραμμα, το *H-Tree Fractal* σχεδιάζεται χρησιμοποιώντας τη γεωμετρία του "H" και την αναδρομικότητα, δημιουργώντας ένα εντυπωσιακό συμμετρικό σχέδιο που επεκτείνεται σε όλα τα άκρα.

Gasket of Triangles Fractal



Το *Gasket of Triangles Fractal* (επίσης γνωστό ως Sierpinski Triangle) δημιουργείται διαιρώντας ένα τρίγωνο σε τρία μικρότερα ίσα τρίγωνα, αφήνοντας το κεντρικό κομμάτι κενό. Αυτή η διαδικασία επαναλαμβάνεται αναδρομικά σε κάθε μικρότερο τρίγωνο, δημιουργώντας ένα φράκταλ που μοιάζει με τρίγωνα μέσα σε τρίγωνα.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 700;

/**
 * @brief Σχεδιάζει ένα τρίγωνο χρησιμοποιώντας τις τρεις κορυφές του.
 * @param x1 Η x συντεταγμένη της πρώτης κορυφής.
 * @param y1 Η y συντεταγμένη της πρώτης κορυφής.
 * @param x2 Η x συντεταγμένη της δεύτερης κορυφής.
```

```
 * @param y2 Η y συντεταγμένη της δεύτερης κορυφής.
 * @param x3 Η x συντεταγμένη της τρίτης κορυφής.
 * @param y3 Η y συντεταγμένη της τρίτης κορυφής.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, const
graphics::Brush& brush) {
    // Σχεδίαση των τριών πλευρών του τριγώνου
    graphics::drawLine(x1, y1, x2, y2, brush);
    graphics::drawLine(x2, y2, x3, y3, brush);
    graphics::drawLine(x3, y3, x1, y1, brush);
}

/**
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του fractal Gasket of Triangles.
 *        Σχεδιάζει ένα τρίγωνο και το διαιρεί αναδρομικά σε τρία μικρότερα τρίγωνα.
 * @param x1 Η x συντεταγμένη της πρώτης κορυφής του τριγώνου.
 * @param y1 Η y συντεταγμένη της πρώτης κορυφής του τριγώνου.
 * @param x2 Η x συντεταγμένη της δεύτερης κορυφής του τριγώνου.
 * @param y2 Η y συντεταγμένη της δεύτερης κορυφής του τριγώνου.
 * @param x3 Η x συντεταγμένη της τρίτης κορυφής του τριγώνου.
 * @param y3 Η y συντεταγμένη της τρίτης κορυφής του τριγώνου.
 * @param depth Το βάθος της αναδρομής. Όταν φτάνει το 0, σταματάει η αναδρομή.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawGasket(float x1, float y1, float x2, float y2, float x3, float y3, int depth,
const graphics::Brush& brush) {
    // Βάση αναδρομής: αν το βάθος είναι 0, σχεδιάζουμε το τρίγωνο και επιστρέφουμε
    if (depth == 0) {
        drawTriangle(x1, y1, x2, y2, x3, y3, brush);
        return;
    }

    // Υπολογισμός των μεσαίων σημείων των πλευρών
    float mx1 = (x1 + x2) / 2;
    float my1 = (y1 + y2) / 2;
    float mx2 = (x2 + x3) / 2;
    float my2 = (y2 + y3) / 2;
    float mx3 = (x1 + x3) / 2;
    float my3 = (y1 + y3) / 2;

    // Αναδρομική κλήση για τα τρία περιφερειακά τρίγωνα
    drawGasket(x1, y1, mx1, my1, mx3, my3, depth - 1, brush);
    drawGasket(mx1, my1, x2, y2, mx2, my2, depth - 1, brush);
    drawGasket(mx3, my3, mx2, my2, x3, y3, depth - 1, brush);
}

/**
 * @brief Κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ
της οθόνης.
 *        Ρυθμίζει το χρώμα του fractal και καλεί τη συνάρτηση `drawGasket` για την
εκκίνηση της σχεδίασης.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_opacity = 1.0f;
    brush.outline_opacity = 0.5f;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f; // Μαύρο χρώμα για το fractal

    // Εκκίνηση σχεδίασης του Gasket of Triangles με το αρχικό τρίγωνο και βάθος 5
    drawGasket(400, 50, 50, 650, 750, 650, 5, brush);
}
```

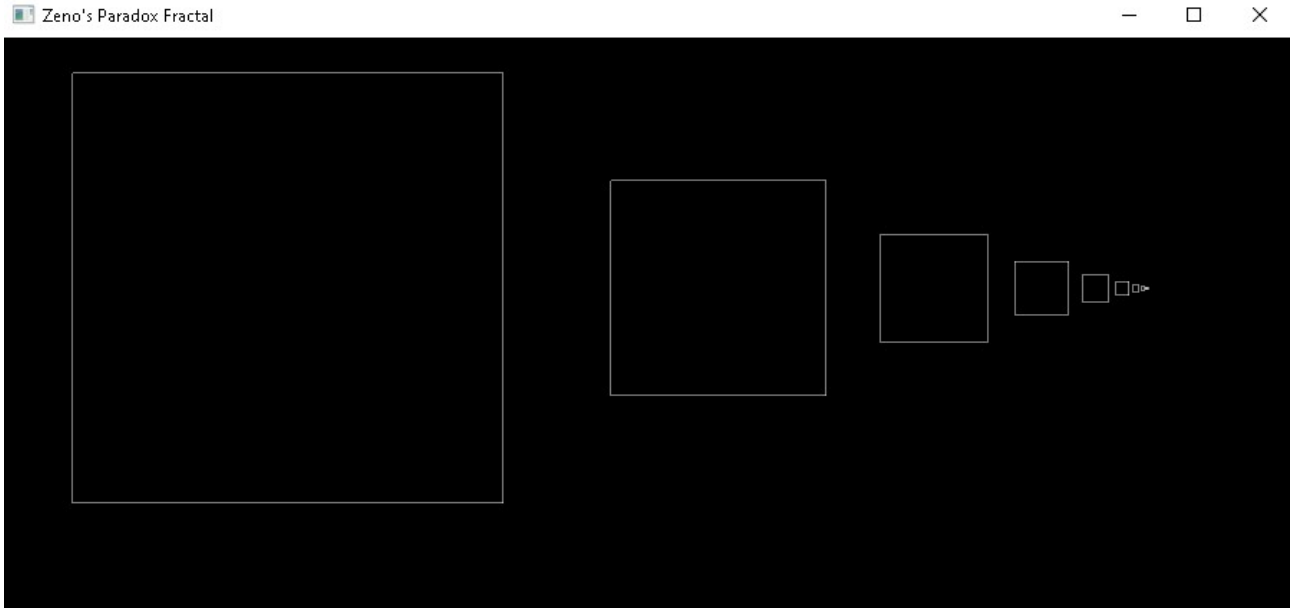
```
/**
 * @brief Κύρια συνάρτηση του προγράμματος. Δημιουργεί το παράθυρο και ξεκινά τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με διαστάσεις 800x700 και τίτλο "Gasket of Triangles Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Gasket of Triangles Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για τη συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawTriangle**:
    - Σχεδιάζει ένα τρίγωνο με τις κορυφές `(x1, y1)`, `(x2, y2)`, και `(x3, y3)`.
    - Χρησιμοποιείται για τη βασική σχεδίαση κάθε τριγώνου στο φράκταλ.
2. **drawGasket**:
    - Αναδρομική συνάρτηση που σχεδιάζει το Gasket of Triangles.
    - Σε κάθε βήμα, υπολογίζει τα μεσαία σημεία των πλευρών και καλεί τη συνάρτηση ξανά για τα τρία εξωτερικά τρίγωνα.
    - Σταματά την αναδρομή όταν το βάθος φτάσει στο 0, όπου και σχεδιάζει το τελικό τρίγωνο.
3. **draw**:
    - Αρχικοποιεί το `Brush` και καθορίζει το χρώμα για το φράκταλ.
    - Καλεί τη συνάρτηση `drawGasket` για το αρχικό τρίγωνο στο κέντρο του παραθύρου.
4. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων της SGG.

Αυτός ο κώδικας δημιουργεί το Gasket of Triangles Fractal με βάση την αναδρομή, δημιουργώντας συμμετρικά μικρότερα τρίγωνα σε κάθε βήμα.

Zeno’s Paradox Fractal



Το *Zeno's Paradox Fractal* βασίζεται στο παράδοξο του Ζήνωνα, όπου ένα αντικείμενο πλησιάζει έναν προορισμό σε συνεχώς μικρότερα βήματα, χωρίς ποτέ να τον φτάνει. Στην οπτική απεικόνιση του fractal, χρησιμοποιείται μια γραμμική διάταξη τετραγώνων, με κάθε τετράγωνο να είναι η μισή απόσταση του προηγούμενου, σχηματίζοντας ένα επαναλαμβανόμενο μοτίβο που δείχνει την ιδέα της ατέρμονης προσέγγισης.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>

// Διαστάσεις παραθύρου σχεδίασης
const float WINDOW_WIDTH = 900;
const float WINDOW_HEIGHT = 400;

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Zeno's Paradox Fractal.
 *        Σχεδιάζει ένα τετράγωνο και καλεί αναδρομικά τη συνάρτηση για το μισού μεγέθους τετράγωνο, που τοποθετείται στα δεξιά.
 * @param x Η x συντεταγμένη του τετραγώνου.
 * @param y Η y συντεταγμένη του τετραγώνου.
 * @param size Το μέγεθος του τετραγώνου (πλάτος και ύψος).
 * @param depth Το βάθος αναδρομής. Όταν φτάνει το 0, σταματά η αναδρομή.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawZenoFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: Αν το βάθος είναι 0, σταματάμε
    if (depth == 0) return;

    // Σχεδίαση τετραγώνου
    graphics::drawRect(x, y, size, size, brush);

    // Αναδρομική κλήση για το επόμενο μικρότερο τετράγωνο
    // Το νέο τετράγωνο τοποθετείται δίπλα από το τρέχον, έχει μέγεθος στο μισό και βάθος μειωμένο κατά 1
    drawZenoFractal(x + size, y, size / 2, depth - 1, brush);
```

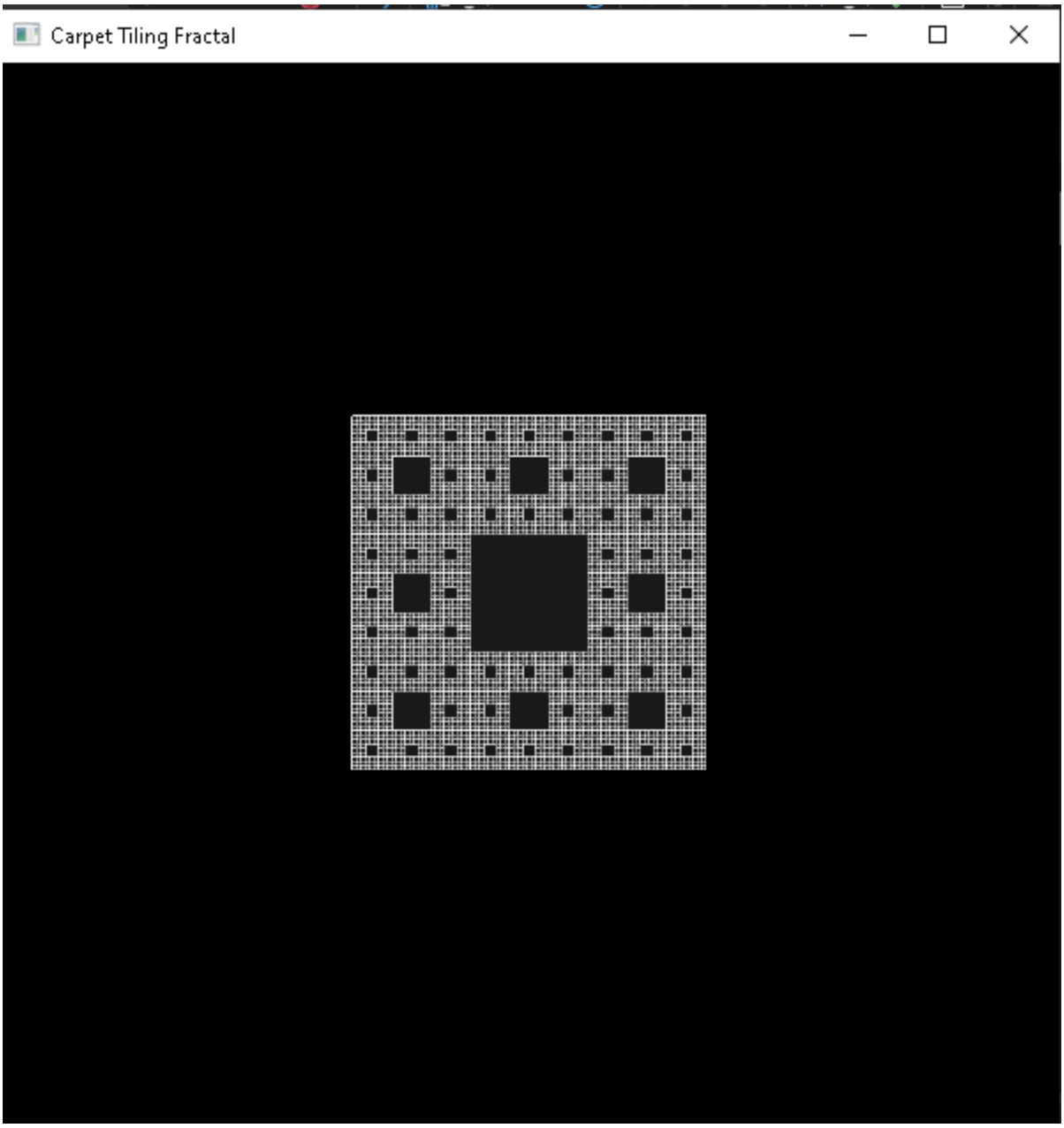
```
}

/**
 * @brief Κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
 *        Καθορίζει τις παραμέτρους σχεδίασης για το fractal και εκκινεί τη
διαδικασία.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;  // Μαύρο χρώμα γέμισμα για τα τετράγωνα
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f;
    brush.outline_opacity = 0.5f;  // Διαφάνεια περιγράμματος

    // Εκκίνηση του fractal από το σημείο (200, κεντρικό ύψος) με αρχικό μέγεθος 300
και βάθος αναδρομής 20
    drawZenoFractal(200, WINDOW_HEIGHT / 2 - 25, 300, 20, brush);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος που αρχικοποιεί το παράθυρο σχεδίασης και
εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τις διαστάσεις που καθορίστηκαν και τίτλο "Zeno's
Paradox Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Zeno's Paradox Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων για συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawZenoFractal**:
    - Αναδρομική συνάρτηση που σχεδιάζει το Zeno’s Paradox Fractal.
    - Σχεδιάζει ένα τετράγωνο στο σημείο `(x, y)` με το καθορισμένο μέγεθος.
    - Στη συνέχεια, καλεί τον εαυτό της για να σχεδιάσει το επόμενο τετράγωνο στο μισό της απόστασης και του μεγέθους, μειώνοντας το βάθος της αναδρομής κατά 1.
2. **draw**:
    - Αρχικοποιεί το `Brush` και καθορίζει το χρώμα του fractal.
    - Καλεί τη συνάρτηση `drawZenoFractal` για να σχεδιάσει το fractal από την αρχική θέση `(50, 100)`.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων της SGG.

Αυτό το fractal δείχνει τη σταδιακή μείωση του μεγέθους των τετραγώνων καθώς προχωρούν προς τα δεξιά, απεικονίζοντας οπτικά το παράδοξο του Ζήνωνα.

Carpet Tiling Fractal



Το *Carpet Tiling Fractal* είναι ένας τύπος fractal που μοιάζει με χαλί και δημιουργείται επαναληπτικά τοποθετώντας τετράγωνα μέσα σε τετράγωνα. Ένα κλασικό παράδειγμα είναι το *Sierpinski Carpet*, όπου κάθε τετράγωνο χωρίζεται σε 9 υπο-τετράγωνα, και το μεσαίο αφαιρείται για να δημιουργήσει το fractal μοτίβο.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;

/**
```

```
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το Carpet Tiling Fractal.
 *        Σε κάθε αναδρομικό επίπεδο, το κεντρικό τετράγωνο διαιρείται σε εννέα
μικρότερα τετράγωνα.
 *        Η συνάρτηση επαναλαμβάνει αυτή τη διαδικασία μόνο για τα περιφερειακά
τετράγωνα.
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του τετραγώνου.
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του τετραγώνου.
 * @param size Το μέγεθος της πλευράς του τετραγώνου.
 * @param depth Το βάθος αναδρομής, το οποίο μειώνεται σε κάθε κλήση.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawCarpetFractal(float x, float y, float size, int depth, const graphics::Brush&
brush) {
    // Βάση αναδρομής: Όταν το βάθος φτάσει στο 0, σταματά η αναδρομή
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου στο τρέχον επίπεδο
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του μεγέθους για τα υποτετράγωνα, το οποίο είναι το 1/3 του
τρέχοντος τετραγώνου
    float newSize = size / 3;

    // Αναδρομική σχεδίαση των 8 περιφερειακών τετραγώνων γύρω από το κεντρικό
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            // Παράλειψη του κεντρικού τετραγώνου (όταν dx και dy είναι 0)
            if (dx != 0 || dy != 0) {
                // Σχεδίαση υποτετραγώνου στη θέση που υποδεικνύεται από dx και dy
                drawCarpetFractal(x + dx * newSize, y + dy * newSize, newSize, depth -
1, brush);
            }
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
 *        Καθορίζει τις παραμέτρους για το fractal και ξεκινά τη διαδικασία σχεδίασης.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f;  // Σκούρο γκρι χρώμα για το fractal
    brush.fill_color[1] = 0.1f;
    brush.fill_color[2] = 0.1f;
    brush.outline_opacity = 0.5f;  // Ελαφρώς ημιδιαφανές περίγραμμα

    // Εκκίνηση της σχεδίασης από το κέντρο του παραθύρου με μέγεθος 1/3 του παραθύρου
και βάθος 5
    drawCarpetFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 3, 5,
brush);
}

/**
 * @brief Η κύρια συνάρτηση που αρχικοποιεί το παράθυρο σχεδίασης και εκκινεί το βρόχο
σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου σχεδίασης με τίτλο "Carpet Tiling Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Carpet Tiling Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);
```

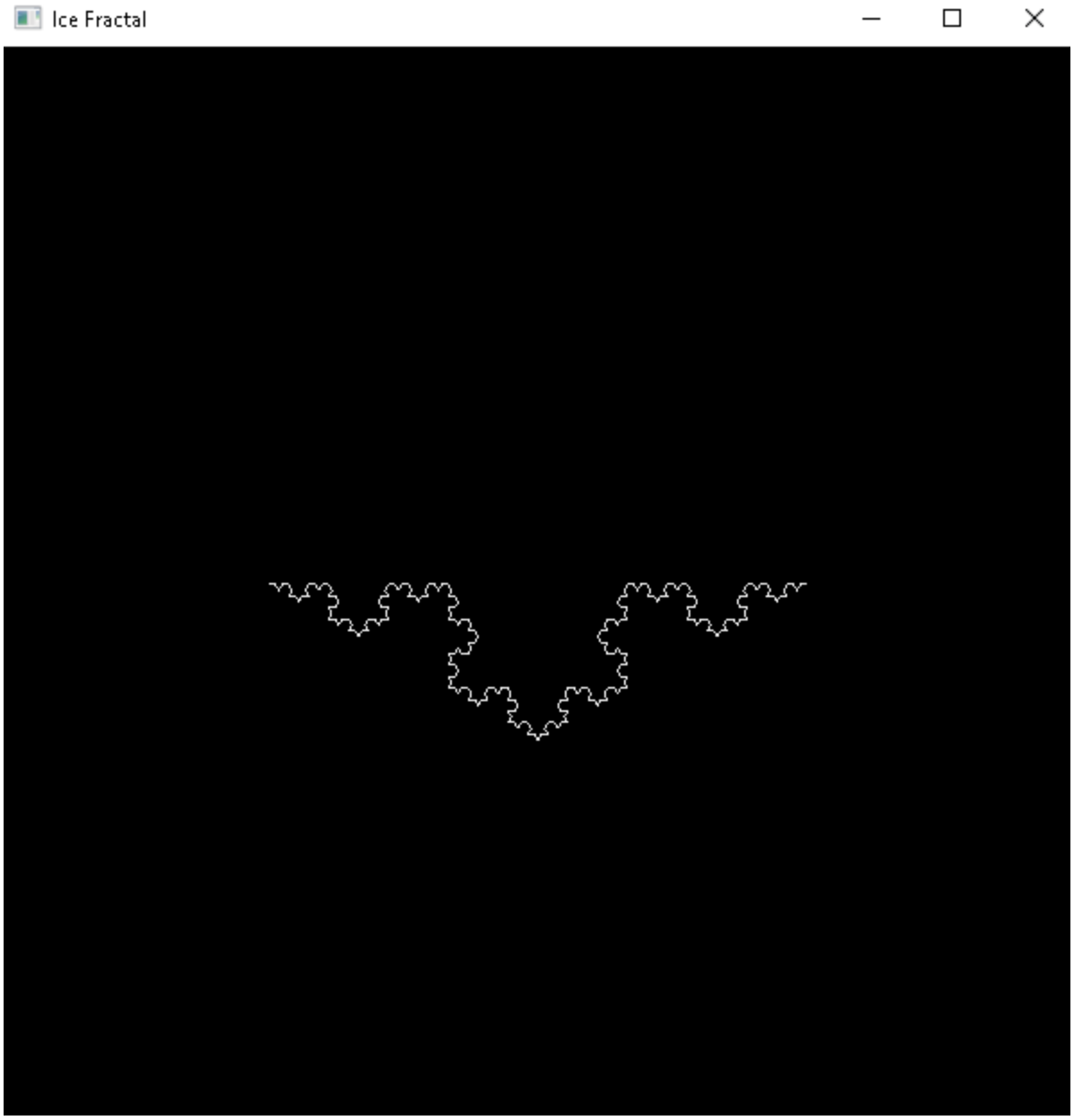
```
    // Εκκίνηση του βρόχου μηνυμάτων της βιβλιοθήκης για συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawCarpetFractal**:
    - Αναδρομική συνάρτηση για τη σχεδίαση του fractal.
    - Σχεδιάζει ένα κεντρικό τετράγωνο με το καθορισμένο `size`.
    - Υπολογίζει το νέο μέγεθος για τα υποτετράγωνα και καλεί αναδρομικά τον εαυτό της για να σχεδιάσει τα 8 περιφερειακά τετράγωνα, αφήνοντας το κεντρικό κενό.
2. **draw**:
    - Αρχικοποιεί το `Brush` και καθορίζει το χρώμα του fractal.
    - Καλεί τη συνάρτηση `drawCarpetFractal` για να σχεδιάσει το fractal από το κέντρο του παραθύρου με συγκεκριμένο μέγεθος και βάθος.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων της SGG.

Αυτό το fractal δημιουργεί το χαρακτηριστικό μοτίβο χαλιού, όπου κάθε νέα γενιά προσθέτει μικρότερα τετράγωνα σε θέσεις που σχηματίζουν ένα επαναλαμβανόμενο μοτίβο.

Ice Fractal



Το *Ice Fractal* είναι ένα γεωμετρικό fractal που βασίζεται στη συμμετρία και την αναπαραγωγή μοτίβων που μοιάζουν με κρυστάλλους πάγου ή νιφάδες χιονιού. Συνήθως ξεκινά από μια ευθεία γραμμή που επαναλαμβάνεται με συμμετρία και μετασχηματισμούς.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <iostream>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;

/**
```

```
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Ice Fractal.
 *        Σε κάθε επανάληψη, η γραμμή χωρίζεται σε τρία τμήματα, ενώ προστίθεται ένα
επιπλέον σημείο
 *        για να σχηματίσει γωνία στο κέντρο της γραμμής, δίνοντας την εμφάνιση
παγοκρυστάλλου.
 *
 * @param x1 Η x-συντεταγμένη του αρχικού σημείου της γραμμής.
 * @param y1 Η y-συντεταγμένη του αρχικού σημείου της γραμμής.
 * @param x2 Η x-συντεταγμένη του τελικού σημείου της γραμμής.
 * @param y2 Η y-συντεταγμένη του τελικού σημείου της γραμμής.
 * @param depth Το τρέχον βάθος αναδρομής, που καθορίζει το επίπεδο λεπτομέρειας του
fractal.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawIceFractal(float x1, float y1, float x2, float y2, int depth, const
graphics::Brush& brush) {
    // Βάση της αναδρομής: αν το βάθος φτάσει στο 0, σχεδιάζεται μια ευθεία γραμμή και
η αναδρομή σταματά
    if (depth == 0) {
        graphics::drawLine(x1, y1, x2, y2, brush);
        return;
    }

    // Υπολογισμός μετατόπισης για τα τμήματα της γραμμής
    float dx = x2 - x1;
    float dy = y2 - y1;

    // Σημεία διαίρεσης της γραμμής σε τρία μέρη
    float xA = x1 + dx / 3;
    float yA = y1 + dy / 3;
    float xB = x1 + 2 * dx / 3;
    float yB = y1 + 2 * dy / 3;

    // Υπολογισμός του σημείου που σχηματίζει την γωνία της γραμμής στο κέντρο
    float xPeak = (xA + xB) / 2 - (yB - yA) * std::sqrt(3) / 2;
    float yPeak = (yA + yB) / 2 + (xB - xA) * std::sqrt(3) / 2;

    // Αναδρομική κλήση για σχεδίαση των τεσσάρων τμημάτων που δημιουργούνται
    drawIceFractal(x1, y1, xA, yA, depth - 1, brush);         // Πρώτο τμήμα
    drawIceFractal(xA, yA, xPeak, yPeak, depth - 1, brush);   // Δεύτερο τμήμα με
γωνία
    drawIceFractal(xPeak, yPeak, xB, yB, depth - 1, brush);   // Τρίτο τμήμα με γωνία
    drawIceFractal(xB, yB, x2, y2, depth - 1, brush);         // Τέταρτο τμήμα
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
 *        Θέτει το αρχικό χρώμα και το αρχικό μέγεθος της γραμμής, και καλεί τη
συνάρτηση για να σχεδιάσει το Ice Fractal.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;  // Χρώμα μπλε
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 1.0f;

    // Ορισμός αρχικής ευθείας γραμμής στο κέντρο του παραθύρου
    float startX = WINDOW_WIDTH / 4;
    float startY = WINDOW_HEIGHT / 2;
    float endX = 3 * WINDOW_WIDTH / 4;
    float endY = WINDOW_HEIGHT / 2;

    // Σχεδίαση Ice Fractal με βάθος αναδρομής 4 για τη δημιουργία λεπτομερών
σχηματισμών
    drawIceFractal(startX, startY, endX, endY, 4, brush);
```

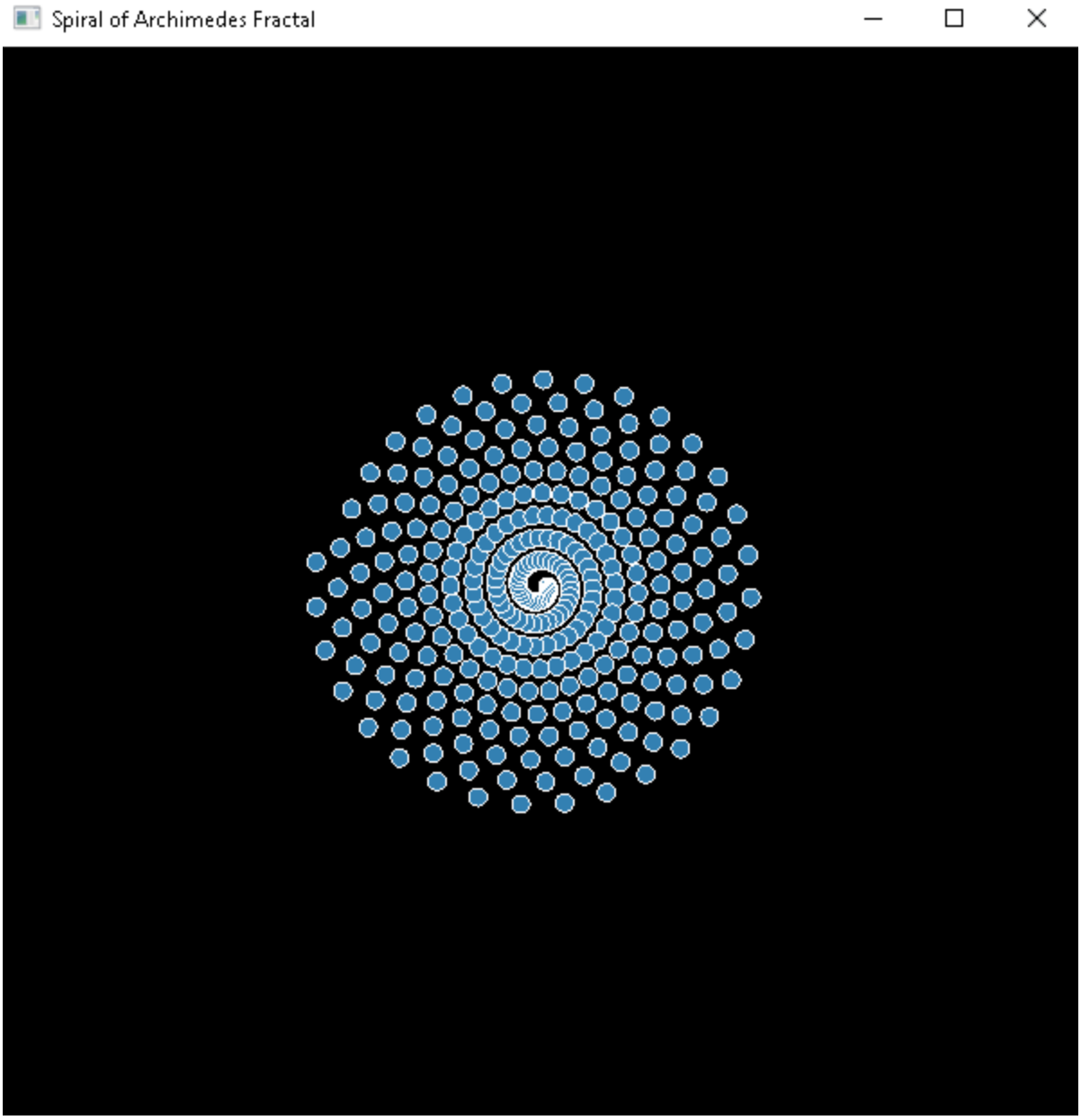
```
}

/**
 * @brief Η κύρια συνάρτηση που αρχικοποιεί το παράθυρο σχεδίασης και εκκινεί το βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου σχεδίασης με τίτλο "Ice Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Ice Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων της βιβλιοθήκης για συνεχή ενημέρωση του παραθύρου
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawIceFractal**:
    - Αναδρομική συνάρτηση που σχεδιάζει το Ice Fractal.
    - Ξεκινά με μια ευθεία γραμμή από το σημείο `(x1, y1)` στο σημείο `(x2, y2)`.
    - Διαιρεί τη γραμμή σε τρία τμήματα και δημιουργεί μια ακίδα στο κέντρο της γραμμής, υπολογίζοντας το σημείο της κορυφής (`xPeak`, `yPeak`).
    - Καλεί αναδρομικά τον εαυτό της για τα τέσσερα νέα τμήματα, μειώνοντας το `depth` κάθε φορά.
2. **draw**:
    - Ορίζει το χρώμα του fractal με το `brush`.
    - Καλεί τη συνάρτηση `drawIceFractal` για να σχεδιάσει το fractal με καθορισμένο αρχικό σημείο και βάθος.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Αυτό το fractal δημιουργεί ένα μοτίβο που θυμίζει κρύσταλλα πάγου, καθώς η γραμμή "σπάει" συμμετρικά, σχηματίζοντας αιχμηρές γωνίες που μοιάζουν με κρυστάλλους.

Spiral of Archimedes



Το *Spiral of Archimedes Fractal* είναι ένας τύπος fractal που χρησιμοποιεί τη σπείρα του Αρχιμήδη, στην οποία η απόσταση ανάμεσα στις περιελίξεις είναι σταθερή. Το fractal δημιουργείται με επαναλαμβανόμενα σχέδια που ακολουθούν την πορεία μιας σπείρας, σχηματίζοντας έναν εντυπωσιακό γεωμετρικό σχεδιασμό.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;

/**
```

```
 * @brief Συνάρτηση για τη σχεδίαση της σπείρας του Αρχιμήδη,
 *        χρησιμοποιώντας έναν αριθμό από μικρούς κύκλους σε κάθε σημείο της σπείρας.
 *
 * @param centerX Η x-συντεταγμένη του κέντρου της σπείρας.
 * @param centerY Η y-συντεταγμένη του κέντρου της σπείρας.
 * @param iterations Ο αριθμός των σημείων/επανάληψεων που σχεδιάζονται στη σπείρα.
 * @param a Ο αρχικός παράγοντας απόστασης από το κέντρο (αρχικό μέγεθος της σπείρας).
 * @param b Ο παράγοντας αύξησης της απόστασης για κάθε βήμα (ταχύτητα ανάπτυξης της
σπείρας).
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawArchimedeanSpiral(float centerX, float centerY, int iterations, float a,
float b, const graphics::Brush& brush) {
    // Αρχική τιμή γωνίας για την ανάπτυξη της σπείρας
    float angle = 0.0f;

    // Έναρξη επανάληψης για τον υπολογισμό και σχεδίαση των σημείων της σπείρας
    for (int i = 0; i < iterations; i++) {
        // Υπολογισμός της ακτίνας και των συντεταγμένων του νέου σημείου
        float r = a + b * angle;  // Ακτίνα με βάση τους παράγοντες a και b
        float x = centerX + r * cos(angle); // Υπολογισμός x-συντεταγμένης
        float y = centerY + r * sin(angle); // Υπολογισμός y-συντεταγμένης

        // Σχεδίαση ενός μικρού κύκλου στο υπολογισμένο σημείο της σπείρας
        graphics::drawDisk(x, y, 5, brush);

        // Αύξηση της γωνίας για το επόμενο σημείο, ελέγχοντας την απόσταση των
σημείων
        angle += 0.2f; // Ελέγχει την απόσταση μεταξύ των σημείων στη σπείρα
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *        Ρυθμίζει το χρώμα και το μέγεθος της σπείρας και καλεί τη συνάρτηση
σχεδίασης.
 */
void draw() {
    // Ρύθμιση του πινέλου για την εμφάνιση της σπείρας
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f; // Ρύθμιση μπλε-πράσινης απόχρωσης για το fractal
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.7f;

    // Ορισμός του κέντρου της σπείρας στο κέντρο του παραθύρου
    float centerX = WINDOW_WIDTH / 2;
    float centerY = WINDOW_HEIGHT / 2;

    // Κλήση της συνάρτησης σχεδίασης της σπείρας του Αρχιμήδη
    // Παράμετροι:
    // - 300 σημεία/επαναλήψεις
    // - α = 5 (αρχικό μέγεθος σπείρας)
    // - β = 2 (ρυθμός ανάπτυξης)
    drawArchimedeanSpiral(centerX, centerY, 300, 5, 2, brush);
}

/**
 * @brief Κύρια συνάρτηση που αρχικοποιεί το παράθυρο και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Spiral of Archimedes Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Spiral of Archimedes
Fractal");
```

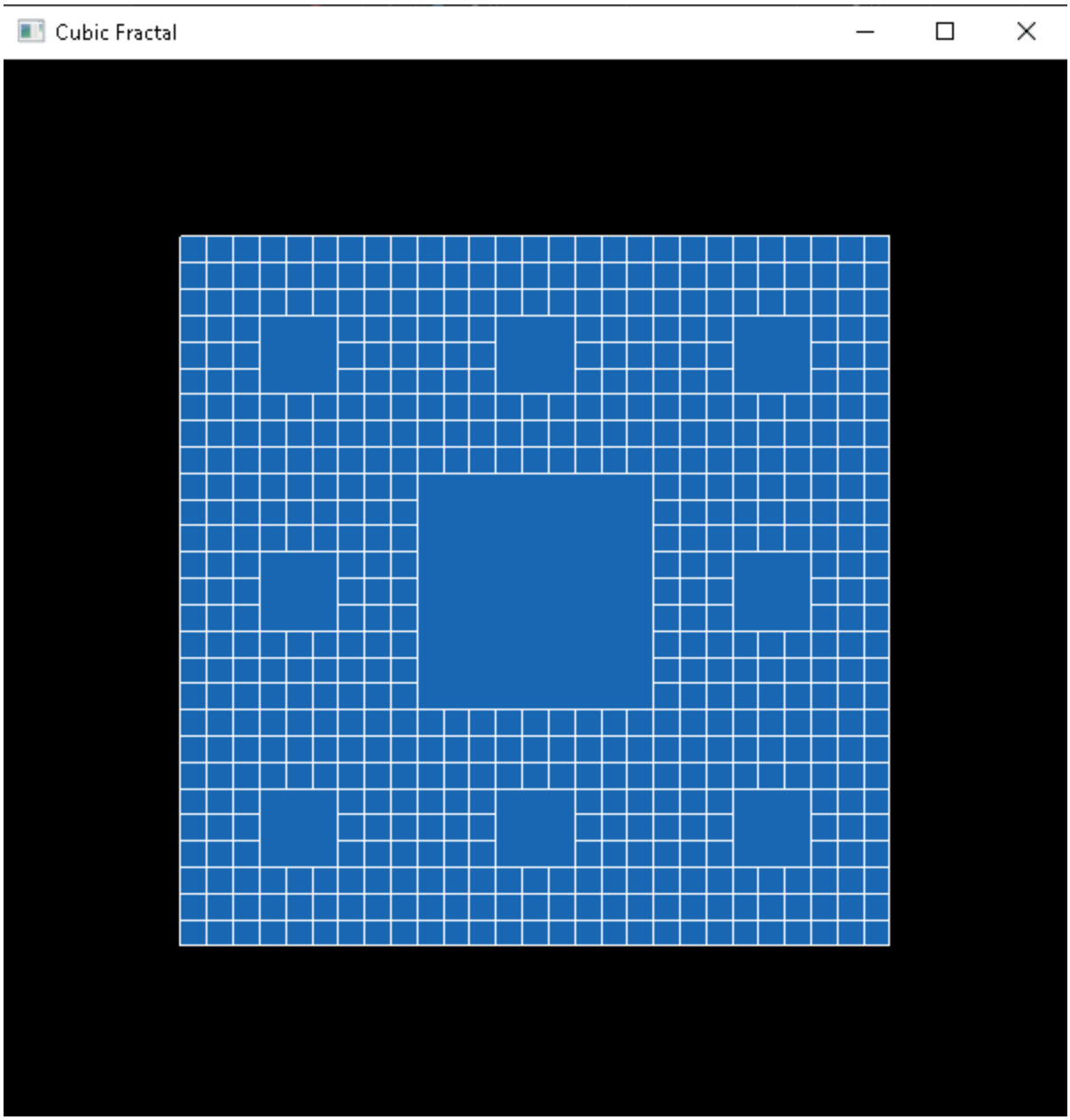
```
    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Εκκίνηση του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawArchimedeanSpiral**:
    - Υπολογίζει τις συντεταγμένες για κάθε σημείο της σπείρας.
    - Χρησιμοποιεί τους παράγοντες `a` και `b`, όπου:
        - `a` ορίζει την αρχική ακτίνα της σπείρας.
        - `b` ορίζει τον ρυθμό απόστασης των περιελίξεων.
    - Για κάθε σημείο, υπολογίζει τη νέα ακτίνα `r` και σχεδιάζει έναν μικρό κύκλο.
2. **draw**:
    - Ορίζει το χρώμα της σπείρας με το `brush`.
    - Καλεί τη συνάρτηση `drawArchimedeanSpiral` για να σχεδιάσει τη σπείρα με συγκεκριμένο κέντρο, αριθμό επαναλήψεων και παραμέτρους σπείρας.
3. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Αυτό το πρόγραμμα σχεδιάζει τη σπείρα του Αρχιμήδη ως fractal, όπου κάθε σημείο διαγράφει την πορεία μιας αναπτυσσόμενης σπείρας.

Cubic Fractal



Το *Cubic Fractal* είναι ένα γεωμετρικό fractal που χρησιμοποιεί τετράγωνα για τη δημιουργία ενός επαναλαμβανόμενου μοτίβου. Στο συγκεκριμένο fractal, ένα μεγάλο τετράγωνο χωρίζεται σε μικρότερα τετράγωνα σε κάθε επανάληψη, δημιουργώντας την ψευδαίσθηση της απείρως επαναλαμβανόμενης δομής.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

// Ορισμός των διαστάσεων του παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 4; // Ορισμός του μέγιστου βάθους αναδρομής για το fractal
```

```
/**
 * @brief Συνάρτηση σχεδίασης ενός τετραγώνου με συγκεκριμένες συντεταγμένες και
μέγεθος.
 *
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου.
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου.
 * @param side Το μήκος κάθε πλευράς του τετραγώνου.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawSquare(float x, float y, float side, const graphics::Brush& brush) {
    // Σχεδίαση τετραγώνου χρησιμοποιώντας την κεντρική θέση x, y και πλευρά side
    graphics::drawRect(x, y, side, side, brush);
}

/**
 * @brief Αναδρομική συνάρτηση που δημιουργεί το "Cubic Fractal" σχεδιάζοντας
τετράγωνα.
 *
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου.
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου.
 * @param side Το μήκος κάθε πλευράς του τετραγώνου για το τρέχον επίπεδο.
 * @param depth Το τρέχον επίπεδο βάθους αναδρομής.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawCubicFractal(float x, float y, float side, int depth, const graphics::Brush&
brush) {
    if (depth == 0) return; // Διακοπή αναδρομής όταν φτάσουμε στο μέγιστο βάθος

    // Σχεδίαση του κεντρικού τετραγώνου για το τρέχον επίπεδο
    drawSquare(x, y, side, brush);

    // Υπολογισμός νέου μεγέθους για τα υποτετράγωνα
    float newSide = side / 3.0f;
    int nextDepth = depth - 1;

    // Σχεδίαση 8 περιφερειακών τετραγώνων γύρω από το κεντρικό τετράγωνο
    drawCubicFractal(x - newSide, y - newSide, newSide, nextDepth, brush); //
Αριστερά-πάνω
    drawCubicFractal(x, y - newSide, newSide, nextDepth, brush);           // Κέντρο-
πάνω
    drawCubicFractal(x + newSide, y - newSide, newSide, nextDepth, brush); // Δεξιά-
πάνω

    drawCubicFractal(x - newSide, y, newSide, nextDepth, brush);           //
Αριστερά-κέντρο
    drawCubicFractal(x + newSide, y, newSide, nextDepth, brush);           // Δεξιά-
κέντρο

    drawCubicFractal(x - newSide, y + newSide, newSide, nextDepth, brush); //
Αριστερά-κάτω
    drawCubicFractal(x, y + newSide, newSide, nextDepth, brush);           // Κέντρο-
κάτω
    drawCubicFractal(x + newSide, y + newSide, newSide, nextDepth, brush); // Δεξιά-
κάτω
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *        Καθορίζει το κεντρικό τετράγωνο και ξεκινά την αναδρομική σχεδίαση του
fractal.
 */
void draw() {
    // Ρύθμιση του πινέλου σχεδίασης με μπλε-πράσινη απόχρωση
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f;
```

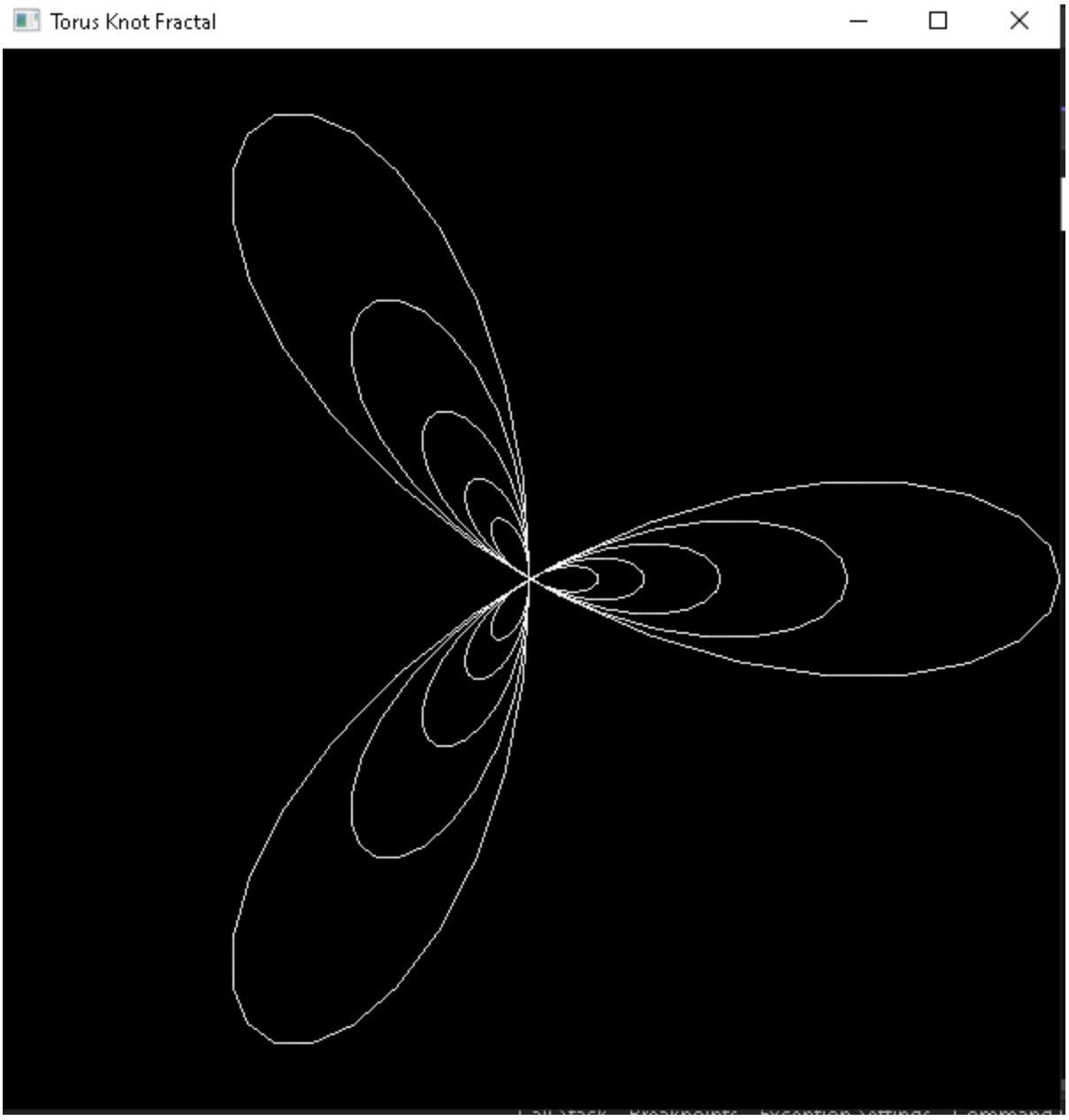
```
    brush.fill_color[1] = 0.4f;
    brush.fill_color[2] = 0.7f;

    // Τοποθέτηση του κεντρικού τετραγώνου στο κέντρο του παραθύρου
    float centerX = WINDOW_WIDTH / 2;
    float centerY = WINDOW_HEIGHT / 2;
    float initialSize = 400.0f; // Αρχικό μέγεθος του κεντρικού τετραγώνου

    // Κλήση της αναδρομικής συνάρτησης σχεδίασης fractal
    drawCubicFractal(centerX, centerY, initialSize, MAX_DEPTH, brush);
}

/**
 * @brief Κύρια συνάρτηση που αρχικοποιεί το παράθυρο και εκκινεί τον βρόχο σχεδίασης.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Cubic Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Cubic Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();

    return 0;
}
```

Επεξήγηση Κώδικα

1. **drawSquare**:
    - Σχεδιάζει ένα τετράγωνο με κέντρο στις συντεταγμένες `(x, y)` και πλευρά `side`.
2. **drawCubicFractal**:
    - Αναδρομική συνάρτηση που σχεδιάζει το cubic fractal.
    - Σχεδιάζει το κεντρικό τετράγωνο και, αν δεν έχει φτάσει στο μέγιστο βάθος, χωρίζει το τετράγωνο σε 9 μικρότερα και καλεί τη συνάρτηση για τα περιφερειακά τετράγωνα.
    - Με αυτόν τον τρόπο, η συνάρτηση αναπαράγει τη δομή του fractal επαναληπτικά, έως ότου φτάσει στο μέγιστο βάθος.
3. **draw**:
    - Ορίζει το χρώμα του τετραγώνου και καθορίζει το κέντρο και το αρχικό μέγεθος του fractal.
    - Καλεί τη συνάρτηση `drawCubicFractal` για να σχεδιάσει το fractal με προκαθορισμένο βάθος και αρχικές παραμέτρους.
4. **main**:
    - Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων για την εμφάνιση του fractal.

Αυτό το πρόγραμμα δημιουργεί ένα cubic fractal χρησιμοποιώντας ένα προκαθορισμένο μέγιστο βάθος, με αποτέλεσμα μια συμμετρική και γεωμετρικά ενδιαφέρουσα δομή.

Torus knot fractal



Το Torus Knot Fractal είναι ένα εντυπωσιακό μαθηματικό μοτίβο που βασίζεται σε μια καμπύλη που βρίσκεται στην επιφάνεια ενός τόρου (δακτυλίου). Αυτό το fractal μπορεί να παραχθεί προσεγγιστικά σε 2D χρησιμοποιώντας την επανάληψη και τη συνεχή περιστροφή ενός συνόλου σημείων κατά μήκος του τόρου.

Παρακάτω είναι ο κώδικας που δημιουργεί μια απεικόνιση του Torus Knot Fractal χρησιμοποιώντας τη βιβλιοθήκη SGG.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846
```

```
// Ορισμός παραθύρου
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;
const int MAX_ITERATIONS = 5;  // Αριθμός επαναλήψεων για τη σχεδίαση του fractal

/**
 * @brief Συνάρτηση που σχεδιάζει τον κόμπο Torus Knot σε σπειροειδές fractal μοτίβο.
 *
 * @param centerX Η x-συντεταγμένη του κέντρου.
 * @param centerY Η y-συντεταγμένη του κέντρου.
 * @param radius Η αρχική ακτίνα του κόμβου.
 * @param iterations Το βάθος αναδρομής για το fractal.
 * @param lineWidth Το πάχος της γραμμής για τη σχεδίαση.
 */
void drawTorusKnot(float centerX, float centerY, float radius, int iterations, float
lineWidth) {
    if (iterations <= 0) return;  // Βάση αναδρομής: σταματάμε όταν οι επαναλήψεις
μηδενιστούν

    // Ρύθμιση πινέλου με ελαφριά αδιαφάνεια και μεταβαλλόμενο χρώμα
    graphics::Brush brush;
    brush.fill_opacity = 0.8f;
    brush.fill_color[0] = 0.5f + 0.1f * iterations;  // Ενίσχυση κόκκινου χρώματος ανά
επανάληψη
    brush.fill_color[1] = 0.3f;
    brush.fill_color[2] = 0.8f - 0.1f * iterations;  // Μείωση μπλε απόχρωσης ανά
επανάληψη

    // Παράμετροι του κόμβου Torus Knot
    float p = 3;       // Αριθμός κυρίων καμπυλών
    float q = 2;       // Πλήθος περιστροφών γύρω από τον τόρο
    int points = 100;  // Αριθμός σημείων που σχηματίζουν τον κόμπο

    // Σχεδίαση του κόμβου Torus Knot
    for (int i = 0; i < points; ++i) {
        // Γωνία τρέχοντος σημείου και υπολογισμός συντεταγμένων (x, y)
        float angle = 2 * M_PI * i / points;
        float x = centerX + radius * cos(p * angle) * cos(angle);
        float y = centerY + radius * cos(p * angle) * sin(angle);

        // Υπολογισμός επόμενου σημείου για τη σχεδίαση γραμμής
        float nextAngle = 2 * M_PI * (i + 1) / points;
        float nextX = centerX + radius * cos(p * nextAngle) * cos(nextAngle);
        float nextY = centerY + radius * cos(p * nextAngle) * sin(nextAngle);

        // Σχεδίαση γραμμής μεταξύ των σημείων
        graphics::drawLine(x, y, nextX, nextY, brush);
    }

    // Αναδρομική κλήση για το επόμενο επίπεδο του fractal, με μικρότερη ακτίνα και
πάχος γραμμής
    drawTorusKnot(centerX, centerY, radius * 0.6f, iterations - 1, lineWidth * 0.7f);
}

/**
 * @brief Συνάρτηση σχεδίασης για το παράθυρο SGG.
 *        Ορίζει το κέντρο και την αρχική ακτίνα και καλεί τη συνάρτηση σχεδίασης του
κόμβου.
 */
void draw() {
    // Ορισμός κέντρου και αρχικής ακτίνας
    float centerX = WINDOW_WIDTH / 2;
    float centerY = WINDOW_HEIGHT / 2;
    float initialRadius = 300.0f;
```

```
    // Κλήση της συνάρτησης σχεδίασης του Torus Knot Fractal
    drawTorusKnot(centerX, centerY, initialRadius, MAX_ITERATIONS, 2.0f);
}

/**
 * @brief Κύρια συνάρτηση που δημιουργεί το παράθυρο και ξεκινά τη σχεδίαση.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Torus Knot Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Torus Knot Fractal");

    // Ορισμός της συνάρτησης σχεδίασης που θα καλείται σε κάθε καρέ
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();

    return 0;
}
```

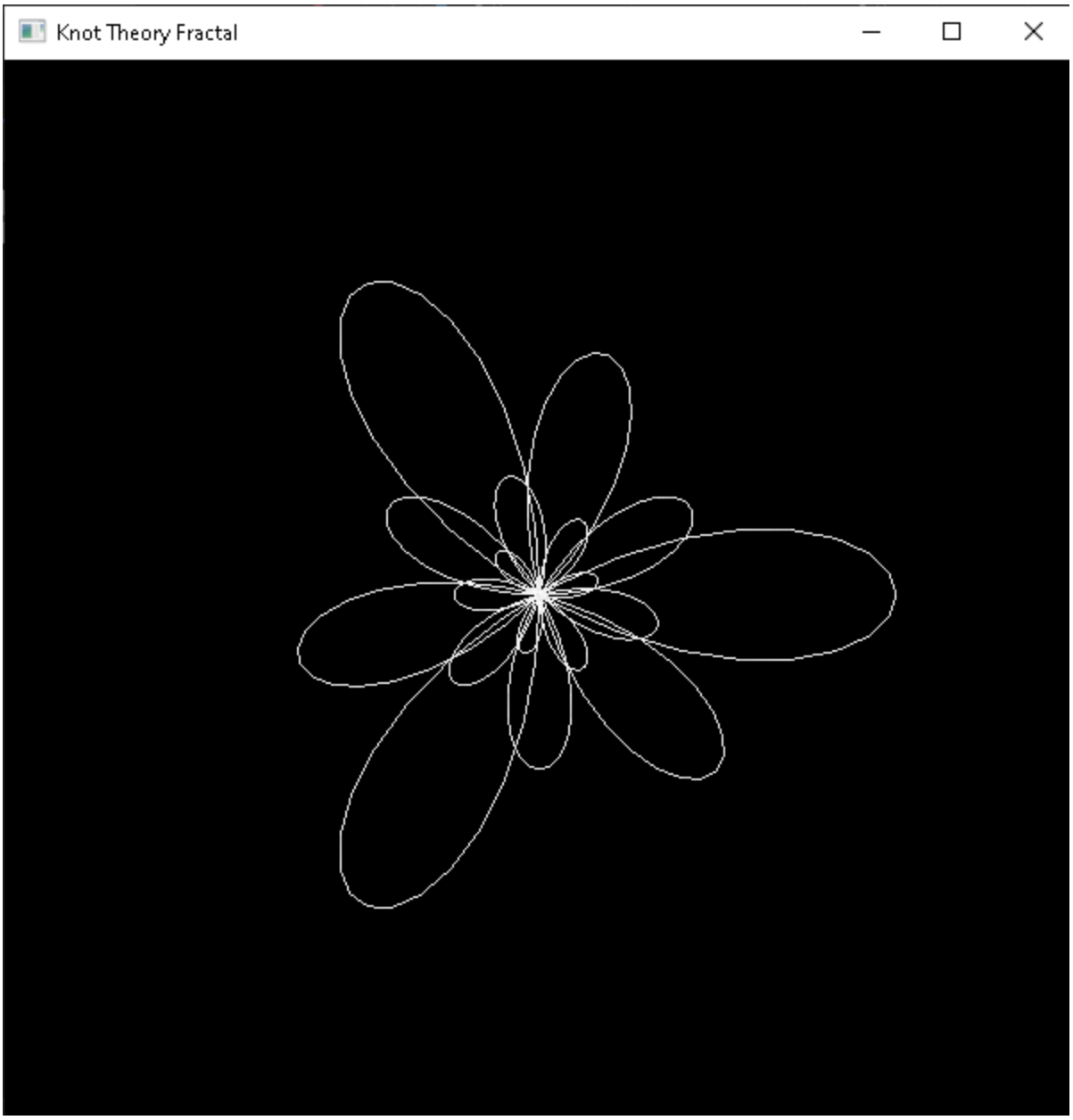
Περιγραφή και Επεξήγηση Κώδικα

1. **drawTorusKnot**:
    - Η συνάρτηση σχεδιάζει τον κόμπο Torus Knot με παραμετρική αναπαράσταση. Κάθε επανάληψη μειώνει το μέγεθος του κόμβου και τη γραμμή, δημιουργώντας ένα επαναλαμβανόμενο μοτίβο.
    - Οι μεταβλητές `p` και `q` καθορίζουν τον τύπο του κόμβου, ενώ `points` είναι ο αριθμός των σημείων για την απεικόνιση κάθε καμπύλης.
    - Η αναδρομική κλήση της `drawTorusKnot` μειώνει την ακτίνα και το πλάτος γραμμής με κάθε επανάληψη για να δημιουργήσει το fractal.
2. **draw**:
    - Ορίζει τις αρχικές παραμέτρους για το κέντρο και την ακτίνα και καλεί την `drawTorusKnot`.
3. **main**:
    - Δημιουργεί το παράθυρο, θέτει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Επεξήγηση της Λειτουργίας του Torus Knot

Αυτό το fractal αναπαράγει καμπύλες μέσα στον τόρο (δακτύλιο) και δημιουργεί έναν κόμπο με διαδοχικές στροφές. Με την αναδρομική κλήση, το fractal μειώνει το μέγεθος και προσθέτει πυκνότητα στη σχεδίαση, προσφέροντας μια αίσθηση βάθους και πολυπλοκότητας στον κόμπο.

Knot theory fractal



Ο σχεδιασμός ενός "Knot Theory Fractal" περιλαμβάνει τη χρήση επαναλαμβανόμενων και περιστρεφόμενων καμπυλών για την προσομοίωση περίπλοκων κόμβων και δεσμών που χαρακτηρίζουν τη θεωρία κόμβων. Παρακάτω είναι ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη SGG για τον σχεδιασμό ενός τέτοιου fractal.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

// Ορισμός των διαστάσεων του παραθύρου και του μέγιστου αριθμού επαναλήψεων
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;
const int MAX_ITERATIONS = 6;  // Μέγιστος αριθμός επαναλήψεων για το fractal
```

```
/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το fractal κόμβου με περιστροφή και σμίκρυνση.
 *
 * @param centerX Η x-συντεταγμένη του κέντρου.
 * @param centerY Η y-συντεταγμένη του κέντρου.
 * @param radius Η ακτίνα του αρχικού κύκλου.
 * @param iterations Το βάθος αναδρομής του fractal.
 * @param rotationAngle Η γωνία περιστροφής για κάθε επίπεδο.
 */
void drawKnotFractal(float centerX, float centerY, float radius, int iterations, float rotationAngle) {
    // Τερματίζουμε την αναδρομή όταν φτάσουμε σε μηδενικές επαναλήψεις
    if (iterations <= 0) return;

    // Ρύθμιση πινέλου και χρώματος με ελαφριά αδιαφάνεια
    graphics::Brush brush;
    brush.fill_opacity = 0.7f;
    brush.fill_color[0] = 0.3f + 0.1f * iterations;  // Χρωματική προσαρμογή για κάθε επίπεδο
    brush.fill_color[1] = 0.2f;
    brush.fill_color[2] = 0.5f + 0.1f * iterations;

    // Ορισμός των σημείων της καμπύλης
    int points = 100;

    // Σχεδίαση της καμπύλης που αποτελεί τον κόμβο
    for (int i = 0; i < points; ++i) {
        // Υπολογισμός της γωνίας του τρέχοντος σημείου και των συντεταγμένων (x, y)
        float angle = 2 * M_PI * i / points;
        float x = centerX + radius * cos(3 * angle) * cos(angle + rotationAngle);
        float y = centerY + radius * cos(3 * angle) * sin(angle + rotationAngle);

        // Υπολογισμός της επόμενης γωνίας και των επόμενων συντεταγμένων (nextX, nextY)
        float nextAngle = 2 * M_PI * (i + 1) / points;
        float nextX = centerX + radius * cos(3 * nextAngle) * cos(nextAngle + rotationAngle);
        float nextY = centerY + radius * cos(3 * nextAngle) * sin(nextAngle + rotationAngle);

        // Σχεδίαση γραμμής μεταξύ των τρεχόντων και των επόμενων σημείων
        graphics::drawLine(x, y, nextX, nextY, brush);
    }

    // Αναδρομική κλήση για το επόμενο, μικρότερο επίπεδο του fractal με περιστροφή
    drawKnotFractal(centerX, centerY, radius * 0.7f, iterations - 1, rotationAngle + M_PI / 4);
}

/**
 * @brief Συνάρτηση σχεδίασης για το παράθυρο SGG.
 *        Καλεί την `drawKnotFractal` για τη σχεδίαση του fractal κόμβου.
 */
void draw() {
    // Αρχικές συντεταγμένες για το κέντρο του fractal και η ακτίνα του
    float centerX = WINDOW_WIDTH / 2;
    float centerY = WINDOW_HEIGHT / 2;
    float initialRadius = 200.0f;

    // Κλήση της συνάρτησης για τη σχεδίαση του κόμβου
    drawKnotFractal(centerX, centerY, initialRadius, MAX_ITERATIONS, 0);
}
```

```
/**
 * @brief Κύρια συνάρτηση που δημιουργεί το παράθυρο και ξεκινά τη σχεδίαση του
fractal.
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Knot Theory Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Knot Theory Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();

    return 0;
}
```

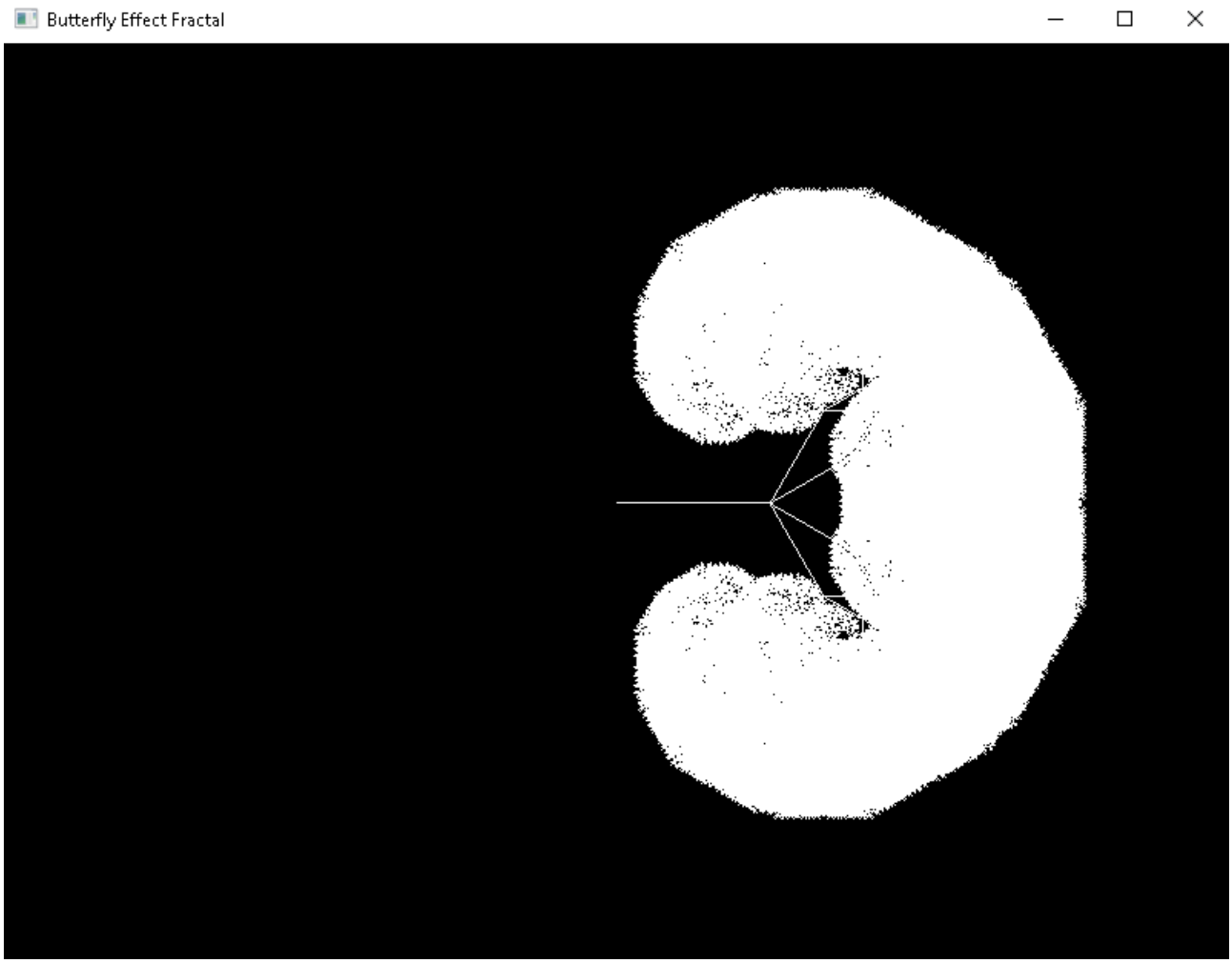
Περιγραφή και Λειτουργία Κώδικα

1. **drawKnotFractal**:
    - Σχεδιάζει την καμπύλη του fractal που αντιπροσωπεύει το Knot Theory Fractal. Η συνάρτηση χρησιμοποιεί μια παραμετρική εξίσωση για τον καθορισμό των σημείων στην καμπύλη.
    - Το χρώμα και η διαφάνεια των καμπυλών προσαρμόζονται ανάλογα με τις επαναλήψεις.
    - Κάθε επανάληψη σμικρύνει το fractal, περιστρέφοντας ελαφρώς την καμπύλη και μειώνοντας το μέγεθος για να δημιουργηθεί η εμφάνιση ενός σύνθετου κόμβου.
2. **draw**:
    - Θέτει τις αρχικές παραμέτρους και καλεί την `drawKnotFractal` με τις προκαθορισμένες παραμέτρους για τη δημιουργία του fractal.
3. **main**:
    - Δημιουργεί το παράθυρο και εκκινεί το loop σχεδίασης.

Επεξήγηση

Το `Knot Theory Fractal` προσομοιώνει τον σχηματισμό κόμβων με μια συνεχώς περιστρεφόμενη και σμικρυνόμενη καμπύλη. Με την αναδρομική προσέγγιση, το fractal δημιουργεί διαδοχικές καμπύλες που συστρέφονται, σχηματίζοντας έτσι έναν πολύπλοκο κόμβο, κάτι το οποίο αποτελεί έναν όμορφο γεωμετρικό συνδυασμό στο πεδίο των fractals και της θεωρίας κόμβων.

Butterfly Effect Fractal



Για τον σχεδιασμό του **Butterfly Effect Fractal**, μπορούμε να δημιουργήσουμε ένα μοτίβο που συμβολίζει την ευαίσθητη εξάρτηση από αρχικές συνθήκες. Μία καλή προσέγγιση είναι η χρήση μίας παραλλαγής fractal που διαχωρίζει τα μοτίβα με μικρές αποκλίσεις σε κάθε επανάληψη, κάτι που δίνει την εντύπωση του "φαινομένου της πεταλούδας". Παρακάτω, χρησιμοποιούμε μία γεωμετρική παράσταση fractal που μοιάζει με τα φτερά της πεταλούδας.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI  3.14159265358979323846

// Ορισμός διαστάσεων παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Butterfly Effect Fractal.
 *
 * @param x Η x-συντεταγμένη του αρχικού σημείου.
 * @param y Η y-συντεταγμένη του αρχικού σημείου.
```

```
 * @param length Το μήκος της γραμμής που θα σχεδιαστεί σε κάθε επίπεδο.
 * @param angle Η γωνία με την οποία θα σχεδιαστεί η γραμμή.
 * @param depth Το βάθος της αναδρομής που απομένει.
 * @param brush Το χρώμα της γραμμής.
 */
void drawButterflyEffectFractal(float x, float y, float length, float angle, int
depth, const graphics::Brush& brush) {
    if (depth == 0) return; // Βάση της αναδρομής: αν το βάθος είναι 0, σταματάμε

    // Υπολογισμός συντεταγμένων για το επόμενο σημείο της γραμμής
    float newX = x + length * cos(angle);
    float newY = y + length * sin(angle);

    // Σχεδίαση γραμμής από το τρέχον σημείο στο νέο σημείο
    graphics::drawLine(x, y, newX, newY, brush);

    // Παράμετροι για τη διακλάδωση του fractal
    float branchAngle = M_PI / 3; // Γωνία διακλάδωσης (60 μοίρες)
    float lengthReduction = 0.7f; // Μείωση μήκους γραμμής σε κάθε επίπεδο

    // Αναδρομικές κλήσεις για τα "φτερά" της πεταλούδας
    // Κλήση για το αριστερό "φτερό" με περιστροφή προς τα αριστερά
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle +
branchAngle, depth - 1, brush);

    // Κλήση για το δεξί "φτερό" με περιστροφή προς τα δεξιά
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle -
branchAngle, depth - 1, brush);

    // Πρόσθετες διακλαδώσεις για να δημιουργηθεί πιο πλούσια δομή "πεταλούδας"
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle +
branchAngle / 2, depth - 1, brush);
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle -
branchAngle / 2, depth - 1, brush);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για να ζωγραφίσει το
fractal.
 *        Καθαρίζει το φόντο και καλεί τη `drawButterflyEffectFractal`.
 */
void draw() {
    // Καθαρισμός φόντου σε μαύρο χρώμα
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ορισμός χρώματος για το fractal
    graphics::Brush fractalBrush;
    fractalBrush.fill_color[0] = 0.0f;
    fractalBrush.fill_color[1] = 0.0f;
    fractalBrush.fill_color[2] = 1.0f; // Μπλε χρώμα για το fractal

    // Εκκίνηση της σχεδίασης από το κέντρο του παραθύρου
    drawButterflyEffectFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 0, 10,
fractalBrush);
}

/**
 * @brief Κύρια συνάρτηση. Δημιουργεί το παράθυρο και ξεκινά τον βρόχο σχεδίασης.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
```

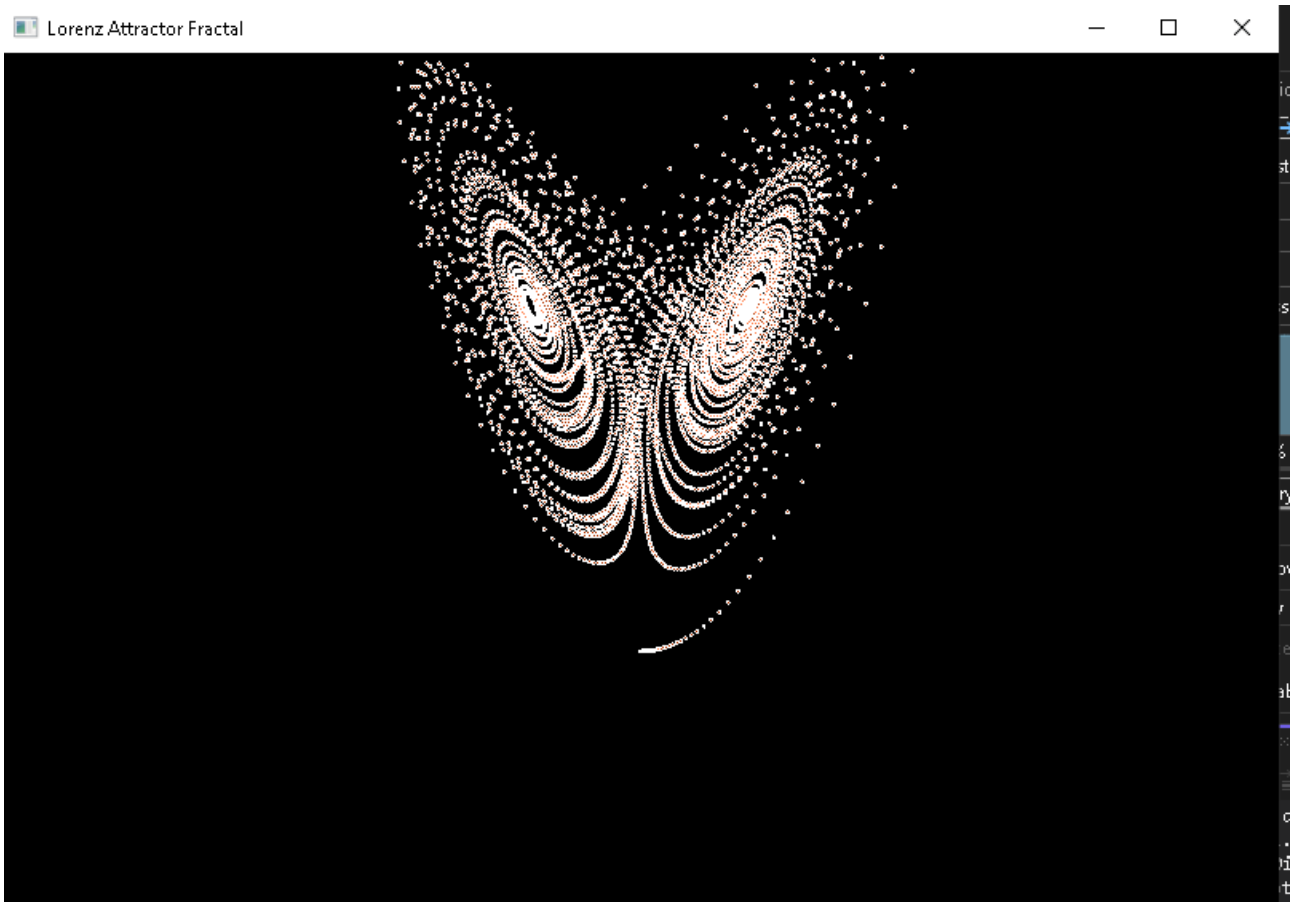
```
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Butterfly Effect Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Butterfly Effect Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης SGG
    graphics::startMessageLoop();

    return 0;
}
```

Περιγραφή Κώδικα:

1. `drawButterflyEffectFractal`: Η κύρια συνάρτηση που δημιουργεί το Butterfly Effect Fractal.
    - Ξεκινά από το σημείο (x, y) και σχεδιάζει μία γραμμή προς το σημείο `(newX, newY)`, το οποίο υπολογίζεται βάσει του μήκους και της γωνίας.
    - Για κάθε κλάδο, δημιουργεί δύο αναδρομικές κλήσεις με διαφορετικές γωνίες ώστε να δημιουργηθεί ένα μοτίβο που μοιάζει με φτερά.
    - Ορίζει μικρές αποκλίσεις γωνίας για την εντύπωση του φαινομένου της πεταλούδας.
2. `draw`: Η συνάρτηση σχεδίασης που καλείται κάθε καρέ.
    - Αρχικοποιεί τον καμβά και ξεκινά το fractal από το κέντρο του παραθύρου.
3. `main`: Δημιουργεί το παράθυρο, θέτει τη συνάρτηση σχεδίασης και εκκινεί τον κύκλο μηνυμάτων.

Αυτό το πρόγραμμα δημιουργεί ένα όμορφο "Butterfly Effect Fractal" όπου τα κλαδιά εκτείνονται με μικρές αποκλίσεις, δίνοντας την αίσθηση της ευαίσθητης εξάρτησης από τις αρχικές συνθήκες.

Lorenz Attractor Fractal



Το **Lorenz Attractor Fractal** είναι μια οπτικοποίηση του διάσημου ελκυστή Lorenz, που χρησιμοποιείται για να περιγράψει τη συμπεριφορά χαοτικών συστημάτων. Ο κώδικας που ακολουθεί προσομοιώνει το σύστημα Lorenz μέσω της επίλυσης των διαφορικών εξισώσεων του.

Κώδικας

```
#include "sgg/graphics.h"
#include <iostream>
#include <vector>

// Διαστάσεις του παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 750;

// Μέγιστος αριθμός σημείων που θα σχεδιαστούν
const int MAX_POINTS = 5000;

// Παράμετροι Lorenz Attractor
const float sigma = 10.0f;       // Παράμετρος σύζευξης των εξισώσεων
const float rho = 28.0f;         // Παράμετρος που καθορίζει το χάος στο σύστημα
const float beta = 8.0f / 3.0f;  // Παράμετρος απόσβεσης

// Αρχικές συνθήκες για το σύστημα Lorenz
float x = 0.1f, y = 0.0f, z = 0.0f; // Αρχικές συντεταγμένες
float dt = 0.01f;                   // Βήμα χρόνου για την αριθμητική ολοκλήρωση

// Δομή για την αποθήκευση των σημείων σε 2D
std::vector<std::pair<float, float>> points;
```

```
/**
 * @brief Υπολογισμός των σημείων του Lorenz Attractor.
 *
 * Αυτή η συνάρτηση υλοποιεί τις εξισώσεις Lorenz για να υπολογίσει
 * διαδοχικά σημεία του ελκυστή. Κάθε σημείο προστίθεται σε έναν πίνακα για
 * μεταγενέστερη σχεδίαση.
 */
void calculateLorenzAttractor() {
    for (int i = 0; i < MAX_POINTS; ++i) {
        // Υπολογισμός παραγώγων σύμφωνα με τις εξισώσεις Lorenz
        float dx = sigma * (y - x) * dt;             // Παραγωγός για x
        float dy = (x * (rho - z) - y) * dt;         // Παραγωγός για y
        float dz = (x * y - beta * z) * dt;          // Παραγωγός για z

        // Ενημέρωση των συντεταγμένων x, y, z
        x += dx;
        y += dy;
        z += dz;

        // Προσθήκη του σημείου στην προβολή 2D για σχεδίαση
        points.emplace_back(WINDOW_WIDTH / 2 + x * 8, WINDOW_HEIGHT / 2 - z * 8);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
 *
 * Η συνάρτηση `draw` χρησιμοποιείται για να σχεδιάσει όλα τα σημεία
 * του ελκυστή Lorenz σε μια 2D προβολή.
 */
void draw() {
    // Ορισμός του χρώματος σχεδίασης
    graphics::Brush brush;
    brush.fill_color[0] = 1.0f;  // Κόκκινο
    brush.fill_color[1] = 0.3f;  // Πράσινο
    brush.fill_color[2] = 0.0f;  // Μπλε
    brush.fill_opacity = 0.7f;   // Αδιαφάνεια

    // Σχεδίαση όλων των σημείων Lorenz Attractor
    for (auto& point : points) {
        graphics::drawDisk(point.first, point.second, 1.0f, brush); // Κάθε σημείο
σχεδιάζεται ως μικρός κύκλος
    }
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *
 * Η `main` υλοποιεί τη βασική λειτουργία του προγράμματος, η οποία περιλαμβάνει
 * τον υπολογισμό των σημείων και τη δημιουργία του παραθύρου γραφικών.
 *
 * @return int Επιστρέφει το 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου γραφικών με τον τίτλο "Lorenz Attractor Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Lorenz Attractor Fractal");

    calculateLorenzAttractor(); // Υπολογισμός των σημείων πριν από τη σχεδίαση

    graphics::setDrawFunction(draw); // Ορισμός της συνάρτησης σχεδίασης

    // Έναρξη του βρόχου μηνυμάτων για συνεχή σχεδίαση και ανανέωση
    graphics::startMessageLoop();
```

```
    return 0;
}
```

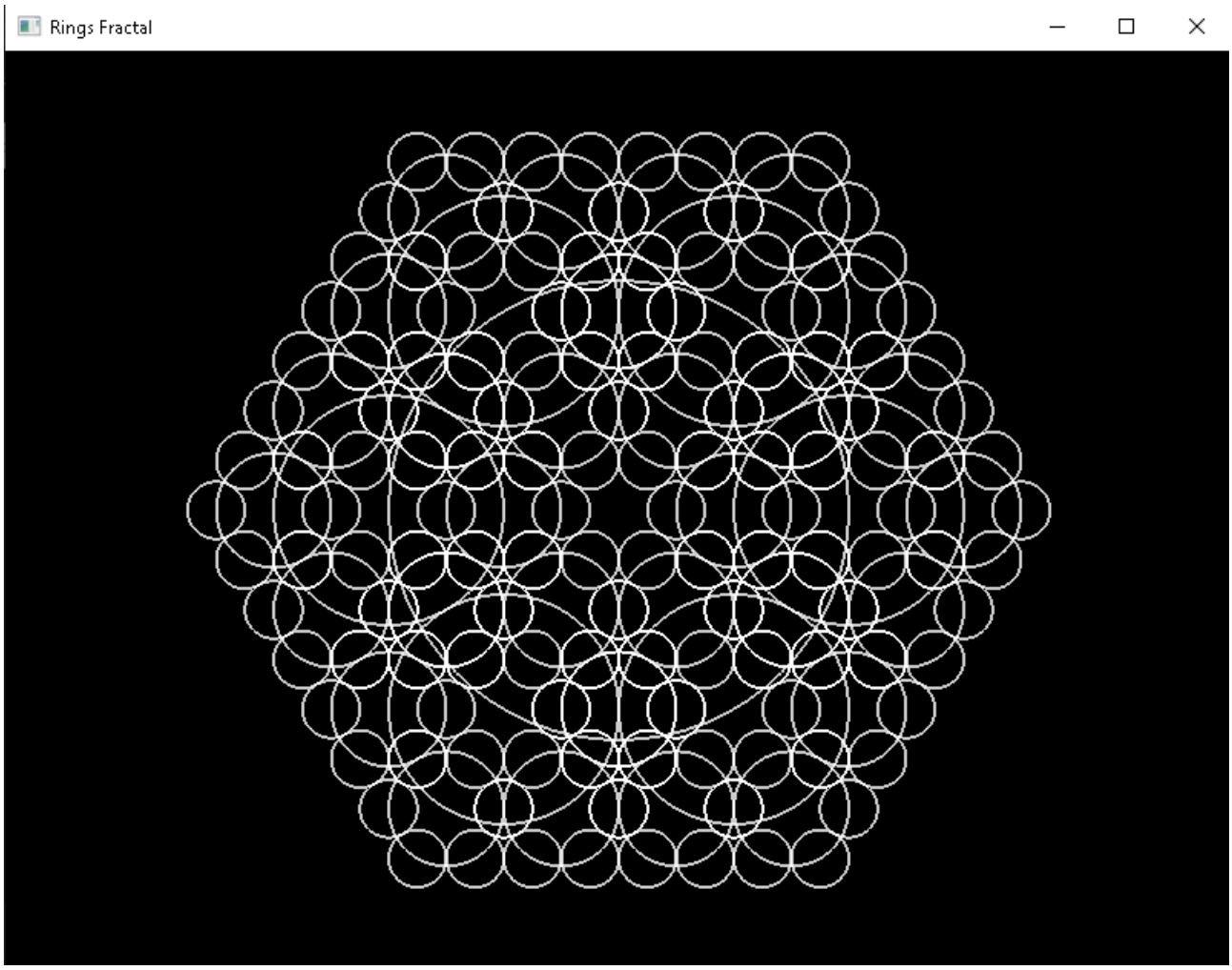
Επεξήγηση Κώδικα

1. **Παράμετροι Lorenz Attractor**:
    - Οι παράμετροι `sigma`, `rho`, και `beta` καθορίζουν τη συμπεριφορά του ελκυστή. Αυτές είναι σταθερές που επηρεάζουν τη ροή και το χάος του συστήματος.
2. **Συνάρτηση** `calculateLorenzAttractor`:
    - Υπολογίζει τα σημεία του ελκυστή χρησιμοποιώντας τις διαφορικές εξισώσεις Lorenz.
    - Κάθε σημείο προκύπτει από τις μεταβολές `dx`, `dy`, `dz`, που υπολογίζονται με βάση τις τιμές των `x`, `y`, `z`.
    - Τα σημεία αποθηκεύονται σε ένα διάνυσμα, `points`, για να μπορούν να προβληθούν σε 2D.
3. **Συνάρτηση** `draw`:
    - Χρησιμοποιεί τα αποθηκευμένα σημεία στο `points` για να σχεδιάσει τη διαδρομή του Lorenz Attractor στην οθόνη.
    - Καθένα από αυτά τα σημεία σχεδιάζεται με ένα μικρό κύκλο.
4. **Κύρια Συνάρτηση**:
    - Ανοίγει ένα παράθυρο και ξεκινά τον κύκλο μηνυμάτων.
    - Καλεί τη `calculateLorenzAttractor` για τον υπολογισμό των σημείων πριν από τη σχεδίαση, και αναθέτει τη `draw` για να τα εμφανίσει στο παράθυρο.

Περιγραφή

Το **Lorenz Attractor** αποτυπώνει την πορεία ενός χαοτικού συστήματος και δημιουργεί εντυπωσιακές διαδρομές, οι οποίες προσομοιάζουν το "πετάρισμα" της πεταλούδας – αναδεικνύοντας τη θεωρία του χάους.

Ring Fractal



Ο σχεδιασμός του **Rings Fractal** περιλαμβάνει τη σχεδίαση ενός κύκλου που περιβάλλεται από μικρότερους κύκλους, οι οποίοι επαναλαμβάνονται με τον ίδιο τρόπο σε κάθε κύκλο. Το αποτέλεσμα είναι ένα εντυπωσιακό μοτίβο από ομόκεντρους κύκλους.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

#define M_PI 3.14159265358979323846

// Διαστάσεις παραθύρου γραφικών
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός fractal δακτυλίων.
 *
 * Σχεδιάζει έναν κύκλο και περιμετρικά του άλλους μικρότερους κύκλους.
 * Κάθε μικρότερος κύκλος αναπαράγει το ίδιο μοτίβο με μειωμένη ακτίνα.
 *
 * @param x Συντεταγμένη X του κεντρικού κύκλου
 * @param y Συντεταγμένη Y του κεντρικού κύκλου
 * @param radius Ακτίνα του κύκλου
```

```
 * @param depth Βάθος αναδρομής για το fractal
 */
void drawRingsFractal(float x, float y, float radius, int depth) {
    if (depth <= 0) return;  // Βάση αναδρομής: σταματάμε αν το βάθος είναι 0

    // Ορισμός των ιδιοτήτων σχεδίασης του κύκλου
    graphics::Brush brush;
    brush.fill_opacity = 0.0f;        // Διαφανής πλήρωση
    brush.outline_opacity = 0.8f;     // Ημιδιαφανές περίγραμμα
    brush.outline_width = 2.0f;       // Πλάτος περιγράμματος

    // Σχεδίαση του κεντρικού κύκλου
    graphics::drawDisk(x, y, radius, brush);

    // Αριθμός μικρότερων κύκλων γύρω από τον κεντρικό
    int numCircles = 6;

    // Υπολογισμός γωνίας και τοποθέτηση μικρότερων κύκλων περιμετρικά του κεντρικού
    for (int i = 0; i < numCircles; i++) {
        float angle = (2 * M_PI / numCircles) * i;     // Γωνία τοποθέτησης σε ακτίνια
        float newX = x + radius * cos(angle);          // Νέα θέση X του μικρότερου
κύκλου
        float newY = y + radius * sin(angle);          // Νέα θέση Y του μικρότερου
κύκλου

        // Αναδρομική κλήση για τη σχεδίαση του μικρότερου κύκλου με μειωμένη ακτίνα
και βάθος
        drawRingsFractal(newX, newY, radius / 2, depth - 1);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών SGG.
 *
 * Η `draw` χρησιμοποιείται για τη σχεδίαση του κεντρικού φράκταλ στο κέντρο του
παραθύρου.
 */
void draw() {
    // Ορισμός του χρώματος φόντου
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;   // Μαύρο χρώμα
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης για τη σχεδίαση του fractal των δακτυλίων
    drawRingsFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, 4);  // Αρχική ακτίνα
150 και βάθος 4
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *
 * Η συνάρτηση `main` αρχικοποιεί το παράθυρο και ορίζει τη συνάρτηση σχεδίασης.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου γραφικών με τον τίτλο "Rings Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Rings Fractal");

    // Καθορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);
```

```
    // Έναρξη του βρόχου μηνυμάτων της βιβλιοθήκης
    graphics::startMessageLoop();
    return 0;
}
```

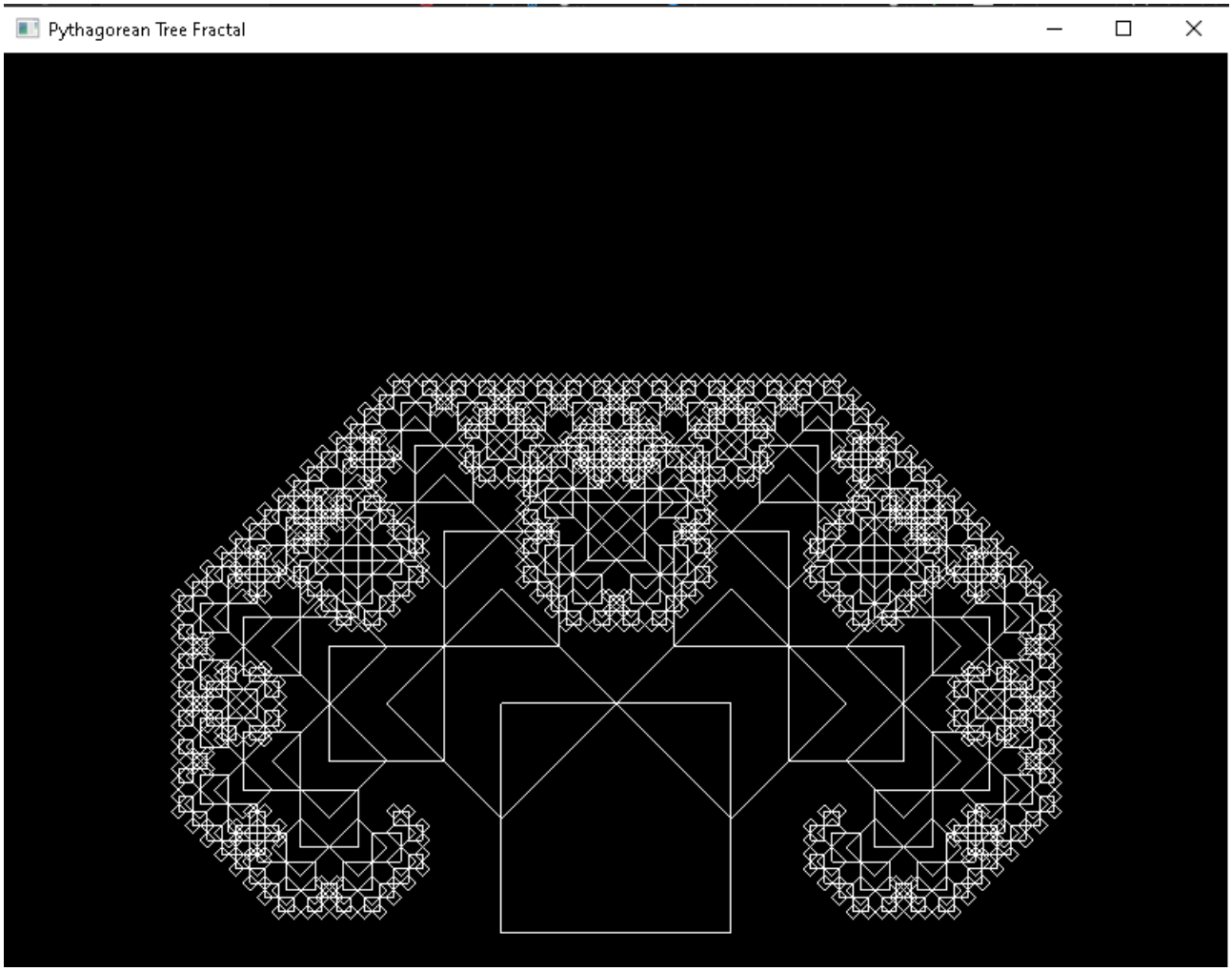
Επεξήγηση Κώδικα

1. **Συνάρτηση** `drawRingsFractal`:
    - Αυτή η συνάρτηση σχεδιάζει το fractal των δακτυλίων. Ξεκινά σχεδιάζοντας έναν κεντρικό κύκλο με την καθορισμένη ακτίνα και στη συνέχεια δημιουργεί μικρότερους κύκλους περιμετρικά σε γωνίες που τους τοποθετούν ομοιόμορφα γύρω από τον κύριο κύκλο.
    - Η συνάρτηση καλείται αναδρομικά για κάθε μικρότερο κύκλο, με το `depth` να μειώνεται κάθε φορά, ορίζοντας το επίπεδο αναδρομής και ελέγχοντας πότε να σταματήσει.
2. **Συνάρτηση** `draw`:
    - Καθαρίζει το φόντο με λευκό και στη συνέχεια καλεί τη `drawRingsFractal` για να δημιουργήσει το fractal στο κέντρο του παραθύρου.
3. **Κύρια Συνάρτηση**:
    - Δημιουργεί το παράθυρο, ορίζει τη `draw` ως συνάρτηση σχεδίασης, και ξεκινά τον κύκλο μηνυμάτων για την απόδοση των δακτυλίων fractal.

Περιγραφή

Αυτός ο κώδικας δημιουργεί ένα fractal από ομόκεντρους δακτυλίους, με κάθε κύκλος να περιβάλλεται από μικρότερους κύκλους σε κάθε επίπεδο αναδρομής, δημιουργώντας ένα εντυπωσιακό και συμμετρικό μοτίβο δακτυλίων.

Pythagorean Tree Fractal



Το **Pythagorean Fractal** βασίζεται στη δημιουργία ενός δέντρου που αναπτύσσεται σε σχήμα Πυθαγόρειου δέντρου. Το fractal αυτό σχηματίζεται επαναλαμβάνοντας την ίδια διαδικασία δημιουργίας μικρότερων τετραγώνων και τριγώνων στις γωνίες του βασικού τετραγώνου.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

#define M_PI 3.14159265358979323846

// Διαστάσεις παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;
const float INITIAL_SIDE = 150.0f; // Πλευρά του αρχικού τετραγώνου

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το φράκταλ του Πυθαγορείου Δέντρου.
 *
 * Σε κάθε επίπεδο της αναδρομής, δύο μικρότερα τετράγωνα προστίθενται
 * στις πάνω γωνίες του τρέχοντος τετραγώνου, δημιουργώντας τη δομή του φράκταλ.
 *
 * @param x Συντεταγμένη x του κέντρου του τετραγώνου
 * @param y Συντεταγμένη y του κέντρου του τετραγώνου
```

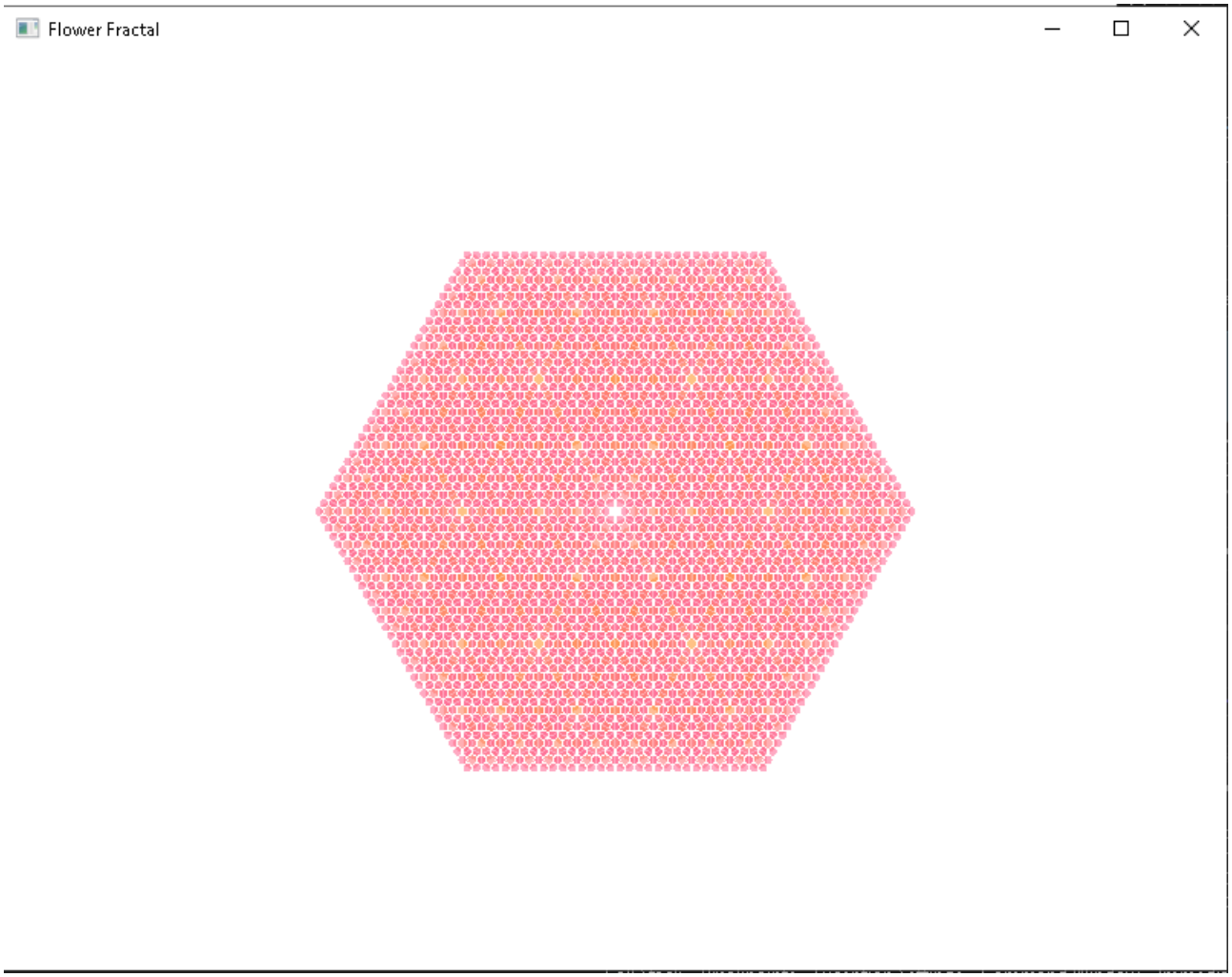
```
 * @param side Η πλευρά του τετραγώνου
 * @param angle Γωνία περιστροφής του τετραγώνου
 * @param depth Το βάθος της αναδρομής
 */
void drawPythagoreanTree(float x, float y, float side, float angle, int depth) {
    if (depth <= 0) return; // Βάση αναδρομής

    // Δημιουργία πινέλου για τη σχεδίαση των τετραγώνων
    graphics::Brush brush;
    brush.fill_opacity = 1.0f;
    brush.fill_color[0] = 0.0f; // Πράσινη απόχρωση
    brush.fill_color[1] = 0.5f + 0.5f * (float)depth / 10;
    brush.fill_color[2] = 0.0f;

    // Υπολογισμός των κορυφών του τετραγώνου με βάση το κέντρο και τη γωνία
περιστροφής
    float halfSide = side / 2.0f;
    float rad = angle * M_PI / 180.0f;

    float x0 = x - halfSide * cos(rad) - halfSide * sin(rad);
    float y0 = y - halfSide * sin(rad) + halfSide * cos(rad);

    float x1 = x + halfSide * cos(rad) - halfSide * sin(rad);
    float y1 = y + halfSide * sin(rad) + halfSide * cos(rad);

    float x2 = x + halfSide * cos(rad) + halfSide * sin(rad);
    float y2 = y + halfSide * sin(rad) - halfSide * cos(rad);

    float x3 = x - halfSide * cos(rad) + halfSide * sin(rad);
    float y3 = y - halfSide * sin(rad) - halfSide * cos(rad);

    // Σχεδίαση των γραμμών που σχηματίζουν το τετράγωνο
    graphics::drawLine(x0, y0, x1, y1, brush);
    graphics::drawLine(x1, y1, x2, y2, brush);
    graphics::drawLine(x2, y2, x3, y3, brush);
    graphics::drawLine(x3, y3, x0, y0, brush);

    // Υπολογισμός νέου μεγέθους πλευράς για τα υπο-τετράγωνα και νέες γωνίες
    float newSide = side * 0.707; // Μείωση μεγέθους κατά √2 / 2
    float leftAngle = angle - 45;
    float rightAngle = angle + 45;

    // Συντεταγμένες για τα κέντρα των νέων τετραγώνων
    float leftX = x3;
    float leftY = y3;
    float rightX = x2;
    float rightY = y2;

    // Αναδρομική κλήση για τη σχεδίαση του αριστερού και του δεξιού τετραγώνου
    drawPythagoreanTree(leftX, leftY, newSide, leftAngle, depth - 1);
    drawPythagoreanTree(rightX, rightY, newSide, rightAngle, depth - 1);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
 *
 * Η συνάρτηση `draw` σχεδιάζει το φόντο και καλεί τη συνάρτηση `drawPythagoreanTree`
 * για να ξεκινήσει η αναδρομική σχεδίαση του Πυθαγόρειου Δέντρου από το κάτω κέντρο.
 */
void draw() {
    // Ορισμός φόντου
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Μαύρο φόντο
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
```

```
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης fractal από το κάτω μέρος του παραθύρου
    drawPythagoreanTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 100, INITIAL_SIDE, 0, 10);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *
 * Δημιουργεί το παράθυρο και ρυθμίζει τη συνάρτηση σχεδίασης.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Pythagorean Tree Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Pythagorean Tree Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή

1. `drawPythagoreanTree`: Αυτή η συνάρτηση σχεδιάζει το Πυθαγόρειο δέντρο αναδρομικά. Τοποθετεί ένα τετράγωνο σε συγκεκριμένη γωνία και μετά καλεί την ίδια συνάρτηση για τα αριστερά και δεξιά υποτετράγωνα σε νέες θέσεις και γωνίες.
2. `draw`: Αρχικοποιεί τον καμβά και καλεί την `drawPythagoreanTree` από το κέντρο του παραθύρου, θέτοντας τη βάση του αρχικού τετραγώνου κοντά στο κάτω μέρος της οθόνης.
3. `main`: Δημιουργεί το παράθυρο και ξεκινά τον κύκλο μηνυμάτων της βιβλιοθήκης γραφικών SGG.

Flower Fractal



Το **Flower Fractal** είναι ένα επαναληπτικό μοτίβο που σχηματίζει πέταλα γύρω από ένα κεντρικό σημείο. Κάθε νέο πέταλο γεννά μικρότερα πέταλα, δίνοντας την αίσθηση ενός λουλουδιού που αναπτύσσεται σε διαφορετικά επίπεδα.

Ακολουθεί ο κώδικας για τον σχεδιασμό του Flower Fractal σε C++ με την βιβλιοθήκη SGG:

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

#define M_PI 3.14159265358979323846

// Διαστάσεις παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το φράκταλ του λουλουδιού.
 *
 * Η συνάρτηση σχεδιάζει έναν κύκλο σε κάθε πέταλο και κατόπιν αναδρομικά
 * καλεί τον εαυτό της για κάθε νέο πέταλο, σχηματίζοντας μικρότερα πέταλα.
 *
```

```
 * @param x Η συντεταγμένη x του κέντρου του κύριου λουλουδιού
 * @param y Η συντεταγμένη y του κέντρου του κύριου λουλουδιού
 * @param radius Η ακτίνα του κύκλου (πέταλου)
 * @param depth Το βάθος της αναδρομής, καθορίζει το επίπεδο λεπτομέρειας
 * @param petals Ο αριθμός των πετάλων για κάθε επίπεδο αναδρομής
 */
void drawFlowerFractal(float x, float y, float radius, int depth, int petals) {
    if (depth <= 0) return; // Βάση αναδρομής

    // Ορισμός πινέλου για τα πέταλα
    graphics::Brush petalBrush;
    petalBrush.fill_opacity = 0.6f;  // Διαφάνεια πετάλων
    petalBrush.fill_color[0] = 1.0f; // Χρώμα πετάλου (κόκκινο-ροζ απόχρωση που
αλλάζει με το βάθος)
    petalBrush.fill_color[1] = 0.4f + 0.2f * depth / 5;
    petalBrush.fill_color[2] = 0.7f - 0.1f * depth;

    // Υπολογισμός γωνίας για κάθε πέταλο ώστε να κατανέμονται ομοιόμορφα γύρω από το
κέντρο
    float angleIncrement = 360.0f / petals;
    for (int i = 0; i < petals; i++) {
        float angle = i * angleIncrement * M_PI / 180.0f;

        // Υπολογισμός θέσης του πετάλου με βάση την ακτίνα και τη γωνία
        float petalX = x + radius * cos(angle);
        float petalY = y + radius * sin(angle);

        // Σχεδίαση του πετάλου ως δίσκος
        graphics::drawDisk(petalX, petalY, radius * 0.5f, petalBrush);

        // Αναδρομική κλήση για το κάθε πέταλο, μειώνοντας το μέγεθος και το βάθος
        drawFlowerFractal(petalX, petalY, radius * 0.5f, depth - 1, petals);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
 *
 * Η συνάρτηση καθαρίζει το παράθυρο με λευκό φόντο και καλεί τη συνάρτηση
 * `drawFlowerFractal` για να σχεδιάσει το φράκταλ του λουλουδιού.
 */
void draw() {
    // Καθαρισμός του παραθύρου με λευκό φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 1.0f; // Λευκό
    bg.fill_color[1] = 1.0f;
    bg.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης σχεδίασης του Flower Fractal
    drawFlowerFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 5, 6);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *
 * Δημιουργεί το παράθυρο και ορίζει τη συνάρτηση σχεδίασης για την απεικόνιση του
φράκταλ.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Flower Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Flower Fractal");
```

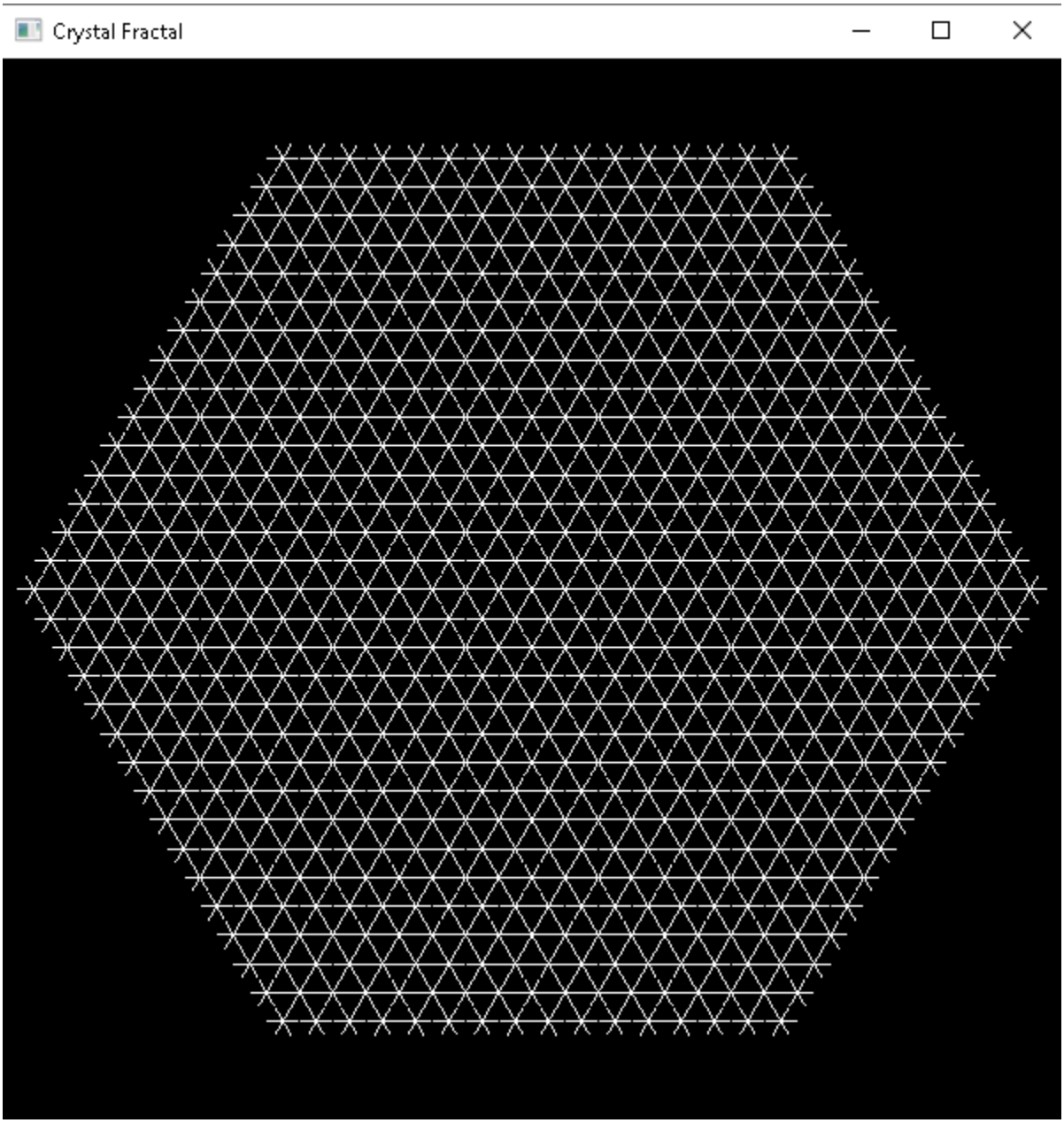
```
    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή του Κώδικα

1. `drawFlowerFractal`: Η συνάρτηση σχεδιάζει τα πέταλα του λουλουδιού σε κάθε επίπεδο βάθους. Για κάθε νέο πέταλο, καλεί την ίδια συνάρτηση αναδρομικά με μικρότερη ακτίνα και χαμηλότερο βάθος, δημιουργώντας έτσι ένα fractal.
2. `draw`: Καθαρίζει το παράθυρο σχεδίασης και καλεί τη συνάρτηση `drawFlowerFractal` από το κέντρο του παραθύρου, με αρχική ακτίνα και προκαθορισμένο αριθμό πετάλων.
3. `main`: Ορίζει το παράθυρο της εφαρμογής και τη βασική συνάρτηση σχεδίασης, ξεκινώντας το κύριο loop μηνυμάτων της βιβλιοθήκης γραφικών.

Με αυτόν τον κώδικα, δημιουργείται ένα εντυπωσιακό fractal λουλούδι με αναδρομικές κλήσεις για να επιτευχθεί η fractal συμμετρία των πετάλων.

Crystal Fractal



Το **Crystal Fractal** είναι ένα όμορφο μοτίβο fractal που σχηματίζει συμμετρικές, ακτινωτές δομές, θυμίζοντας πολύτιμο κρύσταλλο. Για να το σχεδιάσουμε, δημιουργούμε ένα κεντρικό σημείο και "ακτίνες" ή "πέταλα" που εκτείνονται από αυτό, και κάθε ακτίνα γεννά νέες μικρότερες ακτίνες.

Ακολουθεί ο κώδικας για τον σχεδιασμό του Crystal Fractal σε C++ χρησιμοποιώντας τη βιβλιοθήκη SGG:

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>

#define M_PI 3.14159265358979323846

// Διαστάσεις παραθύρου
const float WINDOW_WIDTH = 600;
```

```
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το Crystal Fractal
 *
 * Η συνάρτηση δημιουργεί ένα ακτινικό fractal όπου κάθε κλάδος δημιουργεί
 * μικρότερους κλάδους σε προκαθορισμένες γωνίες, σχηματίζοντας έτσι έναν
 * συμμετρικό σχηματισμό.
 *
 * @param x Η συντεταγμένη x του κέντρου του fractal
 * @param y Η συντεταγμένη y του κέντρου του fractal
 * @param length Το μήκος κάθε κλάδου του fractal
 * @param depth Το βάθος της αναδρομής, καθορίζει το επίπεδο λεπτομέρειας
 * @param branches Ο αριθμός των κλάδων που εκτείνονται από κάθε σημείο
 */
void drawCrystalFractal(float x, float y, float length, int depth, int branches) {
    if (depth <= 0) return;  // Βάση αναδρομής

    // Ορισμός πινέλου για την σχεδίαση του κλάδου
    graphics::Brush crystalBrush;
    crystalBrush.fill_color[0] = 0.5f + 0.5f * (depth % 2);  // Εναλλαγή χρώματος ανά
επίπεδο
    crystalBrush.fill_color[1] = 0.8f - 0.2f * depth / 5;
    crystalBrush.fill_color[2] = 1.0f - 0.1f * depth;

    // Υπολογισμός γωνίας ανάμεσα στους κλάδους ώστε να τοποθετηθούν ομοιόμορφα γύρω
από το κέντρο
    float angleIncrement = 360.0f / branches;

    // Σχεδίαση και αναδρομική κλήση για κάθε κλάδο
    for (int i = 0; i < branches; i++) {
        // Υπολογισμός γωνίας και νέων συντεταγμένων για κάθε κλάδο
        float angle = i * angleIncrement * M_PI / 180.0f;
        float newX = x + length * cos(angle);
        float newY = y + length * sin(angle);

        // Σχεδίαση του κλάδου ως γραμμή από το σημείο (x, y) στο (newX, newY)
        graphics::drawLine(x, y, newX, newY, crystalBrush);

        // Αναδρομική κλήση για την δημιουργία μικρότερων κλάδων στο τέλος κάθε
γραμμής
        drawCrystalFractal(newX, newY, length * 0.5f, depth - 1, branches);
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών
 *
 * Η συνάρτηση καθαρίζει το παράθυρο με μαύρο φόντο και καλεί τη συνάρτηση
 * `drawCrystalFractal` για να ξεκινήσει η σχεδίαση του fractal από το κέντρο του
παραθύρου.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης σχεδίασης του Crystal Fractal
    drawCrystalFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, 5, 6);
}
```

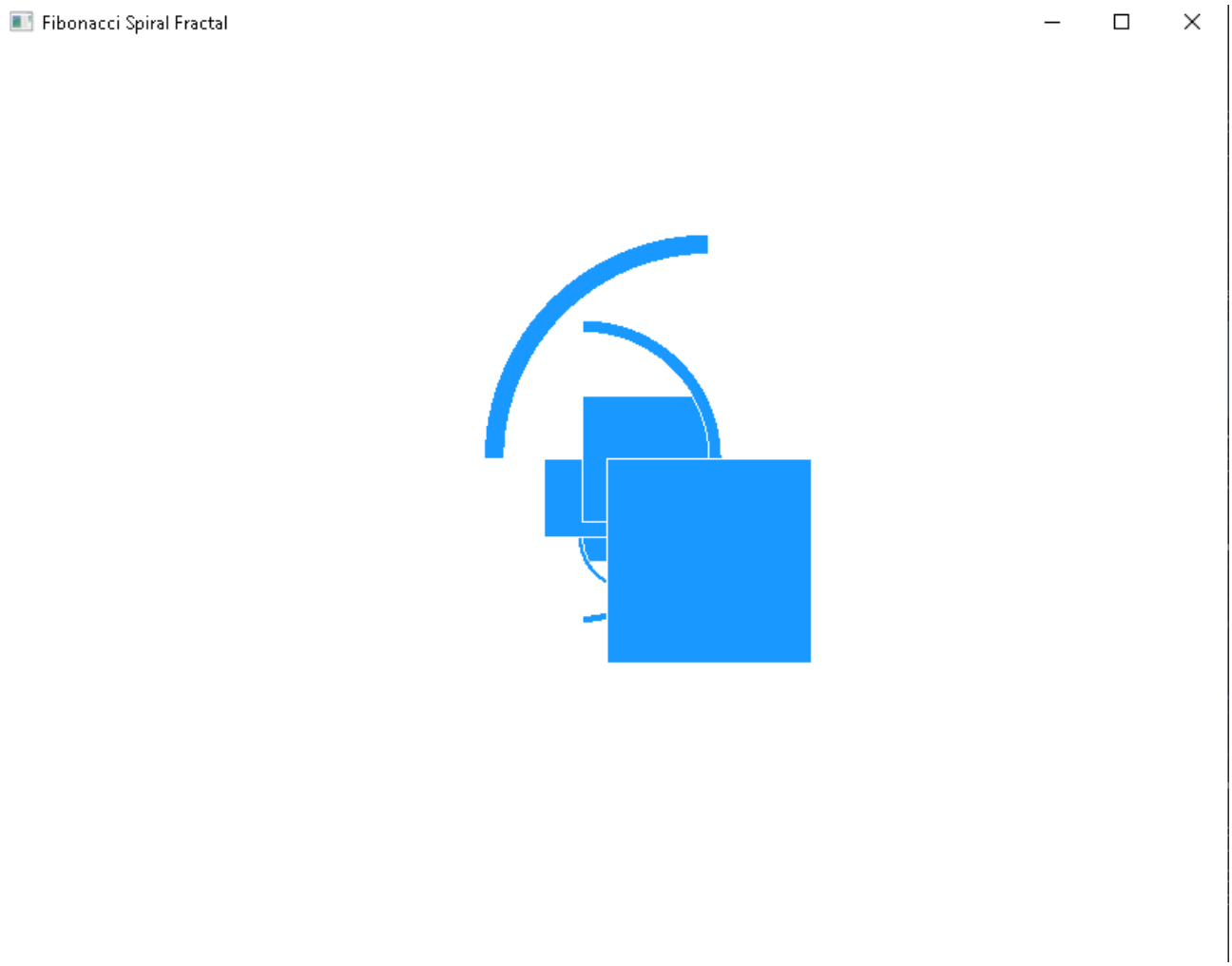
```
/**
 * @brief Κύρια συνάρτηση του προγράμματος
 *
 * Δημιουργεί το παράθυρο και ορίζει τη συνάρτηση σχεδίασης για την απεικόνιση του
Crystal Fractal.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    // Δημιουργία παραθύρου με τίτλο "Crystal Fractal"
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Crystal Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή του Κώδικα

1. `drawCrystalFractal`: Αυτή η συνάρτηση σχεδιάζει το fractal. Ξεκινά από ένα σημείο και σχεδιάζει κλάδους σε διάφορες κατευθύνσεις. Κάθε κλάδος σχεδιάζει με αναδρομή νέους κλάδους από το τέλος του, μικρότερους σε μήκος, μειώνοντας το βάθος κάθε φορά, για να δώσει τη fractal συμμετρία.
2. `draw`: Καθαρίζει την οθόνη και καλεί τη συνάρτηση `drawCrystalFractal` από το κέντρο της οθόνης. Ορίζει αρχική τιμή για το μήκος του πρώτου κλάδου και τον αριθμό των επαναλήψεων (βάθος).
3. `main`: Ορίζει τις βασικές παραμέτρους παραθύρου και εισάγει τη συνάρτηση σχεδίασης του fractal.

Αυτό το πρόγραμμα παράγει ένα εντυπωσιακό Crystal Fractal με διακλαδώσεις που επαναλαμβάνονται αναδρομικά και δίνουν την αίσθηση κρυστάλλινων ακτίνων ή δομών.

Fibonacci Spiral Fractal



Το **Fibonacci Spiral Fractal** βασίζεται στη διάσημη ακολουθία Fibonacci και δημιουργεί μια σπείρα με ορθογώνια και κυκλικά τόξα που ακολουθούν το "χρυσό λόγο". Κάθε τετράγωνο έχει μήκος πλευράς ίσο με έναν αριθμό της ακολουθίας Fibonacci και τοποθετείται δίπλα στο προηγούμενο, σχηματίζοντας τη σπείρα. Αυτός ο fractal είναι δημοφιλής σε πολλούς τομείς της τέχνης και των φυσικών επιστημών λόγω της αναλογίας του.

Ακολουθεί ο κώδικας για τον σχεδιασμό του Fibonacci Spiral Fractal σε C++ χρησιμοποιώντας τη βιβλιοθήκη SGG:

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#include <vector>
#define M_PI 3.14159265358979323846

// Ορισμός διαστάσεων παραθύρου και χρυσής αναλογίας
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;
const float GOLDEN_RATIO = 1.618f;

/**
 * @brief Υπολογισμός της ακολουθίας Fibonacci
```

```
 *
 * Η συνάρτηση αυτή δημιουργεί έναν πίνακα με τιμές Fibonacci, οι οποίες χρησιμοποιούνται
 * ως διαδοχικά μήκη πλευράς για τη σχεδίαση των τετραγώνων και τόξων του fractal.
 *
 * @param terms Ο αριθμός των όρων της ακολουθίας Fibonacci που θα υπολογιστούν.
 * @param scaleFactor Συντελεστής κλίμακας για το μήκος των πλευρών.
 * @return Ένας πίνακας με τις τιμές Fibonacci προσαρμοσμένες στον συντελεστή κλίμακας.
 */
std::vector<float> generateFibonacci(int terms, float scaleFactor = 1.0f) {
    std::vector<float> fibonacci = { scaleFactor, scaleFactor * GOLDEN_RATIO };
    for (int i = 2; i < terms; i++) {
        fibonacci.push_back(fibonacci[i - 1] + fibonacci[i - 2]);
    }
    return fibonacci;
}

/**
 * @brief Αναδρομική συνάρτηση για την σχεδίαση της σπείρας Fibonacci.
 *
 * Η συνάρτηση χρησιμοποιεί την ακολουθία Fibonacci για να σχεδιάσει διαδοχικά τετράγωνα
 * και τόξα, τα οποία διατάσσονται σε μια σπείρα ακολουθώντας την αναλογία του χρυσού αριθμού.
 *
 * @param x Η αρχική συντεταγμένη x της σπείρας.
 * @param y Η αρχική συντεταγμένη y της σπείρας.
 * @param terms Ο αριθμός των όρων της ακολουθίας Fibonacci που θα χρησιμοποιηθούν.
 * @param initialLength Το αρχικό μήκος πλευράς για το πρώτο τετράγωνο.
 */
void drawFibonacciSpiral(float x, float y, int terms, float initialLength) {
    // Δημιουργία της ακολουθίας Fibonacci
    std::vector<float> fibonacci = generateFibonacci(terms, initialLength);

    float angle = 0;  // Αρχική γωνία
    graphics::Brush spiralBrush;
    spiralBrush.fill_color[0] = 0.1f;
    spiralBrush.fill_color[1] = 0.6f;
    spiralBrush.fill_color[2] = 1.0f;

    // Σχεδίαση της σπείρας
    for (int i = 0; i < terms; i++) {
        float length = fibonacci[i];  // Μήκος πλευράς για κάθε όρο
        float nextX = x + length * cos(angle * M_PI / 180.0f);  // Νέα θέση x
        float nextY = y + length * sin(angle * M_PI / 180.0f);  // Νέα θέση y

        // Σχεδίαση τετραγώνου Fibonacci
        graphics::drawRect(
            x + length / 2 * cos(angle * M_PI / 180.0f),
            y + length / 2 * sin(angle * M_PI / 180.0f),
            length, length, spiralBrush
        );

        // Σχεδίαση τόξου με γωνία 90° για κάθε όρο της ακολουθίας
        graphics::drawSector(x, y, length, length * 1.1f, angle, angle + 90, spiralBrush);

        // Ενημέρωση συντεταγμένων και γωνίας για το επόμενο τετράγωνο
        x = nextX;
        y = nextY;
        angle += 90;  // Περιστροφή 90° για επόμενη σπείρα
    }
}
```

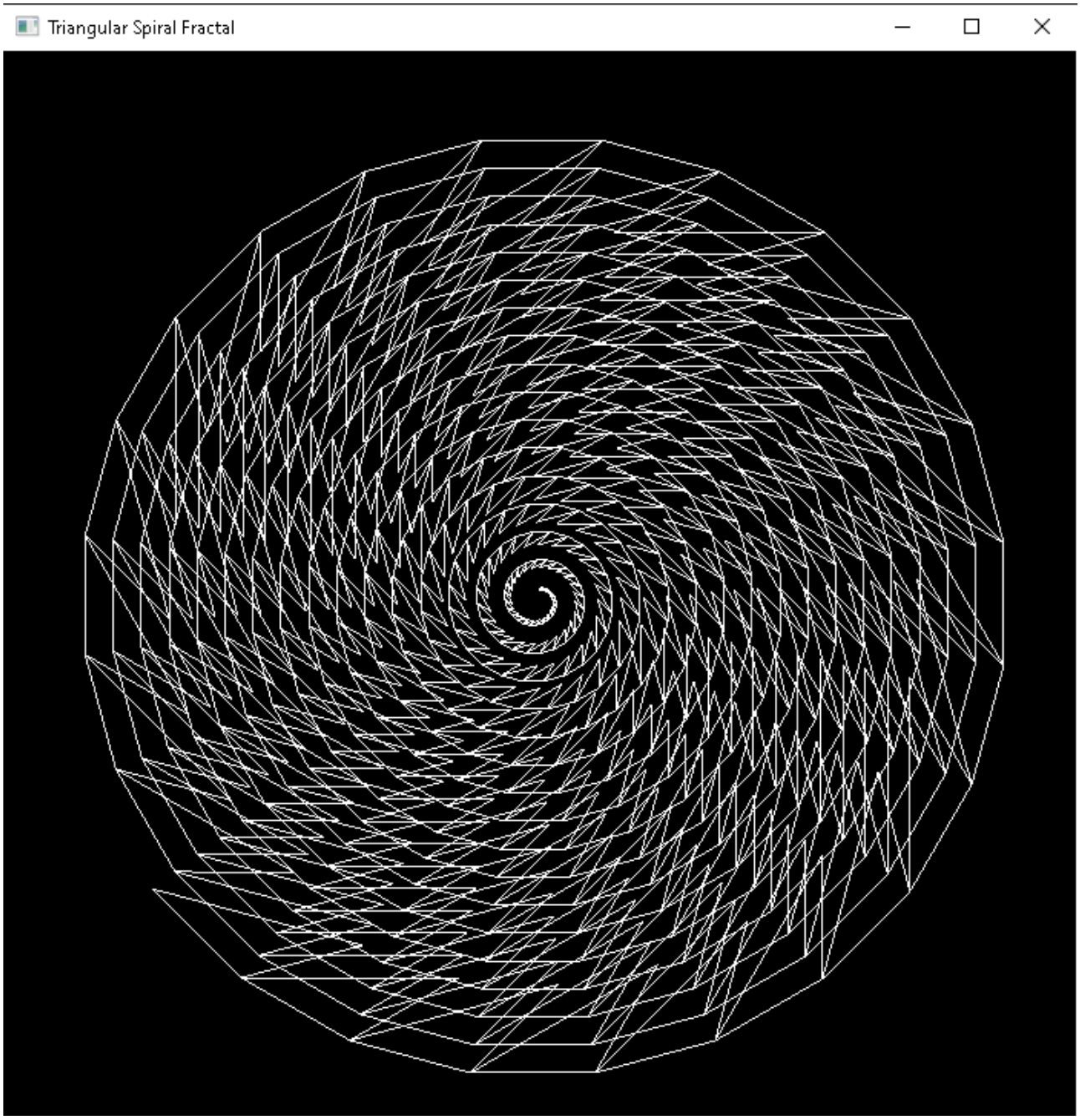
```
/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
 *
 * Η συνάρτηση αυτή καθαρίζει το φόντο και ξεκινά την αναδρομική σχεδίαση
 * της σπείρας Fibonacci από το κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός του παραθύρου με λευκό φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 1.0f;
    bg.fill_color[1] = 1.0f;
    bg.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Κλήση της συνάρτησης σχεδίασης Fibonacci σπείρας από το κέντρο της οθόνης
    drawFibonacciSpiral(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 6, 12);
}

/**
 * @brief Κύρια συνάρτηση του προγράμματος.
 *
 * Δημιουργεί το παράθυρο και καθορίζει την συνάρτηση σχεδίασης για την απεικόνιση του
fractal.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Fibonacci Spiral Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή του Κώδικα

1. `generateFibonacci`: Παράγει την ακολουθία Fibonacci ως διάνυσμα, όπου κάθε τιμή είναι η πλευρά του επόμενου τετραγώνου της σπείρας. Εδώ, χρησιμοποιείται ένας παράγοντας κλίμακας για τη μεγέθυνση ή σμίκρυνση των τετραγώνων.
2. `drawFibonacciSpiral`: Σχεδιάζει τη σπείρα Fibonacci χρησιμοποιώντας την ακολουθία που παράγεται. Σε κάθε βήμα:
    - Τοποθετείται ένα τετράγωνο βασισμένο στον αριθμό Fibonacci.
    - Ένα τόξο σχεδιάζεται από το κέντρο κάθε τετραγώνου, προχωρώντας στη σπείρα.
3. `draw`: Καθαρίζει το υπόβαθρο και καλεί τη συνάρτηση `drawFibonacciSpiral` από το κέντρο της οθόνης.
4. `main`: Δημιουργεί το παράθυρο, καθορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Αυτός ο κώδικας δημιουργεί μια οπτικά εντυπωσιακή σπείρα που αναπτύσσεται σύμφωνα με την ακολουθία Fibonacci, αποκαλύπτοντας τη μαθηματική συμμετρία που παρατηρείται στη φύση.

Triangular Spiral Fractal



Ακολουθεί το πρόγραμμα σε C++ για τον σχεδιασμό του **Triangular Spiral Fractal**, το οποίο δημιουργεί μια σπείρα που αποτελείται από επαναλαμβανόμενα τρίγωνα που περιστρέφονται σε κανονικά διαστήματα. Η απόσταση αυξάνεται με κάθε βήμα για να δημιουργηθεί η σπειροειδής μορφή.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
```

```
const float WINDOW_WIDTH = 700;
const float WINDOW_HEIGHT = 700;

/**
 * @brief Σχεδιάζει ένα ισόπλευρο τρίγωνο με βάση το σημείο εκκίνησης, το μήκος
πλευράς και τη γωνία.
 *
 * @param x Συντεταγμένη x για το σημείο εκκίνησης.
 * @param y Συντεταγμένη y για το σημείο εκκίνησης.
 * @param sideLength Το μήκος πλευράς του τριγώνου.
 * @param angle Η γωνία περιστροφής του τριγώνου σε ακτίνια.
 * @param brush Το αντικείμενο πινέλου για την απόδοση του τριγώνου.
 */
void drawTriangle(float x, float y, float sideLength, float angle, const
graphics::Brush& brush) {
    // Υπολογισμός ύψους ισόπλευρου τριγώνου
    float height = sideLength * sqrt(3) / 2;

    // Υπολογισμός των τριών κορυφών του τριγώνου με βάση τη γωνία
    float x1 = x + sideLength * cos(angle);
    float y1 = y + sideLength * sin(angle);

    float x2 = x + sideLength * cos(angle + 120 * M_PI / 180.0f);
    float y2 = y + sideLength * sin(angle + 120 * M_PI / 180.0f);

    // Σχεδίαση του τριγώνου με τις τρεις πλευρές
    graphics::drawLine(x, y, x1, y1, brush);
    graphics::drawLine(x1, y1, x2, y2, brush);
    graphics::drawLine(x2, y2, x, y, brush);
}

/**
 * @brief Σχεδιάζει το Triangular Spiral Fractal.
 *
 * Η σπείρα σχηματίζεται από μια αλληλουχία τριγώνων που περιστρέφονται και αυξάνονται
σε μέγεθος,
 * δημιουργώντας ένα fractal σπειροειδούς μορφής.
 *
 * @param startX Συντεταγμένη x για το σημείο εκκίνησης της σπείρας.
 * @param startY Συντεταγμένη y για το σημείο εκκίνησης της σπείρας.
 * @param depth Ο αριθμός των τριγώνων στη σπείρα.
 * @param initialSideLength Το αρχικό μήκος πλευράς του πρώτου τριγώνου.
 * @param angleIncrement Η γωνία περιστροφής ανά τρίγωνο σε μοίρες.
 */
void drawTriangularSpiral(float startX, float startY, int depth, float
initialSideLength, float angleIncrement) {
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;
    brush.fill_color[1] = 0.6f;
    brush.fill_color[2] = 1.0f;

    float currentAngle = 0;          // Αρχική γωνία περιστροφής
    float currentX = startX;          // Αρχική θέση x
    float currentY = startY;          // Αρχική θέση y
    float sideLength = initialSideLength; // Αρχικό μήκος πλευράς

    // Αναδρομική σχεδίαση τριγώνων για τη δημιουργία της σπείρας
    for (int i = 0; i < depth; i++) {
        // Σχεδίαση του τρέχοντος τριγώνου
        drawTriangle(currentX, currentY, sideLength, currentAngle, brush);

        // Υπολογισμός νέας θέσης και γωνίας για το επόμενο τρίγωνο
        currentX += sideLength * cos(currentAngle);
        currentY += sideLength * sin(currentAngle);
```

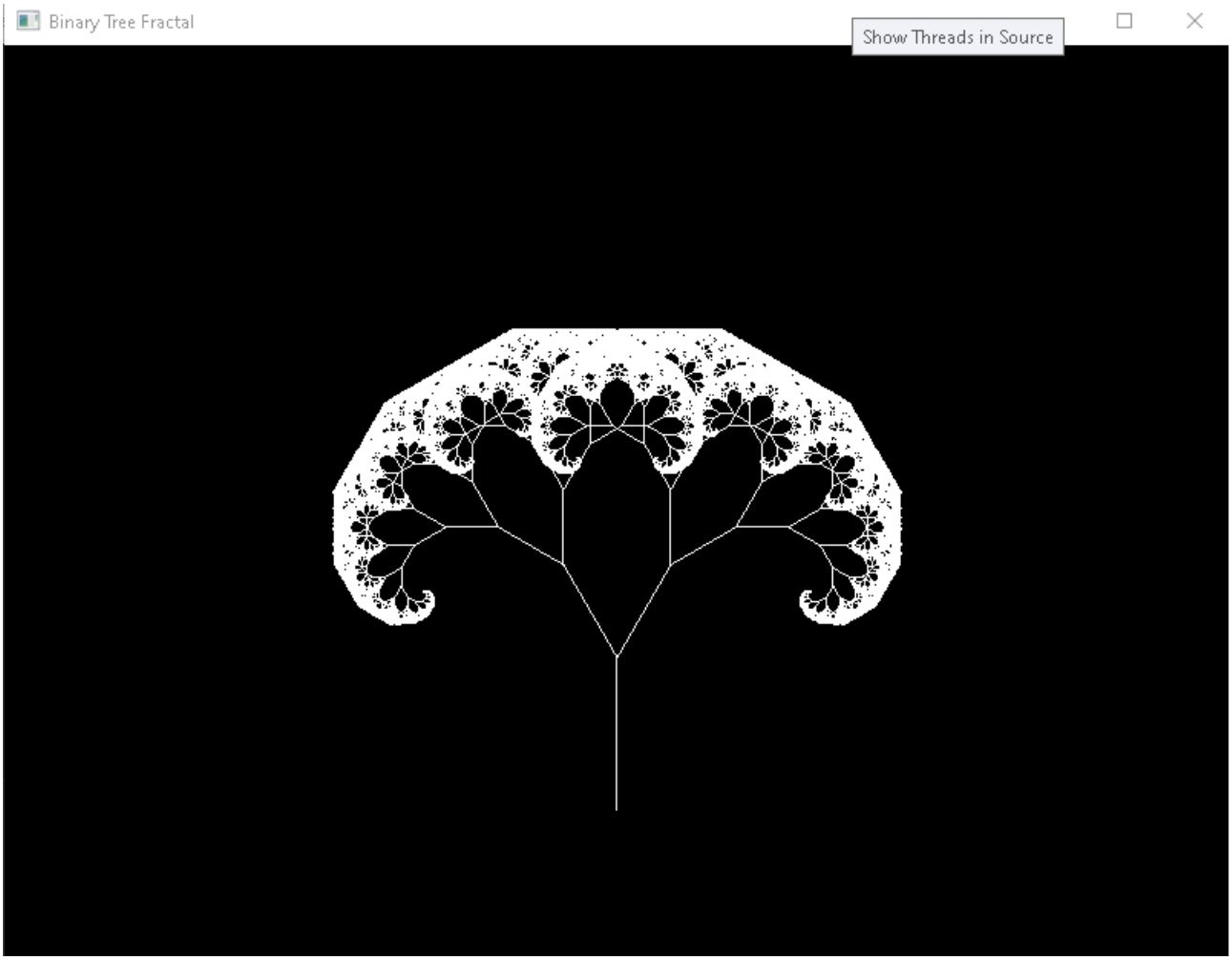
```
        // Αύξηση πλευράς και περιστροφή γωνίας για τη σπείρα
        sideLength += initialSideLength * 0.1f;
        currentAngle += angleIncrement * M_PI / 180.0f;
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Καθαρίζει το φόντο και ξεκινά τη σχεδίαση του Triangular Spiral Fractal
 * από το κέντρο του παραθύρου.
 */
void draw() {
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;  // Μαύρο φόντο
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Εκκίνηση σχεδίασης της σπείρας από το κέντρο του παραθύρου
    drawTriangularSpiral(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 400, 2, 15);
}

/**
 * @brief Κύρια συνάρτηση που εκκινεί το πρόγραμμα.
 *
 * Δημιουργεί το παράθυρο και καθορίζει τη συνάρτηση σχεδίασης για να απεικονιστεί το
fractal.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Triangular Spiral Fractal");

    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή του Κώδικα

1. `drawTriangle`: Σχεδιάζει ένα ισόπλευρο τρίγωνο με συγκεκριμένη γωνία προσανατολισμού, χρησιμοποιώντας τις κορυφές που υπολογίζονται με τριγωνομετρικές συναρτήσεις.
2. `drawTriangularSpiral`: Δημιουργεί μια σπειροειδή ακολουθία από τρίγωνα.
    - Ξεκινά από ένα κέντρο και σε κάθε βήμα σχεδιάζει ένα τρίγωνο.
    - Η πλευρά αυξάνεται σταδιακά με κάθε βήμα, ενώ η γωνία προσανατολισμού αλλάζει για να δημιουργήσει τη σπείρα.
3. `draw`: Καθαρίζει το υπόβαθρο και καλεί τη συνάρτηση `drawTriangularSpiral` από το κέντρο της οθόνης.
4. `main`: Δημιουργεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων.

Αυτό το πρόγραμμα παράγει μια όμορφη σπείρα από τρίγωνα που μοιάζει με φράκταλ, χρησιμοποιώντας τη δυναμική αυξομείωση πλευρών και περιστροφών, δίνοντας ένα εντυπωσιακό αποτέλεσμα.

Binary Tree Fractal



Ο παρακάτω κώδικας δημιουργεί ένα **Binary Tree Fractal** χρησιμοποιώντας αναδρομή για να σχεδιάσει τα κλαδιά του δέντρου. Σε κάθε επανάληψη, το πρόγραμμα σχεδιάζει δύο μικρότερα κλαδιά σε καθορισμένες γωνίες, μέχρι να φτάσει στο επιθυμητό βάθος.

Κώδικας

```
#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

// Ορισμός παραμέτρων παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το δυαδικό δέντρο fractal.
 *
 * Η συνάρτηση σχεδιάζει ένα κλαδί, στη συνέχεια καλείται αναδρομικά για να σχεδιάσει δύο
 * μικρότερα κλαδιά στα άκρα, καθένα σε γωνία απόκλισης από το προηγούμενο.
 *
 * @param x Συντεταγμένη x για την αρχή του κλαδιού.
 * @param y Συντεταγμένη y για την αρχή του κλαδιού.
 * @param length Το μήκος του τρέχοντος κλαδιού.
```

```
 * @param angle Η γωνία με την οποία εκτείνεται το κλαδί.
 * @param depth Το βάθος της αναδρομής, καθορίζει το πλήθος των επιπέδων του δέντρου.
 * @param brush Το πινέλο που χρησιμοποιείται για τη σχεδίαση του κλαδιού.
 */
void drawBinaryTree(float x, float y, float length, float angle, int depth, const
graphics::Brush& brush) {
    if (depth == 0) return;  // Βάση της αναδρομής - σταματάμε όταν το βάθος είναι 0

    // Υπολογισμός των νέων συντεταγμένων του άκρου του κλαδιού
    float newX = x + length * cos(angle);
    float newY = y - length * sin(angle);  // Αφαιρούμε για να σχεδιάσουμε προς τα
πάνω

    // Σχεδίαση του κλαδιού
    graphics::drawLine(x, y, newX, newY, brush);

    // Αναδρομικές κλήσεις για τα δύο κλαδιά με αλλαγές στη γωνία και το μήκος
    float branchAngle = M_PI / 6;           // Γωνία διακλάδωσης (30 μοίρες)
    float branchLengthFactor = 0.7f;        // Μείωση του μήκους του κλαδιού σε κάθε
επίπεδο

    // Κλήση για το αριστερό και δεξί κλαδί
    drawBinaryTree(newX, newY, length * branchLengthFactor, angle - branchAngle, depth
- 1, brush);
    drawBinaryTree(newX, newY, length * branchLengthFactor, angle + branchAngle, depth
- 1, brush);
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Καθαρίζει το φόντο και ξεκινά τη σχεδίαση του Binary Tree Fractal
 * από το κέντρο της βάσης του παραθύρου.
 */
void draw() {
    // Καθαρισμός του φόντου
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Μαύρο φόντο
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Δημιουργία του πινέλου για το δέντρο
    graphics::Brush treeBrush;
    treeBrush.fill_color[0] = 0.0f;
    treeBrush.fill_color[1] = 0.5f;
    treeBrush.fill_color[2] = 0.0f;

    // Κλήση της συνάρτησης για σχεδίαση του fractal δέντρου
    drawBinaryTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 100, 100, M_PI / 2, 20,
treeBrush);
}

/**
 * @brief Κύρια συνάρτηση που εκκινεί το πρόγραμμα.
 *
 * Δημιουργεί το παράθυρο και καθορίζει τη συνάρτηση σχεδίασης για να απεικονιστεί το
fractal.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Binary Tree Fractal");
```

```
    // Ορισμός της συνάρτησης σχεδίασης
    graphics::setDrawFunction(draw);

    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```

Περιγραφή του Κώδικα

1. `drawBinaryTree`: Αυτή η συνάρτηση σχεδιάζει αναδρομικά το Binary Tree Fractal.
    - Ξεκινά με το αρχικό κλαδί και αναδρομικά προσθέτει δύο μικρότερα κλαδιά σε κάθε άκρο, στρέφοντας τα κατά μια καθορισμένη γωνία.
    - Με κάθε αναδρομική κλήση, το μήκος του κλαδιού μειώνεται, προσδίδοντας την εμφάνιση ενός δέντρου.
    - Το βάθος καθορίζει πόσες διακλαδώσεις θα σχεδιαστούν.
2. `draw`: Αυτή η συνάρτηση καλείται σε κάθε επανάληψη του προγράμματος και καλεί τη `drawBinaryTree` από το κάτω μέρος του παραθύρου, σχηματίζοντας το βασικό κλαδί.
3. `main`: Αρχικοποιεί το παράθυρο, ορίζει τη συνάρτηση σχεδίασης και ξεκινά τον κύκλο μηνυμάτων του παραθύρου.

Το πρόγραμμα δημιουργεί ένα όμορφο δέντρο με συμμετρικά κλαδιά που μειώνονται σε μήκος καθώς αναπτύσσονται προς τα έξω, δημιουργώντας ένα κλασσικό **Binary Tree Fractal**.

Fractal.h header file and main function example

Header file: fractals.h

```
#pragma once
// fractals.h
#ifndef FRACTALS_H
#define FRACTALS_H

#include "sgg/graphics.h"
#include <cmath>
#include <random>
#include <vector>
#include <cstdlib>   // Χρησιμοποιείται για παραγωγή τυχαίων αριθμών
#include <ctime>     // Χρησιμοποιείται για αρχικοποίηση των τυχαίων αριθμών
#include <string>
#include <iostream>
#include <stack>

#define M_PI 3.14159265358979323846



/**
 * Namespace για το fractal "Sierpinski Triangle"
 */
namespace SierpinskiTriangle {
    const float windowWidth = 800;
    const float windowHeight = 600;
    // Ορισμός του μέγιστου βάθους αναδρομής για το fractal
    const int max_depth = 5; // Αυξάνοντας το βάθος, αυξάνεται η λεπτομέρεια του fractal

    /**
```

```
    * Συνάρτηση σχεδίασης τριγώνου που σχεδιάζει ένα τρίγωνο, δεδομένων των συντεταγμένων των κορυφών του.
    * @param x1, y1 Συντεταγμένες πρώτης κορυφής
    * @param x2, y2 Συντεταγμένες δεύτερης κορυφής
    * @param x3, y3 Συντεταγμένες τρίτης κορυφής
    * @param br Το πινέλο (Brush) που χρησιμοποιείται για το χρώμα και το στυλ του τριγώνου
    */
   void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, graphics::Brush& br) {
      graphics::drawLine(x1, y1, x2, y2, br);
      graphics::drawLine(x2, y2, x3, y3, br);
      graphics::drawLine(x3, y3, x1, y1, br);
   }

   /**
    * Αναδρομική συνάρτηση σχεδίασης του Sierpinski Triangle.
    * Σε κάθε επίπεδο αναδρομής, η συνάρτηση υποδιαιρεί το τρίγωνο σε τρία μικρότερα τρίγωνα
    * και σχεδιάζει μόνο όταν φτάσει στο προκαθορισμένο βάθος.
    * @param x1, y1 Συντεταγμένες πρώτης κορυφής
    * @param x2, y2 Συντεταγμένες δεύτερης κορυφής
    * @param x3, y3 Συντεταγμένες τρίτης κορυφής
    * @param depth Το τρέχον βάθος αναδρομής
    * @param br Το πινέλο (Brush) για το χρώμα και το στυλ του fractal
    */
   void sierpinski(float x1, float y1, float x2, float y2, float x3, float y3, int depth, graphics::Brush& br) {
      // Βασική περίπτωση: αν το βάθος είναι μηδενικό, σχεδιάζουμε το τρίγωνο
      if (depth == 0) {
         drawTriangle(x1, y1, x2, y2, x3, y3, br);
      }
      else {
         // Υπολογισμός των μέσων σημείων κάθε πλευράς του τριγώνου
         float mid_x1 = (x1 + x2) / 2.0f;
```

```
        float mid_y1 = (y1 + y2) / 2.0f;
        float mid_x2 = (x2 + x3) / 2.0f;
        float mid_y2 = (y2 + y3) / 2.0f;
        float mid_x3 = (x3 + x1) / 2.0f;
        float mid_y3 = (y3 + y1) / 2.0f;

        // Αναδρομική κλήση για κάθε ένα από τα τρία νέα τρίγωνα που δημιουργούνται
        sierpinski(x1, y1, mid_x1, mid_y1, mid_x3, mid_y3, depth - 1, br);
        sierpinski(mid_x1, mid_y1, x2, y2, mid_x2, mid_y2, depth - 1, br);
        sierpinski(mid_x3, mid_y3, mid_x2, mid_y2, x3, y3, depth - 1, br);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από την βιβλιοθήκη για την απόδοση του παραθύρου.
 * Χρησιμοποιεί το πινέλο και τις συντεταγμένες για τη σχεδίαση του Sierpinski Triangle.
 */
void draw() {
    // Ορισμός του πινέλου για τη σχεδίαση (λευκό χρώμα για τις γραμμές του τριγώνου)
    graphics::Brush br;
    br.fill_color[0] = 1.0f;  // Κόκκινο
    br.fill_color[1] = 1.0f;  // Πράσινο
    br.fill_color[2] = 1.0f;  // Μπλε (RGB για λευκό χρώμα)

    // Συντεταγμένες κορυφών του βασικού τριγώνου στην οθόνη
    float x1 = windowWidth / 2.0f;
    float y1 = 50.0f;
    float x2 = 50.0f;
    float y2 = windowHeight - 50.0f;
    float x3 = windowWidth - 50.0f;
    float y3 = windowHeight - 50.0f;
```

```
        // Κλήση της συνάρτησης για να σχεδιάσει το Sierpinski Triangle
        sierpinski(x1, y1, x2, y2, x3, y3, max_depth, br);
    }

}

/**
 * Namespace για το fractal "SierpinskiCarpet"
 */
namespace SierpinskiCarpet {
    // Ορισμός των διαστάσεων του παραθύρου σε pixels
    const int windowWidth = 600;
    const int windowHeight = 600;

    // Ορισμός του μέγιστου βάθους αναδρομής για τη σχεδίαση του fractal
    const int max_depth = 5; // Αυξάνοντας το βάθος, αυξάνεται η λεπτομέρεια του fractal

    /**
     * Αναδρομική συνάρτηση που σχεδιάζει το Sierpinski Carpet Fractal.
     * Σε κάθε επίπεδο αναδρομής, η συνάρτηση υποδιαιρεί το τετράγωνο σε 9 μικρότερα
     * τετράγωνα, αφήνοντας το κεντρικό κενό και σχεδιάζει τα υπόλοιπα.
     * @param x Το x του κεντρικού σημείου του τετραγώνου προς σχεδίαση
     * @param y Το y του κεντρικού σημείου του τετραγώνου προς σχεδίαση
     * @param size Το μέγεθος της πλευράς του τετραγώνου
     * @param depth Το τρέχον βάθος αναδρομής
     */
    void sierpinski_carpet(float x, float y, float size, int depth) {
        // Βασική περίπτωση: Όταν φτάσουμε στο μέγιστο βάθος, σχεδιάζουμε το τετράγωνο
        if (depth == 0) {
            graphics::Brush br;
            br.fill_color[0] = 1.0f;  // Ορισμός χρώματος (λευκό) για τα τετράγωνα του fractal
            br.fill_color[1] = 1.0f;
```

```
        br.fill_color[2] = 1.0f;
        graphics::drawRect(x, y, size, size, br);  // Σχεδίαση του λευκού τετραγώνου
    }
    else {
        // Υπολογισμός του νέου μεγέθους για τα μικρότερα τετράγωνα
        float newSize = size / 3.0f;

        // Διάσχιση των εννέα θέσεων για να τοποθετηθούν τα νέα τετράγωνα γύρω από το κεντρικό
        for (int dx = 0; dx < 3; dx++) {
            for (int dy = 0; dy < 3; dy++) {
                // Παράκαμψη του κεντρικού τετραγώνου
                if (dx == 1 && dy == 1) continue;

                // Αναδρομική κλήση για τη σχεδίαση του τετραγώνου στα υπόλοιπα οκτώ σημεία
                sierpinski_carpet(x + dx * newSize, y + dy * newSize, newSize, depth - 1);
            }
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
 * Ορίζει το φόντο και ξεκινά τη σχεδίαση του Sierpinski Carpet.
 */
void draw() {
    // Ορισμός του φόντου του παραθύρου σε μαύρο χρώμα για αντίθεση
    graphics::Brush background;
    background.fill_color[0] = 0.0f; // Κόκκινο
    background.fill_color[1] = 0.0f; // Πράσινο
    background.fill_color[2] = 0.0f; // Μπλε (RGB για μαύρο χρώμα)
    graphics::setWindowBackground(background);
```

```
        // Κλήση της συνάρτησης για τη σχεδίαση του Sierpinski Carpet, ξεκινώντας από το κέντρο του παραθύρου
        sierpinski_carpet(0, 0, windowWidth, max_depth);
    }

}

/**
 * Namespace για το fractal "Koch Snowflake"
 */
namespace KochSnowflake {
    // Ορισμός διαστάσεων παραθύρου σε pixels
    const int windowWidth = 600;
    const int windowHeight = 600;

    // Μέγιστος αριθμός επαναλήψεων για τη σχεδίαση του fractal Koch Snowflake
    const int max_depth = 4; // Αυξάνοντας το βάθος, προστίθενται περισσότερες λεπτομέρειες

    /**
     * Συνάρτηση που σχεδιάζει το Koch Snowflake αναδρομικά.
     * Διαιρεί κάθε γραμμή σε τρία τμήματα, δημιουργώντας ένα ισόπλευρο τρίγωνο στο μέσο της γραμμής.
     * @param x1 Η x-συντεταγμένη του αρχικού σημείου της γραμμής
     * @param y1 Η y-συντεταγμένη του αρχικού σημείου της γραμμής
     * @param x2 Η x-συντεταγμένη του τελικού σημείου της γραμμής
     * @param y2 Η y-συντεταγμένη του τελικού σημείου της γραμμής
     * @param depth Το τρέχον βάθος αναδρομής
     * @param br Το πινέλο (brush) που χρησιμοποιείται για τη σχεδίαση
     */
    void koch_snowflake(float x1, float y1, float x2, float y2, int depth, graphics::Brush& br) {
        // Βασική περίπτωση: Όταν φτάσουμε στο μέγιστο βάθος, σχεδιάζουμε την απλή γραμμή
```

```
    if (depth == 0) {
        graphics::drawLine(x1, y1, x2, y2, br);
    }
    else {
        // Υπολογισμός των τριών σημείων διαίρεσης της γραμμής
        float dx = (x2 - x1) / 3.0f;
        float dy = (y2 - y1) / 3.0f;

        // Σημεία στα 1/3 και 2/3 της γραμμής
        float xA = x1 + dx;
        float yA = y1 + dy;
        float xB = x1 + 2 * dx;
        float yB = y1 + 2 * dy;

        // Υπολογισμός του σημείου κορυφής του ισόπλευρου τριγώνου που δημιουργείται
        float midX = (x1 + x2) / 2.0f;
        float midY = (y1 + y2) / 2.0f;
        float peakX = midX + sqrt(3.0f) * (yA - yB) / 2.0f;
        float peakY = midY + sqrt(3.0f) * (xB - xA) / 2.0f;

        // Αναδρομική κλήση για τις τέσσερις γραμμές που αποτελούν το επόμενο επίπεδο
        koch_snowflake(x1, y1, xA, yA, depth - 1, br);
        koch_snowflake(xA, yA, peakX, peakY, depth - 1, br);
        koch_snowflake(peakX, peakY, xB, yB, depth - 1, br);
        koch_snowflake(xB, yB, x2, y2, depth - 1, br);
    }
}

/**
 * Συνάρτηση σχεδίασης για το Koch Snowflake, βασισμένη σε ισόπλευρο τρίγωνο.
 * Αναδρομικά καλεί τη `koch_snowflake` για τη σχεδίαση και των τριών πλευρών.
 */
```

```
void drawKochSnowflake() {
    graphics::Brush br;
    br.fill_color[0] = 1.0f;  // Λευκό χρώμα για τις γραμμές του fractal
    br.fill_color[1] = 1.0f;
    br.fill_color[2] = 1.0f;

    // Συντεταγμένες για τις κορυφές του βασικού ισόπλευρου τριγώνου
    float x1 = windowWidth / 2.0f;
    float y1 = 100.0f;
    float x2 = 200.0f;
    float y2 = windowHeight - 100.0f;
    float x3 = windowWidth - 200.0f;
    float y3 = windowHeight - 100.0f;

    // Κλήση της συνάρτησης για σχεδίαση του Koch Snowflake στις τρεις πλευρές του τριγώνου
    koch_snowflake(x1, y1, x2, y2, max_depth, br);
    koch_snowflake(x2, y2, x3, y3, max_depth, br);
    koch_snowflake(x3, y3, x1, y1, max_depth, br);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
 * Καθορίζει το φόντο και ξεκινά τη σχεδίαση του Koch Snowflake.
 */
void draw() {
    graphics::Brush background;
    background.fill_color[0] = 0.0f; // Μαύρο χρώμα φόντου για αντίθεση
    background.fill_color[1] = 0.0f;
    background.fill_color[2] = 0.0f;
    graphics::setWindowBackground(background);

    // Κλήση της συνάρτησης για να σχεδιάσει το Koch Snowflake
```

```
        drawKochSnowflake();
    }
}
 /**
 * Namespace για το fractal "Barnsley Fern"
 */
    namespace BarnsleyFern {

        // Ορισμός διαστάσεων παραθύρου
        const int  windowWidth = 350;   // Πλάτος παραθύρου
        const int  windowHeight = 700;  // Ύψος παραθύρου

        // Αριθμός επαναλήψεων
        const int num_iterations = 100000; // Ο αριθμός σημείων που θα σχεδιαστούν για το Barnsley Fern

        // Συντελεστές για μετασχηματισμό συντεταγμένων στον καμβά σχεδίασης
        const float scale_x = 60.0f;   // Κλιμάκωση στον άξονα x για κατάλληλη αναλογία
        const float scale_y = 60.0f;   // Κλιμάκωση στον άξονα y για κατάλληλη αναλογία
        const float offset_x = windowWidth / 2.0f;       // Μετατόπιση κέντρου στον x άξονα
        const float offset_y = windowHeight - 50.0f;     // Μετατόπιση προς τα κάτω στον y άξονα

        /**
         * Συνάρτηση barnsley_fern
         * Εφαρμόζει τέσσερις διαφορετικούς affine μετασχηματισμούς αναδρομικά για να δημιουργήσει το Barnsley Fern.
         * Οι μετασχηματισμοί επιλέγονται τυχαία, δημιουργώντας τον χαρακτηριστικό σχηματισμό του φτέρη (fern).
         * Σχεδιάζει ένα σημείο κάθε φορά στο παράθυρο.
         */
        void barnsley_fern() {
            graphics::Brush br;  // Το πινέλο για το χρώμα του fractal (προεπιλεγμένο χρώμα)
```

```
        float x = 0.0f;   // Αρχική x-συντεταγμένη
        float y = 0.0f;   // Αρχική y-συντεταγμένη

        // Βρόχος για τον υπολογισμό και σχεδίαση των σημείων
        for (int i = 0; i < num_iterations; ++i) {
            float next_x, next_y;
            float r = static_cast<float>(rand()) / RAND_MAX;  // Τυχαίος αριθμός [0,1] για επιλογή μετασχηματισμού

            // Επιλογή μετασχηματισμού βάσει της πιθανότητας που ορίζεται για το κάθε fractal τμήμα
            if (r < 0.01f) {  // Πιθανότητα F1
                // F1: Μικρό τμήμα στο κάτω μέρος της φτέρης
                next_x = 0.0f;
                next_y = 0.16f * y;
            }
            else if (r < 0.86f) {  // Πιθανότητα F2
                // F2: Κύριο τμήμα της φτέρης, καθορίζει τη δομή
                next_x = 0.85f * x + 0.04f * y;
                next_y = -0.04f * x + 0.85f * y + 1.6f;
            }
            else if (r < 0.93f) {  // Πιθανότητα F3
                // F3: Τμήμα που δημιουργεί τις πλευρικές "φύτρες" της φτέρης
                next_x = 0.2f * x - 0.26f * y;
                next_y = 0.23f * x + 0.22f * y + 1.6f;
            }
            else {  // Πιθανότητα F4
                // F4: Τμήμα που προσθέτει περισσότερες μικρές "φύτρες"
                next_x = -0.15f * x + 0.28f * y;
                next_y = 0.26f * x + 0.24f * y + 0.44f;
            }
```

```
            // Ενημέρωση των τιμών των x και y για την επόμενη επανάληψη
            x = next_x;
            y = next_y;

            // Σχεδιασμός του σημείου στο παράθυρο με μετασχηματισμό για την αντιστοίχιση στον καμβά
            graphics::drawRect(offset_x + x * scale_x, offset_y - y * scale_y, 1, 1, br);
        }
    }

    /**
     * Συνάρτηση σχεδίασης draw
     * Καλεί τη συνάρτηση barnsley_fern για τη σχεδίαση του fractal Barnsley Fern στον καμβά.
     */
    void draw() {
        barnsley_fern();  // Κλήση της συνάρτησης που σχεδιάζει το Barnsley Fern
    }

}
/**
 * Namespace για το fractal "Dragon Curve "
*/
namespace DragonCurve {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;  // Πλάτος παραθύρου
    const int windowHeight = 600; // Ύψος παραθύρου

    // Μέγιστο βάθος αναδρομής για τη λεπτομέρεια του Dragon Curve
    const int max_depth = 12; // Αύξηση αυτής της τιμής αυξάνει τη λεπτομέρεια του fractal

    /**
     * Αναδρομική συνάρτηση για τη σχεδίαση του Dragon Curve fractal.
```

```
     * @param x1 Η αρχική x-συντεταγμένη της γραμμής
     * @param y1 Η αρχική y-συντεταγμένη της γραμμής
     * @param x2 Η τελική x-συντεταγμένη της γραμμής
     * @param y2 Η τελική y-συντεταγμένη της γραμμής
     * @param depth Το τρέχον βάθος αναδρομής (σταματάει όταν φτάσει στο 0)
     */
    void drawDragonCurve(float x1, float y1, float x2, float y2, int depth) {
        if (depth == 0) {
            // Εάν το βάθος είναι 0, σχεδιάζουμε τη γραμμή από (x1, y1) έως (x2, y2)
            graphics::Brush br;  // Δημιουργία αντικειμένου πινέλου για τον καθορισμό στυλ
            graphics::drawLine(x1, y1, x2, y2, br);  // Σχεδιάζει γραμμή με τις δεδομένες συντεταγμένες
        }
        else {
            // Υπολογισμός των συντεταγμένων του μέσου σημείου με περιστροφή κατά 45 μοίρες
            float mid_x = (x1 + x2) / 2.0f + (y2 - y1) / 2.0f;
            float mid_y = (y1 + y2) / 2.0f - (x2 - x1) / 2.0f;

            // Αναδρομική κλήση για τα δύο μέρη του fractal
            drawDragonCurve(x1, y1, mid_x, mid_y, depth - 1);  // Πρώτο μισό
            drawDragonCurve(mid_x, mid_y, x2, y2, depth - 1);  // Δεύτερο μισό
        }
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ για τη σχεδίαση του Dragon Curve.
     */
    void draw() {
        graphics::Brush background; // Αντικείμενο πινέλου για το φόντο
        graphics::setWindowBackground(background); // Ορισμός μαύρου φόντου για αντίθεση

        // Ορισμός αρχικών και τελικών σημείων για την αρχική γραμμή
```

```
        float startX = windowWidth / 3.0f;  // Αρχική x-συντεταγμένη
        float startY = windowHeight / 2.0f; // Αρχική y-συντεταγμένη
        float endX = 2.0f * windowWidth / 3.0f; // Τελική x-συντεταγμένη
        float endY = windowHeight / 2.0f;       // Τελική y-συντεταγμένη

        // Κλήση της αναδρομικής συνάρτησης για τη σχεδίαση του Dragon Curve
        drawDragonCurve(startX, startY, endX, endY, max_depth);
    }

}
/**
* Namespace για το fractal "Cantor Set"
*/
namespace CantorSet {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;  // Πλάτος παραθύρου
    const int windowHeight = 600; // Ύψος παραθύρου

    // Ορισμός του μέγιστου βάθους αναδρομής για το Cantor Set
    const int max_depth = 15;  // Ο αριθμός των επιπέδων διακεκομμένων γραμμών
    // Καθορίζει την κατακόρυφη απόσταση μεταξύ των γραμμών
    const int line_spacing = 10;  // Απόσταση μεταξύ των διαδοχικών επιπέδων

    /**
     * Συνάρτηση που σχεδιάζει το Cantor Set χρησιμοποιώντας αναδρομή
     * @param x Η αρχική x-συντεταγμένη της γραμμής
     * @param y Η αρχική y-συντεταγμένη της γραμμής
     * @param length Το μήκος της τρέχουσας γραμμής
     * @param depth Το τρέχον βάθος αναδρομής (σταματά όταν φτάσει στο 0)
     */
    void drawCantorSet(float x, float y, float length, int depth) {
        if (depth == 0) {
```

```
        return; // Σταματάει αν φτάσουμε το μέγιστο βάθος
    }

    // Δημιουργία πινέλου για σχεδίαση
    graphics::Brush br;

    // Σχεδίαση γραμμής στο τρέχον επίπεδο
    graphics::drawLine(x, y, x + length, y, br);

    // Υπολογισμός μήκους για το επόμενο επίπεδο
    float newLength = length / 3.0f;  // Μειώνουμε το μήκος κάθε νέας γραμμής στο ένα τρίτο

    // Αναδρομική κλήση για τις δύο νέες γραμμές
    drawCantorSet(x, y + line_spacing, newLength, depth - 1);  // Αριστερή γραμμή
    drawCantorSet(x + 2 * newLength, y + line_spacing, newLength, depth - 1);  // Δεξιά γραμμή
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη για κάθε καρέ
 */
void draw() {
    // Δημιουργία πινέλου για το φόντο
    graphics::Brush background;
    graphics::setWindowBackground(background); // Ορισμός μαύρου φόντου

    // Αρχικές συντεταγμένες και μήκος για το Cantor Set
    float startX = 50.0f;    // Αρχικό x για την πρώτη γραμμή
    float startY = 50.0f;    // Αρχικό y για την πρώτη γραμμή
    float startLength = 500.0f;  // Μήκος της πρώτης γραμμής

    // Κλήση της συνάρτησης σχεδίασης για το Cantor Set
```

```
        drawCantorSet(startX, startY, startLength, max_depth);
    }



}
/**
* Namespace για το fractal "ApollonianGasket Set"
*/
namespace  ApollonianGasket {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;  // Πλάτος παραθύρου
    const int windowHeight = 600; // Ύψος παραθύρου

    // Ορισμός του μέγιστου βάθους αναδρομής για το Apollonian Gasket
    const int max_depth = 5;  // Ο αριθμός των επαναλήψεων για το βάθος του fractal

    /**
     * Συνάρτηση που σχεδιάζει έναν κύκλο με καθορισμένη ακτίνα και θέση
     * @param x Η x-συντεταγμένη του κέντρου του κύκλου
     * @param y Η y-συντεταγμένη του κέντρου του κύκλου
     * @param radius Η ακτίνα του κύκλου
     * @param br Το πινέλο (Brush) που χρησιμοποιείται για το χρώμα του κύκλου
     */
    void drawCircle(float x, float y, float radius, graphics::Brush& br) {
        graphics::drawDisk(x, y, radius, br);
    }

    /**
     * Αναδρομική συνάρτηση που δημιουργεί το Apollonian Gasket fractal
     * @param x Η x-συντεταγμένη του κέντρου του κύκλου
     * @param y Η y-συντεταγμένη του κέντρου του κύκλου
     * @param radius Η ακτίνα του τρέχοντος κύκλου
```

```
 * @param depth Το τρέχον βάθος αναδρομής. Η συνάρτηση σταματά όταν depth == 0.
 */
void apollonianGasket(float x, float y, float radius, int depth) {
    if (depth == 0 || radius < 1.0f) {  // Όριο για το βάθος και το μέγεθος του κύκλου
        return;
    }

    // Δημιουργία πινέλου για τον κύκλο
    graphics::Brush br;
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.8f; // Προσαρμογή χρώματος
    br.fill_color[1] = 0.2f;
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.8f;

    // Σχεδίαση του κεντρικού κύκλου
    drawCircle(x, y, radius, br);

    // Υπολογισμός ακτίνας για τους επόμενους μικρότερους κύκλους
    float newRadius = radius / 2.0f;

    // Αναδρομική κλήση για τους τρεις νέους κύκλους σε διάφορες θέσεις
    apollonianGasket(x - newRadius, y, newRadius, depth - 1);        // Αριστερά
    apollonianGasket(x + newRadius, y, newRadius, depth - 1);        // Δεξιά
    apollonianGasket(x, y - newRadius, newRadius, depth - 1);        // Κάτω
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);  // Ορισμός μαύρου φόντου
```

```
        // Ορισμός κεντρικής θέσης και αρχικής ακτίνας για το Apollonian Gasket
        float centerX = windowWidth / 2.0f;
        float centerY = windowHeight / 2.0f;
        float initialRadius = 300.0f;  // Ακτίνα για τον πρώτο κύκλο

        // Κλήση της αναδρομικής συνάρτησης για να σχεδιάσει το Apollonian Gasket
        apollonianGasket(centerX, centerY, initialRadius, max_depth);
    }

}
/**
* Namespace για το fractal "Peano Curve"
*/
namespace PeanoCurve {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;   // Πλάτος παραθύρου σε pixel
    const int windowHeight = 600;  // Ύψος παραθύρου σε pixel

    // Μέγιστο βάθος αναδρομής για τη δημιουργία της Peano Curve
    const int max_depth = 4;  // Μπορεί να αυξηθεί για περισσότερη λεπτομέρεια

    /**
     * Αναδρομική συνάρτηση για σχεδίαση της Peano Curve
     * @param x Η x-συντεταγμένη για την αρχική θέση σχεδίασης
     * @param y Η y-συντεταγμένη για την αρχική θέση σχεδίασης
     * @param length Το μήκος της τρέχουσας πλευράς του τετραγώνου
     * @param depth Το τρέχον βάθος αναδρομής
     * @param orientation Προσανατολισμός της σχεδίασης (δεν χρησιμοποιείται εδώ)
     *
     * @details Αυτή η συνάρτηση σχεδιάζει τη Peano Curve με αναδρομή. Σε κάθε επίπεδο
     * βάθους, το τετράγωνο διαιρείται σε εννέα μικρότερα τετράγωνα και σχεδιάζεται
     * γραμμικά με μπλε χρώμα.
```

```
 */
void drawPeanoCurve(float x, float y, float length, int depth, int orientation) {
    if (depth == 0) {
        // Σχεδιάζουμε ένα τετράγωνο όταν φτάσουμε στο μέγιστο βάθος
        graphics::Brush br;
        br.fill_color[0] = 0.0f;
        br.fill_color[1] = 0.0f;
        br.fill_color[2] = 1.0f;  // Μπλε χρώμα για τη γραμμή
        graphics::drawRect(x, y, length, length, br);
    }
    else {
        // Υπολογισμός του νέου μήκους για το επόμενο επίπεδο
        float newLength = length / 3.0f;

        // Αναδρομική κλήση για κάθε μικρό τετράγωνο σύμφωνα με το μοτίβο Peano
        drawPeanoCurve(x, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + newLength, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + 2 * newLength, y, newLength, depth - 1, orientation);
        drawPeanoCurve(x + 2 * newLength, y + newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x + newLength, y + newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x, y + newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x, y + 2 * newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x + newLength, y + 2 * newLength, newLength, depth - 1, orientation);
        drawPeanoCurve(x + 2 * newLength, y + 2 * newLength, newLength, depth - 1,
orientation);
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
 *
 * @details Η συνάρτηση αυτή καλείται αυτόματα από τη βιβλιοθήκη και εκκινεί
```

```
     * τη σχεδίαση της Peano Curve από την αρχική θέση στην οθόνη.
     */
    void draw() {
        graphics::Brush background;
        graphics::setWindowBackground(background);  // Ορισμός μαύρου φόντου

        // Αρχικές συντεταγμένες και μήκος για την Peano Curve
        float startX = 50.0f;    // x-συντεταγμένη εκκίνησης
        float startY = 50.0f;    // y-συντεταγμένη εκκίνησης
        float startLength = 700.0f;  // Αρχικό μήκος πλευράς του τετραγώνου

        // Κλήση της συνάρτησης για σχεδίαση της Peano Curve
        drawPeanoCurve(startX, startY, startLength, max_depth, 0);
    }

}
// namespace Hilbert Curve
namespace HilbertCurve {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;   // Πλάτος παραθύρου σε pixel
    const int windowHeight = 600;  // Ύψος παραθύρου σε pixel

    // Μέγιστο βάθος αναδρομής για τη δημιουργία της Hilbert Curve
    const int max_depth = 5;  // Αυξάνεται για περισσότερη λεπτομέρεια στη σχεδίαση

    /**
     * Αναδρομική συνάρτηση για σχεδίαση της Hilbert Curve
     * @param x Η x-συντεταγμένη της αρχικής θέσης σχεδίασης
     * @param y Η y-συντεταγμένη της αρχικής θέσης σχεδίασης
     * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου
     * @param depth Το τρέχον επίπεδο αναδρομής
     * @param rotation Η τρέχουσα κατεύθυνση περιστροφής (0, 1, 2, ή 3)
```

```
        *
        * @details Αυτή η συνάρτηση σχεδιάζει αναδρομικά την Hilbert Curve. Ανάλογα με το `rotation`,
        * κάθε κλήση της συνάρτησης διαμορφώνει το μοτίβο σε διαφορετικό προσανατολισμό για να
        * πετύχει τη διαδρομή χωρίς επικάλυψη.
        */
       void drawHilbertCurve(float x, float y, float size, int depth, int rotation) {
          graphics::Brush br;

          if (depth == 0) {
             return; // Σταματάμε όταν φτάσουμε στο μέγιστο βάθος
          }

          // Υπολογισμός του νέου μεγέθους για την επόμενη αναδρομή
          float newSize = size / 2.0f;

          // Αναδρομικές κλήσεις για κάθε γωνία περιστροφής
          if (rotation == 0) {
             drawHilbertCurve(x, y, newSize, depth - 1, 1);
             graphics::drawLine(x + newSize / 2, y + newSize / 2, x + newSize / 2, y + 3 * newSize / 2, br);
             drawHilbertCurve(x, y + newSize, newSize, depth - 1, 0);
             graphics::drawLine(x + newSize / 2, y + 3 * newSize / 2, x + 3 * newSize / 2, y + 3 * newSize / 2, br);
             drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 0);
             graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + 3 * newSize / 2, y + newSize / 2, br);
             drawHilbertCurve(x + newSize, y, newSize, depth - 1, 3);
          }
          else if (rotation == 1) {
             drawHilbertCurve(x, y, newSize, depth - 1, 0);
             graphics::drawLine(x + newSize / 2, y + newSize / 2, x + 3 * newSize / 2, y + newSize / 2, br);
```

```
            drawHilbertCurve(x + newSize, y, newSize, depth - 1, 1);
            graphics::drawLine(x + 3 * newSize / 2, y + newSize / 2, x + 3 * newSize / 2, y + 3 *
newSize / 2, br);
            drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 1);
            graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + newSize / 2, y + 3 *
newSize / 2, br);
            drawHilbertCurve(x, y + newSize, newSize, depth - 1, 2);
        }
        else if (rotation == 2) {
            drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 3);
            graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + newSize / 2, y + 3 *
newSize / 2, br);
            drawHilbertCurve(x, y + newSize, newSize, depth - 1, 2);
            graphics::drawLine(x + newSize / 2, y + 3 * newSize / 2, x + newSize / 2, y + newSize /
2, br);
            drawHilbertCurve(x, y, newSize, depth - 1, 2);
            graphics::drawLine(x + newSize / 2, y + newSize / 2, x + 3 * newSize / 2, y + newSize /
2, br);
            drawHilbertCurve(x + newSize, y, newSize, depth - 1, 1);
        }
        else if (rotation == 3) {
            drawHilbertCurve(x + newSize, y + newSize, newSize, depth - 1, 2);
            graphics::drawLine(x + 3 * newSize / 2, y + 3 * newSize / 2, x + 3 * newSize / 2, y +
newSize / 2, br);
            drawHilbertCurve(x + newSize, y, newSize, depth - 1, 3);
            graphics::drawLine(x + 3 * newSize / 2, y + newSize / 2, x + newSize / 2, y + newSize /
2, br);
            drawHilbertCurve(x, y, newSize, depth - 1, 3);
            graphics::drawLine(x + newSize / 2, y + newSize / 2, x + newSize / 2, y + 3 * newSize /
2, br);
            drawHilbertCurve(x, y + newSize, newSize, depth - 1, 0);
        }
    }
```

```
    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη `sgg` σε κάθε καρέ
     *
     * @details Αυτή η συνάρτηση ξεκινάει τη σχεδίαση της Hilbert Curve από τη
     * θέση `(startX, startY)` και καλεί τη συνάρτηση `drawHilbertCurve` με τις
     * αρχικές παραμέτρους.
     */
    void draw() {
        graphics::Brush background;
        graphics::setWindowBackground(background);  // Μαύρο φόντο για αντίθεση

        // Αρχικές συντεταγμένες και μέγεθος για την Hilbert Curve
        float startX = 100.0f;    // x-συντεταγμένη εκκίνησης
        float startY = 100.0f;    // y-συντεταγμένη εκκίνησης
        float startLength = 600.0f;  // Μέγεθος της πλευράς του πρώτου τετραγώνου

        // Κλήση της συνάρτησης για σχεδίαση της Hilbert Curve
        drawHilbertCurve(startX, startY, startLength, max_depth, 0);
    }
}
namespace TSquare {
    // Ορισμός των διαστάσεων του παραθύρου
    const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
    const int windowHeight = 600; // Ύψος του παραθύρου σε pixels

    // Ορισμός του μέγιστου βάθους για τη σχεδίαση του fractal
    const int max_depth = 5; // Αύξηση του βάθους δημιουργεί περισσότερες λεπτομέρειες

    /**
     * Συνάρτηση που σχεδιάζει ένα τετράγωνο στο κέντρο μιας δεδομένης θέσης
     * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
     * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
```

```
     * @param size Το μέγεθος της πλευράς του τετραγώνου
     * @param br Το `graphics::Brush` που καθορίζει το χρώμα και το στυλ του τετραγώνου
     */
    void drawSquare(float x, float y, float size, graphics::Brush& br) {
        graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
    }

    /**
     * Αναδρομική συνάρτηση για τη δημιουργία του T-Square Fractal
     * @param x Η x-συντεταγμένη του κεντρικού σημείου του τετραγώνου
     * @param y Η y-συντεταγμένη του κεντρικού σημείου του τετραγώνου
     * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου
     * @param depth Το επίπεδο βάθους της αναδρομής
     *
     * @details Η συνάρτηση καλείται για κάθε τετράγωνο με μειωμένο `size` και βάθος, σχεδιάζοντας
     * νέα τετράγωνα σε κάθε γωνία του προηγούμενου τετραγώνου, σχηματίζοντας τη μορφή του fractal.
     */
    void tSquareFractal(float x, float y, float size, int depth) {
        if (depth == 0) {
            return; // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
        }

        // Ορισμός του χρώματος για το τρέχον τετράγωνο
        graphics::Brush br;
        br.fill_color[0] = 0.0f;                    // Red
        br.fill_color[1] = 0.0f;                    // Green
        br.fill_color[2] = 1.0f - (depth / (float)max_depth); // Μείωση του μπλε με το βάθος

        // Σχεδίαση του κεντρικού τετραγώνου
        drawSquare(x, y, size, br);
```

```
        // Υπολογισμός του μεγέθους των επόμενων τετραγώνων
        float newSize = size / 2.0f;

        // Αναδρομική κλήση για τα τέσσερα τετράγωνα που σχηματίζουν το "T"
        tSquareFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
        tSquareFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
        tSquareFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
        tSquareFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών
     * @details Καθαρίζει το παράθυρο και σχεδιάζει το T-Square Fractal στο κέντρο.
     */
    void draw() {
        graphics::Brush background;

        graphics::setWindowBackground(background);  // Μαύρο φόντο για αντίθεση

        // Αρχικές συντεταγμένες και μέγεθος για το κεντρικό τετράγωνο
        float startX = windowWidth / 2.0f; // Κεντρική x-συντεταγμένη
        float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
        float startSize = 200.0f; // Αρχικό μέγεθος για το πρώτο τετράγωνο

        // Κλήση της συνάρτησης για σχεδίαση του T-Square Fractal
        tSquareFractal(startX, startY, startSize, max_depth);
    }

}
namespace HFractal {
    // Ορισμός διαστάσεων παραθύρου
```

```
const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
const int windowHeight = 600; // Ύψος του παραθύρου σε pixels

// Ορισμός του μέγιστου βάθους αναδρομής για το H-fractal
const int max_depth = 5; // Αύξηση του βάθους δημιουργεί περισσότερες λεπτομέρειες

/**
 * Συνάρτηση που σχεδιάζει το γράμμα "H"
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του "H"
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του "H"
 * @param size Το μήκος της κάθε γραμμής του "H"
 * @param br Το `graphics::Brush` που καθορίζει το χρώμα και το στυλ του "H"
 */
void drawH(float x, float y, float size, graphics::Brush& br) {
    float half = size / 2.0f;

    // Σχεδιάζουμε την οριζόντια γραμμή του "H"
    graphics::drawLine(x - half, y, x + half, y, br);

    // Σχεδιάζουμε τις δύο κάθετες γραμμές του "H"
    graphics::drawLine(x - half, y - half, x - half, y + half, br);
    graphics::drawLine(x + half, y - half, x + half, y + half, br);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του H-fractal
 * @param x Η x-συντεταγμένη του κεντρικού σημείου του "H"
 * @param y Η y-συντεταγμένη του κεντρικού σημείου του "H"
 * @param size Το μήκος της κάθε γραμμής του "H"
 * @param depth Το επίπεδο βάθους της αναδρομής
 *
 * @details Σε κάθε επίπεδο, η συνάρτηση καλείται για τέσσερα νέα "H" που τοποθετούνται
```

```
 * στις τέσσερις άκρες του προηγούμενου, μειώνοντας το μέγεθός τους κατά το ήμισυ.
 */
void hFractal(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
    }

    // Ορισμός του χρώματος για το τρέχον "H"
    graphics::Brush br;
    br.fill_color[0] = 0.0f;                    // Red
    br.fill_color[1] = 0.0f;                    // Green
    br.fill_color[2] = 1.0f - (depth / (float)max_depth);  // Μείωση του μπλε καθώς αυξάνεται το βάθος

    // Σχεδίαση του κεντρικού "H" με τις καθορισμένες παραμέτρους
    drawH(x, y, size, br);

    // Υπολογισμός του νέου μεγέθους για τα επόμενα "H"
    float newSize = size / 2.0f;

    // Αναδρομική κλήση για τα τέσσερα "H" στις γωνίες
    hFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
    hFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
    hFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
    hFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το H-fractal ξεκινώντας από το κέντρο.
 */
void draw() {
```

```
        graphics::Brush background;

        graphics::setWindowBackground(background); // Ορισμός του μαύρου φόντου

        // Αρχικές συντεταγμένες και μέγεθος για το πρώτο κεντρικό "H"
        float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
        float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
        float startSize = 200.0f; // Αρχικό μέγεθος του "H"

        // Κλήση της συνάρτησης για σχεδίαση του H-fractal
        hFractal(startX, startY, startSize, max_depth);
    }
}
namespace VicsekFractal {
    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;  // Πλάτος του παραθύρου σε pixels
    const int windowHeight = 600; // Ύψος του παραθύρου σε pixels

    // Μέγιστο βάθος αναδρομής για το Vicsek Fractal
    const int max_depth = 4; // Αύξηση του βάθους αυξάνει τις λεπτομέρειες του fractal

    /**
     * Συνάρτηση που σχεδιάζει ένα τετράγωνο
     * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
     * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
     * @param size Η πλευρά του τετραγώνου
     * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα και το στυλ του τετραγώνου
     */
    void drawSquare(float x, float y, float size, graphics::Brush& br) {
        graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
    }
```

```
/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Vicsek Fractal
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
 * @param size Η πλευρά του τετραγώνου
 * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
 *
 * @details Η συνάρτηση χωρίζει το τετράγωνο σε πέντε μικρότερα, τοποθετώντας τα τέσσερα στις γωνίες και ένα στο κέντρο.
 * Καλείται αναδρομικά για κάθε υποτετράγωνο, μειώνοντας το `depth` κατά 1 μέχρι να φτάσει στο `0`.
 */
void vicsekFractal(float x, float y, float size, int depth) {
    if (depth == 0) {
        return; // Τερματισμός της αναδρομής στο βάθος 0
    }

    // Ορισμός του χρώματος για το τρέχον τετράγωνο, προσαρμόζοντας τη φωτεινότητα με το βάθος
    graphics::Brush br;
    br.fill_color[0] = 0.2f;                    // Red component
    br.fill_color[1] = 0.4f;                    // Green component
    br.fill_color[2] = 1.0f - (depth / (float)max_depth);  // Blue component μειώνεται όσο αυξάνεται το βάθος

    // Σχεδίαση του κεντρικού τετραγώνου με τις καθορισμένες παραμέτρους
    drawSquare(x, y, size, br);

    // Υπολογισμός του νέου μεγέθους για τα υποτετράγωνα στο επόμενο επίπεδο
    float newSize = size / 3.0f;

    // Αναδρομική κλήση για τα τέσσερα τετράγωνα στις γωνίες και το κεντρικό τετράγωνο
```

```
        vicsekFractal(x - newSize, y - newSize, newSize, depth - 1); // Άνω αριστερά
        vicsekFractal(x + newSize, y - newSize, newSize, depth - 1); // Άνω δεξιά
        vicsekFractal(x, y, newSize, depth - 1);                     // Κεντρικό
        vicsekFractal(x - newSize, y + newSize, newSize, depth - 1); // Κάτω αριστερά
        vicsekFractal(x + newSize, y + newSize, newSize, depth - 1); // Κάτω δεξιά
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
     * @details Καθαρίζει το παράθυρο και σχεδιάζει το Vicsek Fractal ξεκινώντας από το κέντρο.
     */
    void draw() {
        graphics::Brush background;
        graphics::setWindowBackground(background); // Μαύρο φόντο για καλύτερη αντίθεση

        // Αρχικές συντεταγμένες και μέγεθος για το πρώτο κεντρικό τετράγωνο
        float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
        float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
        float startSize = 400.0f;           // Αρχικό μέγεθος του τετραγώνου

        // Κλήση της συνάρτησης για σχεδίαση του Vicsek Fractal
        vicsekFractal(startX, startY, startSize, max_depth);
    }

}
namespace MengerSponge {
    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;  // Πλάτος παραθύρου
    const int windowHeight = 600; // Ύψος παραθύρου

    // Μέγιστο βάθος αναδρομής για το Menger Sponge
    const int max_depth = 6; // Μεγαλύτερες τιμές προσφέρουν περισσότερη λεπτομέρεια
```

```
    /**
     * Συνάρτηση που σχεδιάζει ένα τετράγωνο
     * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
     * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
     * @param size Το μέγεθος της πλευράς του τετραγώνου
     * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα του τετραγώνου
     */
    void drawSquare(float x, float y, float size, graphics::Brush& br) {
        // Σχεδίαση τετραγώνου με κεντραρισμένη τη θέση του (x, y)
        graphics::drawRect(x - size / 2, y - size / 2, size, size, br);
    }

    /**
     * Αναδρομική συνάρτηση για τη σχεδίαση του Menger Sponge σε 2D
     * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου
     * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου
     * @param size Το μέγεθος της πλευράς του τετραγώνου
     * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
     *
     * @details Η συνάρτηση τοποθετεί ένα τετράγωνο στο κέντρο και δημιουργεί αναδρομικά οκτώ μικρότερα
     * τετράγωνα γύρω από αυτό, παραλείποντας το κεντρικό τετράγωνο. Η διαδικασία επαναλαμβάνεται
     * μέχρι το επίπεδο `depth` να φτάσει στο μηδέν.
     */
    void mengerSponge(float x, float y, float size, int depth) {
        if (depth == 0) {
            return; // Τερματισμός αναδρομής όταν φτάσουμε στο βάθος 0
        }

        // Ορισμός χρώματος για το τρέχον τετράγωνο, με την απόχρωση να εξαρτάται από το βάθος
```

```
    graphics::Brush br;
    br.fill_color[0] = 0.1f + (depth / (float)max_depth) * 0.5f;  // Red component
    br.fill_color[1] = 0.1f;                                      // Green component
    br.fill_color[2] = 0.6f - (depth / (float)max_depth) * 0.5f;  // Blue component

    // Σχεδίαση του κεντρικού τετραγώνου
    drawSquare(x, y, size, br);

    // Υπολογισμός νέου μεγέθους για τα επόμενα μικρότερα τετράγωνα
    float newSize = size / 3.0f;

    // Αναδρομική κλήση για τα οκτώ τετράγωνα που περιβάλλουν το κεντρικό
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            // Παραλείπουμε το κεντρικό τετράγωνο (dx == 0 && dy == 0)
            if (dx != 0 || dy != 0) {
                mengerSponge(x + dx * newSize, y + dy * newSize, newSize, depth - 1);
            }
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθαρίζει το παράθυρο και σχεδιάζει το Menger Sponge ξεκινώντας από το κέντρο
 */
void draw() {
    graphics::Brush background;

    graphics::setWindowBackground(background); // Μαύρο φόντο για αντίθεση
```

```
        // Αρχικές συντεταγμένες και μέγεθος για το κεντρικό τετράγωνο
        float startX = windowWidth / 2.0f;  // Κεντρική x-συντεταγμένη
        float startY = windowHeight / 2.0f; // Κεντρική y-συντεταγμένη
        float startSize = 400.0f;           // Αρχικό μέγεθος του τετραγώνου

        // Κλήση της συνάρτησης για σχεδίαση του Menger Sponge
        mengerSponge(startX, startY, startSize, max_depth);
    }
}
namespace Hexaflake {
    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;
    const int windowHeight = 600;

    // Μέγιστο βάθος αναδρομής για το Hexaflake
    const int max_depth = 4;

    /**
     * Συνάρτηση για τον υπολογισμό των κορυφών ενός εξαγώνου
     * @param x Το x-κέντρο του εξαγώνου
     * @param y Το y-κέντρο του εξαγώνου
     * @param radius Η ακτίνα του εξαγώνου
     * @param vertices Πίνακας για αποθήκευση των συντεταγμένων των κορυφών
     * @details Υπολογίζει τις συντεταγμένες των κορυφών του εξαγώνου με βάση το κέντρο και την ακτίνα.
     */
    void getHexagonVertices(float x, float y, float radius, float vertices[12]) {
        for (int i = 0; i < 6; i++) {
            vertices[2 * i] = x + radius * cos(M_PI / 3 * i);         // Συντεταγμένη x για κάθε κορυφή
            vertices[2 * i + 1] = y + radius * sin(M_PI / 3 * i);     // Συντεταγμένη y για κάθε κορυφή
        }
    }
```

```
/**
 * Συνάρτηση για σχεδίαση ενός πολύγωνου με τη χρήση της graphics::drawLine
 * @param vertices Πίνακας συντεταγμένων των κορυφών του πολύγωνου
 * @param vertex_count Ο αριθμός κορυφών του πολύγωνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα
 * @details Σχεδιάζει ένα πολύγωνο συνδέοντας τις κορυφές μεταξύ τους.
 */
void drawPolygon(float vertices[], int vertex_count, graphics::Brush& br) {
    for (int i = 0; i < vertex_count; i++) {
        int next_index = (i + 1) % vertex_count;
        graphics::drawLine(vertices[2 * i], vertices[2 * i + 1],
            vertices[2 * next_index], vertices[2 * next_index + 1], br);
    }
}

/**
 * Συνάρτηση που σχεδιάζει ένα εξάγωνο
 * @param x Το x-κέντρο του εξαγώνου
 * @param y Το y-κέντρο του εξαγώνου
 * @param radius Η ακτίνα του εξαγώνου
 * @param br Ένα αντικείμενο `graphics::Brush` που καθορίζει το χρώμα
 * @details Χρησιμοποιεί τις συντεταγμένες από την `getHexagonVertices` για να σχεδιάσει το εξάγωνο.
 */
void drawHexagon(float x, float y, float radius, graphics::Brush& br) {
    float vertices[12];
    getHexagonVertices(x, y, radius, vertices);
    drawPolygon(vertices, 6, br); // Σχεδίαση του εξαγώνου χρησιμοποιώντας την `drawPolygon`
}
```

```
/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Hexaflake
 * @param x Το x-κέντρο του εξαγώνου
 * @param y Το y-κέντρο του εξαγώνου
 * @param radius Η ακτίνα του εξαγώνου
 * @param depth Το τρέχον επίπεδο βάθους της αναδρομής
 * @details Καλεί αναδρομικά για τη σχεδίαση των έξι μικρότερων εξαγώνων γύρω από το κεντρικό εξάγωνο.
 */
void hexaflake(float x, float y, float radius, int depth) {
    if (depth == 0) {
        return;
    }

    // Ορισμός χρώματος για το εξάγωνο, με απόχρωση που εξαρτάται από το βάθος
    graphics::Brush br;
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.6f;
    br.fill_color[1] = 0.2f;
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.6f;

    // Σχεδίαση του κεντρικού εξαγώνου
    drawHexagon(x, y, radius, br);

    // Υπολογισμός της ακτίνας για τα μικρότερα εξάγωνα
    float newRadius = radius / 3.0f;

    // Αναδρομική κλήση για τα έξι εξάγωνα γύρω από το κεντρικό
    for (int i = 0; i < 6; i++) {
        float newX = x + 2 * newRadius * cos(M_PI / 3 * i);
        float newY = y + 2 * newRadius * sin(M_PI / 3 * i);
        hexaflake(newX, newY, newRadius, depth - 1);
    }
```

```
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
     * @details Ρυθμίζει το φόντο και εκκινεί τη σχεδίαση του Hexaflake στο κέντρο του παραθύρου.
     */
    void draw() {
        graphics::Brush background;

        graphics::setWindowBackground(background); // Μαύρο φόντο για αντίθεση

        // Αρχικές συντεταγμένες και ακτίνα για το κεντρικό εξάγωνο
        float startX = windowWidth / 2.0f;
        float startY = windowHeight / 2.0f;
        float startRadius = 200.0f;

        // Κλήση της συνάρτησης για σχεδίαση του Hexaflake
        hexaflake(startX, startY, startRadius, max_depth);
    }

}
namespace GosperCurve {

    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;
    const int windowHeight = 600;

    // Ορισμός του μέγιστου βάθους για τη Gosper Curve
    const int max_depth = 4;

    // Μήκος γραμμής για την αρχική Gosper Curve
```

```
    float lineLength = 10.0f;


    /**
     * Συνάρτηση που μεταφράζει τις εντολές της Gosper Curve σε οδηγίες για σχεδίαση
     * @param x Τρέχουσα συντεταγμένη x
     * @param y Τρέχουσα συντεταγμένη y
     * @param angle Η γωνία σχεδίασης
     * @param length Το μήκος της γραμμής
     * @param instructions Το μοτίβο των εντολών της καμπύλης
     * @details Με βάση τις εντολές στο string `instructions`, η συνάρτηση αυτή σχεδιάζει γραμμές ή περιστρέφει τη γωνία σχεδίασης.
     */
    void drawGosperCurve(float& x, float& y, float angle, float length, const std::string& instructions) {
        graphics::Brush br;
        br.fill_color[0] = 0.0f;  // Χρώμα (κόκκινο)
        br.fill_color[1] = 0.0f;
        br.fill_color[2] = 1.0f;

        // Ερμηνεία και σχεδίαση με βάση τις εντολές στο `instructions`
        for (char command : instructions) {
            if (command == 'F') {
                float newX = x + length * cos(angle);
                float newY = y + length * sin(angle);
                graphics::drawLine(x, y, newX, newY, br);

                // Ενημέρωση της θέσης της γραμμής
                x = newX;
                y = newY;
            }
            else if (command == '+') {
                angle -= M_PI / 3; // Περιστροφή -60 μοίρες
```

```
            }
            else if (command == '-') {
                angle += M_PI / 3; // Περιστροφή +60 μοίρες
            }
        }
    }

    /**
     * Αναδρομική συνάρτηση για τη δημιουργία του μοτίβου της Gosper Curve
     * @param depth Το βάθος της αναδρομής
     * @param isA Αν αληθές, χρησιμοποιεί τον κανόνα Α. Διαφορετικά, χρησιμοποιεί τον κανόνα Β.
     * @return Ένα string με το μοτίβο των εντολών για τη Gosper Curve στο τρέχον βάθος
     * @details Κάθε βαθμίδα αναδρομής χρησιμοποιεί τους κανόνες `aRule` και `bRule` για τη δημιουργία του μοτίβου της καμπύλης.
     */
    std::string generateGosperCurve(int depth, bool isA = true) {
        if (depth == 0) {
            return isA ? "F" : "F";
        }

        // Ορισμός των κανόνων Α και Β
        std::string aRule = "A-F--B+AF+A+FA--F-BF-AF+A+FA+FB--";
        std::string bRule = "B+AF--BFA+FBF--F-AF+BFA+FBF--F-";

        std::string result;
        std::string rule = isA ? aRule : bRule;

        for (char c : rule) {
            if (c == 'A') {
                result += generateGosperCurve(depth - 1, true);
            }
```

```
            else if (c == 'B') {
                result += generateGosperCurve(depth - 1, false);
            }
            else {
                result += c;
            }
        }
        return result;
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
     * @details Θέτει τις αρχικές συντεταγμένες και τη γωνία, δημιουργεί τις οδηγίες της Gosper Curve και εκκινεί τη διαδικασία σχεδίασης.
     */
    void draw() {
        graphics::Brush background;

        graphics::setWindowBackground(background);

        // Αρχικές συντεταγμένες και γωνία για την καμπύλη
        float startX = 200.0f;
        float startY = 400.0f;
        float startAngle = 0.0f;

        // Δημιουργία των οδηγιών για τη Gosper Curve
        std::string gosperInstructions = generateGosperCurve(max_depth);

        // Σχεδίαση της Gosper Curve
        drawGosperCurve(startX, startY, startAngle, lineLength, gosperInstructions);
    }
```

```
}
namespace LevyCCurve {
    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;
    const int windowHeight = 600;

    // Ορισμός του μέγιστου βάθους για τη Levy C Curve
    const int max_depth = 12;

    /**
     * Αναδρομική συνάρτηση για σχεδίαση της Levy C Curve
     * @param x1 Συντεταγμένη x του αρχικού σημείου
     * @param y1 Συντεταγμένη y του αρχικού σημείου
     * @param x2 Συντεταγμένη x του τελικού σημείου
     * @param y2 Συντεταγμένη y του τελικού σημείου
     * @param depth Βάθος αναδρομής
     * @details Αν το βάθος είναι 0, σχεδιάζει μια ευθεία γραμμή μεταξύ των δύο σημείων. Σε μεγαλύτερο βάθος, υπολογίζει το μεσαίο σημείο με περιστροφή 45 μοιρών και καλεί αναδρομικά την ίδια συνάρτηση για κάθε τμήμα.
     */
    void drawLevyCurve(float x1, float y1, float x2, float y2, int depth) {
        // Όταν φτάσουμε στο βάθος 0, σχεδιάζουμε μια ευθεία γραμμή μεταξύ των σημείων
        if (depth == 0) {
            graphics::Brush br;
            br.fill_color[0] = 0.0f;
            br.fill_color[1] = 0.0f;
            br.fill_color[2] = 1.0f; // Μπλε χρώμα για τη γραμμή
            graphics::drawLine(x1, y1, x2, y2, br);
        }
        else {
            // Υπολογισμός του μεσαίου σημείου με γωνία 45 μοιρών
            float mid_x = (x1 + x2) / 2 + (y2 - y1) / 2 * std::sqrt(2) / 2;
```

```
            float mid_y = (y1 + y2) / 2 - (x2 - x1) / 2 * std::sqrt(2) / 2;

            // Αναδρομική κλήση για το πρώτο και το δεύτερο τμήμα της καμπύλης
            drawLevyCurve(x1, y1, mid_x, mid_y, depth - 1);
            drawLevyCurve(mid_x, mid_y, x2, y2, depth - 1);
        }
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
     * @details Ορίζει το φόντο του παραθύρου και σχεδιάζει την Levy C Curve, ξεκινώντας από προκαθορισμένες αρχικές και τελικές συντεταγμένες.
     */
    void draw() {
        graphics::Brush background;

        graphics::setWindowBackground(background);

        // Αρχικές συντεταγμένες για τη Levy C Curve
        float startX = 300.0f;
        float startY = 400.0f;
        float endX = 500.0f;
        float endY = 400.0f;

        // Κλήση της συνάρτησης για τη σχεδίαση της Levy C Curve
        drawLevyCurve(startX, startY, endX, endY, max_depth);
    }

}
namespace Pentaflake {
    // Ορισμός διαστάσεων παραθύρου
    const int windowWidth = 600;
```

```
const int windowHeight = 600;

// Μέγιστο βάθος αναδρομής για το Pentaflake
const int max_depth = 4;

/**
 * Συνάρτηση για τον υπολογισμό των κορυφών ενός πενταγώνου
 * @param x Κέντρο του πενταγώνου στον άξονα x
 * @param y Κέντρο του πενταγώνου στον άξονα y
 * @param radius Ακτίνα του πενταγώνου
 * @param vertices Πίνακας που θα αποθηκεύσει τις συντεταγμένες των κορυφών
 */
void getPentagonVertices(float x, float y, float radius, float vertices[10]) {
    for (int i = 0; i < 5; i++) {
        vertices[2 * i] = x + radius * cos(2 * M_PI * i / 5);         // Συντεταγμένη x για κάθε κορυφή
        vertices[2 * i + 1] = y + radius * sin(2 * M_PI * i / 5);     // Συντεταγμένη y για κάθε κορυφή
    }
}

/**
 * Συνάρτηση για σχεδίαση ενός πολύγωνου χρησιμοποιώντας την graphics::drawLine
 * @param vertices Πίνακας με τις συντεταγμένες των κορυφών
 * @param vertex_count Αριθμός κορυφών του πολυγώνου
 * @param br Αντικείμενο γραφής για την εμφάνιση χρώματος
 */
void drawPolygon(float vertices[], int vertex_count, graphics::Brush& br) {
    for (int i = 0; i < vertex_count; i++) {
        int next_index = (i + 1) % vertex_count;  // Σύνδεση της τρέχουσας κορυφής με την επόμενη
        graphics::drawLine(vertices[2 * i], vertices[2 * i + 1],
            vertices[2 * next_index], vertices[2 * next_index + 1], br);
```

```
    }
}

/**
 * Συνάρτηση που σχεδιάζει ένα πεντάγωνο
 * @param x Συντεταγμένη x του κέντρου του πενταγώνου
 * @param y Συντεταγμένη y του κέντρου του πενταγώνου
 * @param radius Ακτίνα του πενταγώνου
 * @param br Αντικείμενο γραφής για την εμφάνιση χρώματος
 */
void drawPentagon(float x, float y, float radius, graphics::Brush& br) {
    float vertices[10];  // Πίνακας για αποθήκευση συντεταγμένων των κορυφών
    getPentagonVertices(x, y, radius, vertices);  // Υπολογισμός συντεταγμένων των κορυφών
    drawPolygon(vertices, 5, br); // Σχεδίαση του πενταγώνου
}

/**
 * Αναδρομική συνάρτηση για τη σχεδίαση του Pentaflake
 * @param x Συντεταγμένη x του κέντρου του πενταγώνου
 * @param y Συντεταγμένη y του κέντρου του πενταγώνου
 * @param radius Ακτίνα του πενταγώνου
 * @param depth Τρέχον βάθος αναδρομής
 */
void pentaflake(float x, float y, float radius, int depth) {
    if (depth == 0) {
        return; // Σταματάμε αν φτάσουμε στο μέγιστο βάθος
    }

    // Ορισμός χρώματος για το πεντάγωνο με βάση το βάθος αναδρομής
    graphics::Brush br;
    br.fill_color[0] = 0.2f + (depth / (float)max_depth) * 0.6f;
    br.fill_color[1] = 0.2f;
```

```
    br.fill_color[2] = 1.0f - (depth / (float)max_depth) * 0.6f;

    // Σχεδίαση του κεντρικού πενταγώνου
    drawPentagon(x, y, radius, br);

    // Υπολογισμός της ακτίνας για τα μικρότερα πεντάγωνα
    float newRadius = radius / 3.0f;

    // Αναδρομική κλήση για τα πέντε πεντάγωνα γύρω από το κεντρικό
    for (int i = 0; i < 5; i++) {
        float newX = x + 2 * newRadius * cos(2 * M_PI * i / 5);  // Νέα x θέση
        float newY = y + 2 * newRadius * sin(2 * M_PI * i / 5);  // Νέα y θέση
        pentaflake(newX, newY, newRadius, depth - 1);  // Αναδρομική κλήση
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
 * @details Καθορίζει το φόντο του παραθύρου και καλεί τη συνάρτηση pentaflake για να ξεκινήσει η σχεδίαση του fractal από το κέντρο.
 */
void draw() {
    graphics::Brush background;
    graphics::setWindowBackground(background);

    // Έναρξη σχεδίασης του Pentaflake από το κέντρο του παραθύρου
    float startX = windowWidth / 2.0f;
    float startY = windowHeight / 2.0f;
    float startRadius = 200.0f; // Αρχική ακτίνα του πενταγώνου

    // Κλήση της συνάρτησης για σχεδίαση του Pentaflake
    pentaflake(startX, startY, startRadius, max_depth);
```

```
        }

    }
    namespace GoldenDragon{
        // Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 1000;
const int WINDOW_HEIGHT = 600;

// Παράμετροι για το fractal
const int MAX_DEPTH = 18;         // Μέγιστο βάθος αναδρομής
const float SCALE_FACTOR = 0.78f;  // Παράγοντας κλίμακας για κάθε νέο τμήμα
const float ANGLE_OFFSET = 45.0f;  // Γωνία περιστροφής σε κάθε στάδιο (μοίρες)

/**
 * Συνάρτηση για την αναδρομική σχεδίαση του Golden Dragon Fractal
 * @param x Συντεταγμένη x του σημείου εκκίνησης
 * @param y Συντεταγμένη y του σημείου εκκίνησης
 * @param length Μήκος του τρέχοντος τμήματος της γραμμής
 * @param angle Γωνία σχεδίασης σε μοίρες
 * @param depth Τρέχον βάθος αναδρομής
 *
 * Αυτή η συνάρτηση σχεδιάζει το fractal Golden Dragon χρησιμοποιώντας αναδρομή.
 * Σε κάθε στάδιο, υπολογίζει τις συντεταγμένες του επόμενου σημείου και σχεδιάζει
 * γραμμές με διαφορετικές γωνίες περιστροφής.
 */
void drawGoldenDragon(float x, float y, float length, float angle, int depth) {
    // Τερματισμός της αναδρομής όταν φτάσουμε στο βάθος 0
    if (depth == 0) return;

    // Υπολογισμός τελικών συντεταγμένων για το τρέχον τμήμα
    float xEnd = x + length * cos(angle * M_PI / 180.0f);
    float yEnd = y + length * sin(angle * M_PI / 180.0f);
```

```
    // Ρύθμιση του πινέλου για τη σχεδίαση με σταδιακή αλλαγή χρώματος
    graphics::Brush brush;
    brush.fill_color[0] = 0.3f + depth * 0.04f; // Αυξάνει τον τόνο του κόκκινου όσο αυξάνεται το βάθος
    brush.fill_color[1] = 0.2f;                 // Πράσινος τόνος
    brush.fill_color[2] = 0.5f + depth * 0.02f; // Αυξάνει τον τόνο του μπλε όσο αυξάνεται το βάθος

    // Σχεδίαση γραμμής από το σημείο (x, y) στο (xEnd, yEnd)
    graphics::drawLine(x, y, xEnd, yEnd, brush);

    // Αναδρομική κλήση για σχεδίαση των επόμενων τμημάτων
    // Περιστροφή κατά -ANGLE_OFFSET για το πρώτο τμήμα και +ANGLE_OFFSET για το δεύτερο
    drawGoldenDragon(xEnd, yEnd, length * SCALE_FACTOR, angle - ANGLE_OFFSET, depth - 1);
    drawGoldenDragon(xEnd, yEnd, length * SCALE_FACTOR, angle + ANGLE_OFFSET, depth - 1);
}

/**
 * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ
 * @details Καθορίζει το φόντο και καλεί τη συνάρτηση σχεδίασης του fractal.
 */
void draw() {
    // Ρύθμιση του φόντου σε μαύρο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f; // Κόκκινο συστατικό
    bg.fill_color[1] = 0.0f; // Πράσινο συστατικό
    bg.fill_color[2] = 0.0f; // Μπλε συστατικό
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);
```

```
    // Κλήση της συνάρτησης για σχεδίαση του Golden Dragon Fractal από το αριστερό κέντρο του παραθύρου
    drawGoldenDragon(WINDOW_WIDTH / 10, WINDOW_HEIGHT / 2, 200.0f, 0.0f, MAX_DEPTH);
}


    }
    namespace ButterflyFractal {
        // Ορισμός διαστάσεων παραθύρου
        const int WINDOW_WIDTH = 800;
        const int WINDOW_HEIGHT = 600;

        // Παράμετροι για το fractal
        const int MAX_DEPTH = 20;         // Βάθος αναδρομής
        const float SCALE_FACTOR = 0.5f;  // Παράγοντας κλίμακας για κάθε νέο τμήμα
        const float ANGLE_OFFSET = 45.0f; // Γωνία περιστροφής για κάθε στάδιο

        /**
         * Συνάρτηση για την αναδρομική σχεδίαση του Butterfly Fractal
         * @param x Συντεταγμένη x του σημείου εκκίνησης
         * @param y Συντεταγμένη y του σημείου εκκίνησης
         * @param length Μήκος του τρέχοντος τμήματος
         * @param angle Γωνία περιστροφής σε μοίρες για το τρέχον τμήμα
         * @param depth Τρέχον βάθος αναδρομής
         *
         * Αυτή η συνάρτηση σχεδιάζει γραμμές που διακλαδίζονται αναδρομικά,
         * δημιουργώντας ένα σχήμα που μοιάζει με πεταλούδα.
         */
        void drawButterflyFractal(float x, float y, float length, float angle, int depth) {
            // Όταν το βάθος φτάσει στο 0, σταματά η αναδρομή
            if (depth == 0) return;
```

```
    // Υπολογισμός των τελικών συντεταγμένων για τις δύο διακλαδώσεις
    float xEnd1 = x + length * cos(angle * M_PI / 180.0f);
    float yEnd1 = y + length * sin(angle * M_PI / 180.0f);
    float xEnd2 = x + length * cos((angle + 180) * M_PI / 180.0f);
    float yEnd2 = y + length * sin((angle + 180) * M_PI / 180.0f);

    // Ρύθμιση του πινέλου σχεδίασης με χρώμα που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.5f + depth * 0.05f;  // Αυξάνει τον κόκκινο τόνο για κάθε επίπεδο
    brush.fill_color[1] = 0.3f;                  // Σταθερή απόχρωση πράσινου
    brush.fill_color[2] = 0.7f - depth * 0.05f;  // Μειώνει τον μπλε τόνο για κάθε επίπεδο

    // Σχεδίαση των δύο γραμμών που αποτελούν τις διακλαδώσεις του fractal
    graphics::drawLine(x, y, xEnd1, yEnd1, brush);
    graphics::drawLine(x, y, xEnd2, yEnd2, brush);

    // Αναδρομική κλήση για τις δύο γραμμές του επόμενου επιπέδου
    // Το μήκος μειώνεται και η γωνία περιστρέφεται κατά ±ANGLE_OFFSET
    drawButterflyFractal(xEnd1, yEnd1, length * SCALE_FACTOR, angle - ANGLE_OFFSET, depth - 1);
    drawButterflyFractal(xEnd2, yEnd2, length * SCALE_FACTOR, angle + ANGLE_OFFSET, depth - 1);
}

/**
 * Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ
 * @details Καθαρίζει το φόντο και καλεί τη συνάρτηση σχεδίασης του fractal.
 */
void draw() {
    // Ρύθμιση του φόντου σε μαύρο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;  // Κόκκινο συστατικό
```

```
        bg.fill_color[1] = 0.0f;  // Πράσινο συστατικό
        bg.fill_color[2] = 0.0f;  // Μπλε συστατικό
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης σχεδίασης του fractal από το κέντρο του παραθύρου
        drawButterflyFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200.0f, 0.0f, MAX_DEPTH);
    }

}
namespace PlasmaFractal {
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <vector>

    // Ορισμός διαστάσεων παραθύρου
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 600;

    // Διαστάσεις και παράμετροι πλέγματος
    const int GRID_SIZE = 129;      // (2^n) + 1, απαραίτητο για Diamond-Square
    float roughness = 0.5f;         // Παράγοντας τραχύτητας για τον έλεγχο του fractal

    // Πλέγμα για την αποθήκευση τιμών υψομέτρου
    std::vector<std::vector<float>> grid(GRID_SIZE, std::vector<float>(GRID_SIZE));

    /**
     * Αρχικοποίηση του πλέγματος με τυχαίες αρχικές τιμές στις γωνίες.
     * Οι τιμές αυτές είναι μεταξύ 0 και 1.
     */
```

```
    void initializeGrid() {
        grid[0][0] = static_cast<float>(rand()) / RAND_MAX;
        grid[0][GRID_SIZE - 1] = static_cast<float>(rand()) / RAND_MAX;
        grid[GRID_SIZE - 1][0] = static_cast<float>(rand()) / RAND_MAX;
        grid[GRID_SIZE - 1][GRID_SIZE - 1] = static_cast<float>(rand()) / RAND_MAX;
    }

    /**
     * Υπολογίζει τη μέση τιμή τεσσάρων σημείων για χρήση στον αλγόριθμο Diamond-Square.
     * @param a, b, c, d: Τιμές των τεσσάρων σημείων γύρω από ένα σημείο πλέγματος.
     * @return Μέση τιμή των σημείων a, b, c, d.
     */
    float average(float a, float b, float c, float d) {
        return (a + b + c + d) / 4.0f;
    }

    /**
     * Αλγόριθμος Diamond-Square που παράγει το fractal ύψος για το πλέγμα.
     * @param size: Τρέχον μέγεθος του τμήματος του πλέγματος που διαιρείται.
     *
     * Η συνάρτηση εκτελεί τον αλγόριθμο Diamond-Square. Σε κάθε βήμα, υπολογίζει νέα
σημεία
     * χρησιμοποιώντας τον παράγοντα τραχύτητας για την προσθήκη τυχαιότητας.
     */
    void diamondSquare(int size) {
        int half = size / 2;
        if (half < 1) return;

        // Diamond step: Εύρεση μέσης τιμής και προσθήκη τυχαιότητας
        for (int y = half; y < GRID_SIZE - 1; y += size) {
            for (int x = half; x < GRID_SIZE - 1; x += size) {
                float avg = average(grid[x - half][y - half], grid[x + half][y - half],
```

```
                grid[x - half][y + half], grid[x + half][y + half]);
            grid[x][y] = avg + ((static_cast<float>(rand()) / RAND_MAX) * 2 - 1) * roughness;
        }
    }

    // Square step: Υπολογισμός για κάθε κελί γύρω από τα διαμάντια
    for (int y = 0; y < GRID_SIZE; y += half) {
        for (int x = (y + half) % size; x < GRID_SIZE; x += size) {
            float avg = average(grid[(x - half + GRID_SIZE) % GRID_SIZE][y],
                grid[(x + half) % GRID_SIZE][y],
                grid[x][(y + half) % GRID_SIZE],
                grid[x][(y - half + GRID_SIZE) % GRID_SIZE]);
            grid[x][y] = avg + ((static_cast<float>(rand()) / RAND_MAX) * 2 - 1) * roughness;
        }
    }

    // Αναδρομική κλήση για το επόμενο επίπεδο
    diamondSquare(size / 2);
}

/**
 * Σχεδιάζει το fractal χρωματίζοντας το πλέγμα σύμφωνα με τις τιμές υψομέτρου.
 * Το ύψος κάθε κελιού κανονικοποιείται για να παράγει χρώμα.
 */
void drawPlasmaFractal() {
    graphics::Brush brush;

    for (int y = 0; y < GRID_SIZE - 1; y++) {
        for (int x = 0; x < GRID_SIZE - 1; x++) {
            // Κανονικοποίηση της τιμής ύψους για χρωματική απεικόνιση
            float colorValue = (grid[x][y] + 1) / 2.0f;
            brush.fill_color[0] = colorValue * 0.5f + 0.5f; // Κόκκινη απόχρωση
```

```
            brush.fill_color[1] = colorValue * 0.7f;        // Πράσινη απόχρωση
            brush.fill_color[2] = colorValue * 1.0f;        // Μπλε απόχρωση
            float cellSize = static_cast<float>(WINDOW_WIDTH) / GRID_SIZE;
            graphics::drawRect(x * cellSize, y * cellSize, cellSize, cellSize, brush);
        }
    }
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη για να σχεδιάσει το fractal.
 */
void draw() {
    drawPlasmaFractal();
}

}
namespace OctahedronFractal {
    // Ορισμός διαστάσεων παραθύρου
    const int WINDOW_WIDTH = 1000;
    const int WINDOW_HEIGHT = 600;
    const int MAX_DEPTH = 10;  // Μέγιστο βάθος αναδρομής για το fractal

    /**
     * Συνάρτηση που σχεδιάζει ένα τρίγωνο
     * @param x1, y1: Συντεταγμένες της πρώτης κορυφής
     * @param x2, y2: Συντεταγμένες της δεύτερης κορυφής
     * @param x3, y3: Συντεταγμένες της τρίτης κορυφής
     * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
     *
     * Η συνάρτηση αυτή χρησιμοποιείται για τη σχεδίαση των πλευρών ενός τριγώνου
     * με τη βοήθεια των τριών γραμμών που συνδέουν τα σημεία (x1, y1), (x2, y2) και (x3, y3).
     */
```

```
    void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, const graphics::Brush& brush) {
        graphics::drawLine(x1, y1, x2, y2, brush);
        graphics::drawLine(x2, y2, x3, y3, brush);
        graphics::drawLine(x3, y3, x1, y1, brush);
    }

    /**
     * Αναδρομική συνάρτηση για τη δημιουργία του fractal
     * @param x, y: Συντεταγμένες του κεντρικού σημείου του τριγώνου
     * @param size: Μέγεθος του τρέχοντος τριγώνου
     * @param depth: Βάθος της αναδρομής, που καθορίζει το επίπεδο λεπτομέρειας
     * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
     *
     * Η συνάρτηση καλείται αναδρομικά για να δημιουργήσει τέσσερα μικρότερα τρίγωνα,
     * που τοποθετούνται σε τέσσερις διαφορετικές θέσεις γύρω από το κεντρικό τρίγωνο.
     * Όταν φτάσει στο βάθος 0, σχεδιάζει το βασικό τρίγωνο.
     */
    void drawOctahedronFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
        if (depth == 0) {
            // Σχεδίαση ενός κεντρικού τρίγωνου για το αρχικό οκτάεδρο
            drawTriangle(x, y - size, x - size, y + size, x + size, y + size, brush);
            return;
        }

        // Υπολογισμός νέου μεγέθους για τα μικρότερα τρίγωνα
        float newSize = size / 2.0f;

        // Αναδρομική κλήση για τη σχεδίαση τεσσάρων μικρότερων τριγώνων
        drawOctahedronFractal(x, y - size, newSize, depth - 1, brush);        // Πάνω τρίγωνο
        drawOctahedronFractal(x - size, y + size, newSize, depth - 1, brush); // Κάτω αριστερό τρίγωνο
```

```
        drawOctahedronFractal(x + size, y + size, newSize, depth - 1, brush); // Κάτω δεξιό τρίγωνο
        drawOctahedronFractal(x, y + size * 2, newSize, depth - 1, brush);    // Κεντρικό κάτω τρίγωνο
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη
     *
     * Αρχικά καθαρίζει το παράθυρο σχεδίασης και στη συνέχεια καλεί τη συνάρτηση
     * `drawOctahedronFractal` για να σχεδιάσει το fractal στο κέντρο του παραθύρου.
     */
    void draw() {
        // Ορισμός μαύρου φόντου για αντίθεση
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Ορισμός του πινέλου για το fractal με απόχρωση μπλε-πράσινου
        graphics::Brush brush;
        brush.fill_color[0] = 0.0f;
        brush.fill_color[1] = 0.5f;
        brush.fill_color[2] = 1.0f;

        // Κλήση της συνάρτησης για σχεδίαση του fractal
        drawOctahedronFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 4, WINDOW_HEIGHT / 4, MAX_DEPTH, brush);
    }
}
namespace SnowflakeCurve {
    // Ορισμός διαστάσεων παραθύρου
```

```
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5;  // Μέγιστο βάθος αναδρομής για το fractal

/**
 * Συνάρτηση για σχεδίαση ενός τμήματος της καμπύλης Koch.
 * @param x1, y1: Συντεταγμένες αρχής της γραμμής
 * @param x2, y2: Συντεταγμένες τέλους της γραμμής
 * @param depth: Βάθος αναδρομής που καθορίζει το επίπεδο λεπτομέρειας
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Στην αναδρομή, το τμήμα διαιρείται σε τρία ίσα μέρη και σχηματίζεται
 * ισόπλευρο τρίγωνο με κορυφή εκτός της αρχικής ευθείας. Όταν depth = 0, σχεδιάζεται ευθεία γραμμή.
 */
void drawKochSegment(float x1, float y1, float x2, float y2, int depth, const graphics::Brush& brush) {
    if (depth == 0) {
        // Σχεδίαση βασικής γραμμής
        graphics::drawLine(x1, y1, x2, y2, brush);
    }
    else {
        // Υπολογισμός διαφοράς x και y μεταξύ των άκρων του τμήματος
        float dx = x2 - x1;
        float dy = y2 - y1;
        float dist = std::sqrt(dx * dx + dy * dy) / 3.0f;  // Απόσταση για κάθε τμήμα

        float angle = std::atan2(dy, dx);  // Γωνία του τμήματος

        // Υπολογισμός σημείων διαίρεσης
        float xA = x1 + dx / 3;
        float yA = y1 + dy / 3;
```

```
        float xB = x1 + 2 * dx / 3;
        float yB = y1 + 2 * dy / 3;

        // Υπολογισμός κορυφής του ισόπλευρου τριγώνου
        float xC = xA + dist * std::cos(angle - M_PI / 3);
        float yC = yA + dist * std::sin(angle - M_PI / 3);

        // Αναδρομή για σχεδίαση των τεσσάρων επιμέρους τμημάτων
        drawKochSegment(x1, y1, xA, yA, depth - 1, brush);
        drawKochSegment(xA, yA, xC, yC, depth - 1, brush);
        drawKochSegment(xC, yC, xB, yB, depth - 1, brush);
        drawKochSegment(xB, yB, x2, y2, depth - 1, brush);
    }
}

/**
 * Συνάρτηση που σχεδιάζει τη βασική χιονονιφάδα ως ισόπλευρο τρίγωνο.
 * @param x, y: Κέντρο του τριγώνου
 * @param sideLength: Μήκος κάθε πλευράς του τριγώνου
 * @param depth: Βάθος της καμπύλης Koch για κάθε πλευρά
 * @param brush: Αντικείμενο graphics::Brush για το χρώμα σχεδίασης
 *
 * Υπολογίζει τις κορυφές του ισόπλευρου τριγώνου και καλεί την drawKochSegment
 * για κάθε πλευρά με τον ίδιο βαθμό αναδρομής.
 */
void drawSnowflake(float x, float y, float sideLength, int depth, const graphics::Brush& brush)
{
    // Υπολογισμός ύψους ισόπλευρου τριγώνου
    float height = sideLength * std::sqrt(3) / 2;

    // Υπολογισμός συντεταγμένων κορυφών του τριγώνου
```

```
    float x1 = x - sideLength / 2;
    float y1 = y + height / 3;

    float x2 = x + sideLength / 2;
    float y2 = y + height / 3;

    float x3 = x;
    float y3 = y - 2 * height / 3;

    // Σχεδίαση τριών πλευρών του τρίγωνου με χρήση της καμπύλης Koch
    drawKochSegment(x1, y1, x2, y2, depth, brush);
    drawKochSegment(x2, y2, x3, y3, depth, brush);
    drawKochSegment(x3, y3, x1, y1, depth, brush);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG
 *
 * Ρυθμίζει το φόντο και σχεδιάζει το fractal χιονονιφάδας
 * στο κέντρο του παραθύρου, καλώντας τη drawSnowflake.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Ρύθμιση βούρτσας για τη χιονονιφάδα
    graphics::Brush brush;
```

```
        brush.fill_color[0] = 0.0f;
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 1.0f;  // Μπλε χρώμα

        // Κλήση της συνάρτησης σχεδίασης του fractal στο κέντρο
        drawSnowflake(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 300, MAX_DEPTH,
brush);
    }

}
namespace LSystemFractal {
    // Ορισμός παραμέτρων παραθύρου
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 600;
    const float ANGLE_INCREMENT = 25.0f;  // Βήμα περιστροφής για το L-System
    const int MAX_DEPTH = 5;              // Βάθος επανάληψης του L-System
    const float LENGTH = 10.0f;           // Μήκος γραμμής για κάθε κίνηση της χελώνας

    /**
     * Κλάση Turtle για την απεικόνιση σχεδίων με βάση εντολές κινήσεων.
     * Αποθηκεύει την τρέχουσα θέση, γωνία και το χρώμα της γραμμής.
     */
    class Turtle {
    public:
        float x, y;
        float angle;
        bool penDown;
        graphics::Brush brush;

        /**
         * Κατασκευαστής Turtle που θέτει τις αρχικές συντεταγμένες.
         * Αρχική γωνία: -90 μοίρες (επάνω).
```

```
 */
Turtle(float startX, float startY) : x(startX), y(startY), angle(-90), penDown(true) {
    brush.fill_color[0] = 0.0f;   // Πράσινο χρώμα
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.0f;
}

/**
 * Κίνηση της χελώνας προς την κατεύθυνση της τρέχουσας γωνίας.
 * @param distance: Απόσταση κίνησης
 */
void move(float distance) {
    float newX = x + distance * cos(angle * M_PI / 180.0f);
    float newY = y + distance * sin(angle * M_PI / 180.0f);
    if (penDown) {
        graphics::drawLine(x, y, newX, newY, brush);
    }
    x = newX;
    y = newY;
}

/**
 * Περιστροφή της χελώνας κατά μια γωνία.
 * @param degrees: Γωνία περιστροφής σε μοίρες
 */
void rotate(float degrees) {
    angle += degrees;
}

/**
 * Αποθήκευση της τρέχουσας κατάστασης (θέση και γωνία) της χελώνας σε στοίβα.
 * @param stateStack: Η στοίβα που αποθηκεύει την κατάσταση της χελώνας
```

```
     */
    void pushState(std::stack<Turtle>& stateStack) {
        stateStack.push(*this);
    }

    /**
     * Επαναφορά της κατάστασης της χελώνας από τη στοίβα.
     * @param stateStack: Η στοίβα που περιέχει τις προηγούμενες καταστάσεις
     */
    void popState(std::stack<Turtle>& stateStack) {
        if (!stateStack.empty()) {
            *this = stateStack.top();
            stateStack.pop();
        }
    }
};

/**
 * Συνάρτηση για τη δημιουργία της συμβολοσειράς του L-System.
 * @param axiom: Το αρχικό στοιχείο της ακολουθίας
 * @param rule: Κανόνας παραγωγής για το L-System
 * @param depth: Βάθος επανάληψης του κανόνα παραγωγής
 * @return Η τελική συμβολοσειρά του L-System
 */
std::string generateLSystem(const std::string& axiom, const std::string& rule, int depth) {
    std::string current = axiom;
    for (int i = 0; i < depth; ++i) {
        std::string next;
        for (char c : current) {
            if (c == 'F') {
                next += rule;  // Αντικατάσταση 'F' με τον κανόνα
            }
```

```
            else {
                next += c;     // Διατήρηση άλλων συμβόλων
            }
        }
        current = next;
    }
    return current;
}

/**
 * Συνάρτηση σχεδίασης του L-System με βάση την ακολουθία που παράγεται.
 * @param turtle: Η χελώνα που σχεδιάζει το fractal
 * @param sequence: Η ακολουθία εντολών που παράγεται από το L-System
 */
void drawLSystem(Turtle& turtle, const std::string& sequence) {
    std::stack<Turtle> stateStack;
    for (char command : sequence) {
        if (command == 'F') {
            turtle.move(LENGTH);  // Μετακίνηση της χελώνας
        }
        else if (command == '+') {
            turtle.rotate(ANGLE_INCREMENT);  // Περιστροφή δεξιά
        }
        else if (command == '-') {
            turtle.rotate(-ANGLE_INCREMENT); // Περιστροφή αριστερά
        }
        else if (command == '[') {
            turtle.pushState(stateStack);    // Αποθήκευση κατάστασης
        }
        else if (command == ']') {
            turtle.popState(stateStack);     // Επαναφορά κατάστασης
        }
```

```
        }
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ
     */
    void draw() {
        // Καθαρισμός του παραθύρου
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Ορισμός των εντολών και των κανόνων του L-System
        std::string axiom = "F";
        std::string rule = "F[+F]F[-F]F";  // Κανόνας παραγωγής για τη δομή του fractal

        Turtle turtle(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 50);  // Αρχική θέση της χελώνας
        turtle.brush.fill_color[0] = 0.0f;  // Χρώμα χελώνας
        turtle.brush.fill_color[1] = 0.3f;
        turtle.brush.fill_color[2] = 0.1f;

        // Δημιουργία ακολουθίας του L-System και σχεδίαση
        std::string lSystemSequence = generateLSystem(axiom, rule, MAX_DEPTH);
        drawLSystem(turtle, lSystemSequence);
    }

}
namespace VortexFractal {
```

```
// Ορισμός διαστάσεων παραθύρου
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int MAX_DEPTH = 5;            // Μέγιστο βάθος αναδρομής
const float INITIAL_RADIUS = 100.0f; // Αρχική ακτίνα του πρώτου κύκλου
const float SCALE_FACTOR = 0.7f;     // Παράγοντας κλίμακας για κάθε νέο κύκλο
const float ANGLE_INCREMENT = 30.0f; // Γωνιακή περιστροφή για κάθε νέο κύκλο

/**
 * Συνάρτηση σχεδίασης κύκλου (βασικό στοιχείο του fractal).
 * @param x, y: Συντεταγμένες κέντρου του κύκλου
 * @param radius: Ακτίνα του κύκλου
 * @param brush: Χρώμα σχεδίασης του κύκλου
 *
 * Η συνάρτηση σχεδιάζει έναν κύκλο στο σημείο `(x, y)` με ακτίνα `radius` και χρώμα που ορίζεται από το `brush`.
 */
void drawCircle(float x, float y, float radius, const graphics::Brush& brush) {
    graphics::drawDisk(x, y, radius, brush);
}

/**
 * Αναδρομική συνάρτηση για τη δημιουργία του Vortex Fractal.
 * @param x, y: Συντεταγμένες κέντρου του τρέχοντος κύκλου
 * @param radius: Ακτίνα του τρέχοντος κύκλου
 * @param depth: Τρέχον βάθος αναδρομής
 * @param angle: Γωνία περιστροφής για τον επόμενο κύκλο
 *
 * Η συνάρτηση σχεδιάζει έναν κύκλο και κατόπιν καλείται αναδρομικά για να δημιουργήσει έναν μικρότερο κύκλο,
 * περιστρεφόμενο κατά `ANGLE_INCREMENT` μοίρες γύρω από τον προηγούμενο.
 */
```

```
void drawVortex(float x, float y, float radius, int depth, float angle) {
    if (depth == 0) return;  // Όριο βάθους αναδρομής

    // Ορισμός του χρώματος για τον τρέχοντα κύκλο
    graphics::Brush brush;
    brush.fill_color[0] = 0.3f + (depth * 0.1f); // Σταδιακή αλλαγή χρώματος
    brush.fill_color[1] = 0.2f;
    brush.fill_color[2] = 0.5f + (depth * 0.1f);
    drawCircle(x, y, radius, brush);

    // Υπολογισμός παραμέτρων για τον επόμενο κύκλο
    float newRadius = radius * SCALE_FACTOR;  // Μείωση ακτίνας για τον επόμενο κύκλο
    float newAngle = angle + ANGLE_INCREMENT; // Αύξηση γωνίας περιστροφής

    // Υπολογισμός συντεταγμένων του επόμενου κύκλου
    float newX = x + radius * cos(newAngle * M_PI / 180.0f);
    float newY = y + radius * sin(newAngle * M_PI / 180.0f);

    // Αναδρομική κλήση για σχεδίαση του επόμενου κύκλου
    drawVortex(newX, newY, newRadius, depth - 1, newAngle);
}

/**
 * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *
 * Αρχικά καθαρίζει το παράθυρο και στη συνέχεια καλεί τη `drawVortex` για να σχεδιάσει το fractal στο κέντρο του παραθύρου.
 */
void draw() {
    // Καθαρισμός του παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
```

```
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κεντρική κλήση για σχεδίαση του vortex fractal
        drawVortex(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, INITIAL_RADIUS, MAX_DEPTH, 0.0f);
    }

}
namespace DurerPentagonFractal {

    // Ορισμός διαστάσεων παραθύρου
    const int WINDOW_WIDTH = 600;
    const int WINDOW_HEIGHT = 600;
    const int MAX_DEPTH = 5;        // Μέγιστο βάθος αναδρομής
    const float SCALE_FACTOR = 0.38f; // Συντελεστής κλίμακας για κάθε νέο πεντάγωνο

    /**
     * Υπολογίζει τις κορυφές ενός πενταγώνου με κέντρο τις συντεταγμένες (x, y).
     * @param x, y: Κέντρο του πενταγώνου
     * @param radius: Ακτίνα από το κέντρο μέχρι τις κορυφές του πενταγώνου
     * @param rotation: Προαιρετική γωνία περιστροφής του πενταγώνου
     * @return: Διάνυσμα ζευγών συντεταγμένων που αντιπροσωπεύουν τις κορυφές του πενταγώνου
     */
    std::vector<std::pair<float, float>> calculatePentagonVertices(float x, float y, float radius, float rotation = 0) {
        std::vector<std::pair<float, float>> vertices;
        for (int i = 0; i < 5; i++) {
            float angle = rotation + i * 2 * M_PI / 5; // Υπολογισμός γωνίας για κάθε κορυφή
            float vx = x + radius * cos(angle);        // Συντεταγμένη x της κορυφής
```

```
            float vy = y + radius * sin(angle);        // Συντεταγμένη y της κορυφής
            vertices.push_back({ vx, vy });
        }
        return vertices;
    }

    /**
     * Σχεδιάζει ένα πεντάγωνο με βάση τις κορυφές του.
     * @param vertices: Διάνυσμα ζευγών συντεταγμένων για τις κορυφές του πενταγώνου
     * @param brush: Αντικείμενο graphics::Brush για τον καθορισμό χρώματος σχεδίασης
     *
     * Η συνάρτηση σχεδιάζει ένα πεντάγωνο συνδέοντας τις κορυφές μεταξύ τους.
     */
    void drawPentagon(const std::vector<std::pair<float, float>>& vertices, const graphics::Brush& brush) {
        for (size_t i = 0; i < vertices.size(); i++) {
            size_t next = (i + 1) % vertices.size();  // Υπολογισμός της επόμενης κορυφής για σύνδεση
            graphics::drawLine(vertices[i].first, vertices[i].second, vertices[next].first, vertices[next].second, brush);
        }
    }

    /**
     * Αναδρομική συνάρτηση για τη δημιουργία του Dürer Pentagon Fractal.
     * @param x, y: Κέντρο του τρέχοντος πενταγώνου
     * @param radius: Ακτίνα του τρέχοντος πενταγώνου
     * @param depth: Τρέχον επίπεδο βάθους
     *
     * Η συνάρτηση σχεδιάζει ένα πεντάγωνο και αναδρομικά καλεί τον εαυτό της για να δημιουργήσει μικρότερα πεντάγωνα σε κάθε κορυφή του.
     */
    void drawDurerPentagonFractal(float x, float y, float radius, int depth) {
```

```
        if (depth == 0) return;  // Τερματίζουμε την αναδρομή όταν φτάσουμε στο μέγιστο βάθος

        // Ρύθμιση του χρώματος ανάλογα με το βάθος για διαφοροποίηση των επιπέδων
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f + 0.1f * depth; // Μεταβολή κόκκινου χρώματος ανά βάθος
        brush.fill_color[1] = 0.4f + 0.1f * depth; // Μεταβολή πράσινου χρώματος
        brush.fill_color[2] = 0.6f - 0.1f * depth; // Μείωση μπλε χρώματος με το βάθος

        // Υπολογισμός και σχεδίαση του κεντρικού πενταγώνου
        auto vertices = calculatePentagonVertices(x, y, radius);
        drawPentagon(vertices, brush);

        // Αναδρομική κλήση για δημιουργία μικρότερων πενταγώνων σε κάθε κορυφή
        for (auto& vertex : vertices) {
            drawDurerPentagonFractal(vertex.first, vertex.second, radius * SCALE_FACTOR, depth
- 1);
        }
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
     *
     * Η συνάρτηση αρχικά καθαρίζει το παράθυρο και στη συνέχεια καλεί τη
drawDurerPentagonFractal
     * για να ξεκινήσει τη διαδικασία δημιουργίας του fractal από το κέντρο του παραθύρου.
     */
    void draw() {
        // Καθαρισμός του παραθύρου με μαύρο φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
```

```
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);


        // Κεντρική κλήση της συνάρτησης για τη σχεδίαση του fractal
        drawDurerPentagonFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200,
MAX_DEPTH);
    }


}
namespace BarnsleyFernFractal {
    // Ορισμός διαστάσεων παραθύρου
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 600;
    const int ITERATIONS = 100000;  // Αριθμός επαναλήψεων για σχεδίαση του φράκταλ


    /**
     * Κλιμακώνει την τιμή x για να ταιριάζει με τις συντεταγμένες της οθόνης.
     * @param x: Η x συντεταγμένη στο χώρο του φράκταλ
     * @return: Η κλιμακωμένη x συντεταγμένη στο χώρο της οθόνης
     */
    float scaleX(float x) {
        return (x + 2.5f) * (WINDOW_WIDTH / 5.0f);
    }


    /**
     * Κλιμακώνει την τιμή y για να ταιριάζει με τις συντεταγμένες της οθόνης.
     * @param y: Η y συντεταγμένη στο χώρο του φράκταλ
     * @return: Η κλιμακωμένη y συντεταγμένη στο χώρο της οθόνης
     */
    float scaleY(float y) {
        return WINDOW_HEIGHT - (y * (WINDOW_HEIGHT / 10.0f));
    }
```

```
/**
 * Συνάρτηση σχεδίασης του φράκταλ Barnsley Fern.
 *
 * Χρησιμοποιεί μια σειρά πιθανοτικών μετασχηματισμών για την ενημέρωση των συντεταγμένων x και y και σχεδιάζει
 * έναν μικρό κύκλο (pixel) στις αντίστοιχες κλιμακωμένες συντεταγμένες της οθόνης.
 */
void drawFern() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f; // Πράσινο χρώμα
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.0f;

    float x = 0.0f, y = 0.0f; // Αρχικές συντεταγμένες

    // Επαναλαμβανόμενος βρόχος για κάθε επανάληψη
    for (int i = 0; i < ITERATIONS; ++i) {
        float nextX, nextY;
        float r = static_cast<float>(rand()) / RAND_MAX; // Τυχαίος αριθμός μεταξύ 0 και 1

        // Επιλογή μετασχηματισμού με βάση την τιμή του r
        if (r < 0.01f) {
            // F1: Απλός κάθετος μετασχηματισμός (με μικρή πιθανότητα)
            nextX = 0.0f;
            nextY = 0.16f * y;
        }
        else if (r < 0.86f) {
            // F2: Κύριος μετασχηματισμός για τα φύλλα της φτέρης
            nextX = 0.85f * x + 0.04f * y;
            nextY = -0.04f * x + 0.85f * y + 1.6f;
        }
```

```
            else if (r < 0.93f) {
                // F3: Δημιουργία του αριστερού φύλλου
                nextX = 0.2f * x - 0.26f * y;
                nextY = 0.23f * x + 0.22f * y + 1.6f;
            }
            else {
                // F4: Δημιουργία του δεξιού φύλλου
                nextX = -0.15f * x + 0.28f * y;
                nextY = 0.26f * x + 0.24f * y + 0.44f;
            }

            // Ενημέρωση των συντεταγμένων
            x = nextX;
            y = nextY;

            // Κλιμάκωση των συντεταγμένων στο χώρο της οθόνης
            float px = scaleX(x);
            float py = scaleY(y);

            // Σχεδίαση του σημείου (pixel) στη θέση (px, py)
            graphics::drawDisk(px, py, 1, brush);
        }
    }

    /**
     * Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
     *
     * Η συνάρτηση καθαρίζει το παράθυρο με λευκό φόντο και κατόπιν καλεί τη συνάρτηση drawFern()
     * για να σχεδιάσει το φράκταλ της φτέρης.
     */
    void draw() {
```

```
        // Καθαρισμός παραθύρου με λευκό φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 1.0f;
        bg.fill_color[1] = 1.0f;
        bg.fill_color[2] = 1.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης σχεδίασης του φράκταλ
        drawFern();
    }

}
namespace SierpinskiHexagonFractal {
    // Ορισμός παραμέτρων παραθύρου
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 600;
    const int MAX_DEPTH = 5; // Μέγιστο βάθος αναδρομής του fractal

    /**
     * @brief Υπολογίζει τις συντεταγμένες των κορυφών ενός εξαγώνου με βάση το κέντρο και το μέγεθός του.
     * @param cx Συντεταγμένη x του κέντρου του εξαγώνου
     * @param cy Συντεταγμένη y του κέντρου του εξαγώνου
     * @param size Η απόσταση από το κέντρο προς τις κορυφές
     * @return Ένας vector που περιέχει τα ζεύγη συντεταγμένων (x, y) των κορυφών του εξαγώνου
     */
    std::vector<std::pair<float, float>> getHexagonVertices(float cx, float cy, float size) {
        std::vector<std::pair<float, float>> vertices;
        for (int i = 0; i < 6; ++i) {
            float angle = M_PI / 3 * i; // Κάθε γωνία του εξαγώνου είναι 60 μοίρες
            float x = cx + size * cos(angle);
```

```
            float y = cy + size * sin(angle);
            vertices.push_back({ x, y }); // Προσθήκη κορυφής στον vector
        }
        return vertices;
    }

    /**
     * @brief Σχεδιάζει ένα εξάγωνο συνδέοντας τις κορυφές του.
     * @param vertices Ένας vector που περιέχει τις κορυφές του εξαγώνου
     * @param brush Το αντικείμενο Brush που χρησιμοποιείται για τον καθορισμό του χρώματος σχεδίασης
     */
    void drawHexagon(const std::vector<std::pair<float, float>>& vertices, const graphics::Brush& brush) {
        for (size_t i = 0; i < vertices.size(); ++i) {
            size_t next = (i + 1) % vertices.size(); // Υπολογισμός επόμενης κορυφής για σύνδεση γραμμών
            graphics::drawLine(vertices[i].first, vertices[i].second, vertices[next].first, vertices[next].second, brush);
        }
    }

    /**
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το Sierpinski Hexagon fractal.
     * @param cx Συντεταγμένη x του κέντρου του εξαγώνου
     * @param cy Συντεταγμένη y του κέντρου του εξαγώνου
     * @param size Η απόσταση από το κέντρο προς τις κορυφές
     * @param depth Το τρέχον βάθος αναδρομής, που μειώνεται κατά 1 σε κάθε κλήση
     */
    void drawSierpinskiHexagon(float cx, float cy, float size, int depth) {
        // Όταν το βάθος είναι 0, σχεδιάζεται το τρέχον εξάγωνο και η αναδρομή σταματά
        if (depth == 0) {
            graphics::Brush brush;
```

```
            brush.fill_color[0] = 0.2f; // Σκούρο πράσινο
            brush.fill_color[1] = 0.6f;
            brush.fill_color[2] = 0.2f;
            auto vertices = getHexagonVertices(cx, cy, size); // Υπολογισμός κορυφών εξαγώνου
            drawHexagon(vertices, brush); // Σχεδίαση εξαγώνου
            return;
        }

        float newSize = size / 3.0f; // Μείωση του μεγέθους κάθε επόμενου εξαγώνου στο 1/3 του αρχικού

        // Αναδρομική σχεδίαση των εξαγώνων στις έξι κορυφές γύρω από το κεντρικό
        for (int i = 0; i < 6; ++i) {
            float angle = M_PI / 3 * i;
            float nx = cx + 2 * newSize * cos(angle); // Υπολογισμός x για την κορυφή
            float ny = cy + 2 * newSize * sin(angle); // Υπολογισμός y για την κορυφή
            drawSierpinskiHexagon(nx, ny, newSize, depth - 1); // Αναδρομική κλήση για την κορυφή
        }

        // Κλήση αναδρομής για το κεντρικό εξάγωνο στο παρόν επίπεδο
        drawSierpinskiHexagon(cx, cy, newSize, depth - 1);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ από τη βιβλιοθήκη γραφικών.
     * Καθαρίζει την οθόνη και σχεδιάζει το Sierpinski Hexagon στο κέντρο του παραθύρου.
     */
    void draw() {
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f; // Μαύρο φόντο
        bg.fill_color[1] = 0.0f;
```

```
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Σχεδίαση του Sierpinski Hexagon με κέντρο το κέντρο του παραθύρου
        drawSierpinskiHexagon(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 2, MAX_DEPTH);
    }

}
namespace CrystalGrowthFractal {
    // Ορισμός παραμέτρων παραθύρου
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 700;
    const int MAX_DEPTH = 4;  // Μέγιστο βάθος αναδρομής για το fractal

    /**
     * @struct Point
     * @brief Απλή δομή που αναπαριστά ένα σημείο σε 2D επίπεδο.
     */
    struct Point {
        float x, y;
    };

    /**
     * @brief Δημιουργεί τις κορυφές ενός κανονικού πολυγώνου με βάση το κέντρο, την ακτίνα και τον αριθμό πλευρών.
     * @param cx Συντεταγμένη x του κέντρου του πολυγώνου
     * @param cy Συντεταγμένη y του κέντρου του πολυγώνου
     * @param radius Η απόσταση από το κέντρο προς τις κορυφές
     * @param sides Ο αριθμός των πλευρών του πολυγώνου
     * @return Ένας vector που περιέχει τα σημεία των κορυφών του πολυγώνου
     */
```

```
std::vector<Point> getPolygonVertices(float cx, float cy, float radius, int sides) {
    std::vector<Point> vertices;
    for (int i = 0; i < sides; ++i) {
        float angle = 2 * M_PI * i / sides; // Υπολογισμός γωνίας για κάθε κορυφή
        vertices.push_back({ cx + radius * cos(angle), cy + radius * sin(angle) });
    }
    return vertices;
}

/**
 * @brief Σχεδιάζει ένα πολύγωνο συνδέοντας διαδοχικά τις κορυφές που δίνονται.
 * @param vertices Ένας vector που περιέχει τα σημεία των κορυφών του πολυγώνου
 * @param brush Το αντικείμενο Brush που χρησιμοποιείται για τον καθορισμό του χρώματος σχεδίασης
 */
void drawPolygon(const std::vector<Point>& vertices, const graphics::Brush& brush) {
    for (size_t i = 0; i < vertices.size(); ++i) {
        size_t next = (i + 1) % vertices.size(); // Υπολογισμός επόμενης κορυφής για κλείσιμο πολυγώνου
        graphics::drawLine(vertices[i].x, vertices[i].y, vertices[next].x, vertices[next].y, brush);
    }
}

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το Crystal Growth Fractal.
 * @param cx Συντεταγμένη x του κέντρου του πολυγώνου
 * @param cy Συντεταγμένη y του κέντρου του πολυγώνου
 * @param radius Η ακτίνα του πολυγώνου
 * @param sides Ο αριθμός των πλευρών του πολυγώνου
 * @param depth Το τρέχον βάθος αναδρομής, που μειώνεται κατά 1 σε κάθε κλήση
 */
void drawCrystalGrowth(float cx, float cy, float radius, int sides, int depth) {
```

```
        // Όταν το βάθος είναι 0, σταματάει η αναδρομή
        if (depth == 0) return;

        graphics::Brush brush;
        brush.fill_color[0] = 0.3f + 0.1f * depth; // Αυξάνεται η φωτεινότητα του χρώματος με το βάθος
        brush.fill_color[1] = 0.5f;
        brush.fill_color[2] = 0.7f;

        // Υπολογισμός και σχεδίαση του πολυγώνου στο τρέχον επίπεδο
        auto vertices = getPolygonVertices(cx, cy, radius, sides);
        drawPolygon(vertices, brush);

        // Δημιουργία μικρότερων πολυγώνων στις κορυφές με αναδρομή
        float newRadius = radius * 0.5f; // Νέα ακτίνα για τα μικρότερα πολύγωνα
        for (const auto& vertex : vertices) {
            drawCrystalGrowth(vertex.x, vertex.y, newRadius, sides, depth - 1);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών σε κάθε καρέ.
     * Καθαρίζει την οθόνη και καλεί την αναδρομική συνάρτηση σχεδίασης του Crystal Growth Fractal.
     */
    void draw() {
        // Ρύθμιση μαύρου φόντου για αντίθεση με το fractal
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);
```

```
        // Κλήση της συνάρτησης για τη σχεδίαση του Crystal Growth Fractal στο κέντρο της οθόνης
        drawCrystalGrowth(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 4, 6, MAX_DEPTH);
    }

}
namespace FractalHands {
    // Σταθερές διαστάσεων παραθύρου και βάθους fractal
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 700;
    const int MAX_DEPTH = 20;  // Μέγιστο βάθος αναδρομής για το fractal

    /**
     * @brief Σχεδιάζει ένα "χέρι" με συγκεκριμένο μήκος και πάχος.
     * @param x Η αρχική συντεταγμένη x του χεριού.
     * @param y Η αρχική συντεταγμένη y του χεριού.
     * @param length Το μήκος του χεριού.
     * @param thickness Το πάχος του χεριού.
     * @param brush Το πινέλο (χρώμα) που θα χρησιμοποιηθεί για το χέρι.
     */
    void drawHand(float x, float y, float length, float thickness, const graphics::Brush& brush) {
        // Σχεδιάζουμε το κύριο "χέρι" ως γραμμή από το (x, y) μέχρι (x + length, y)
        graphics::drawLine(x, y, x + length, y, brush);

        // Δύο μικρότερες γραμμές (δάχτυλα) εκτείνονται από το τέλος του "χεριού" προς τα πάνω και κάτω
        float fingerLength = length / 3;
        for (int i = -1; i <= 1; i += 2) {  // Δύο δάχτυλα: ένα πάνω και ένα κάτω
            float fingerX = x + length;      // Η αρχική x για το δάχτυλο
            float fingerY = y + i * thickness;  // Η αρχική y για το δάχτυλο (ανάλογα με το i)
            graphics::drawLine(fingerX, fingerY, fingerX + fingerLength, fingerY, brush);
```

```
    }
}

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του fractal "Fractal Hands".
 *        Σε κάθε αναδρομική κλήση, δημιουργούνται δύο νέα "χέρια" που εκτείνονται
 *        από το τέλος του κύριου "χεριού".
 * @param x Η συντεταγμένη x του σημείου εκκίνησης για το χέρι.
 * @param y Η συντεταγμένη y του σημείου εκκίνησης για το χέρι.
 * @param length Το μήκος του χεριού.
 * @param thickness Το πάχος του χεριού.
 * @param depth Το τρέχον βάθος αναδρομής. Όταν φτάσει το μηδέν, η αναδρομή τερματίζεται.
 * @param angle Η γωνία εκκίνησης για τη σχεδίαση των νέων "χεριών".
 */
void drawFractalHands(float x, float y, float length, float thickness, int depth, float angle) {
    // Αν έχουμε φτάσει στο τέλος της αναδρομής, τερματίζουμε τη σχεδίαση
    if (depth == 0) return;

    // Ορισμός του πινέλου (χρώματος) για το τρέχον βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f + 0.2f * depth; // Προοδευτική αλλαγή κόκκινου
    brush.fill_color[1] = 0.5f;                // Σταθερό πράσινο
    brush.fill_color[2] = 0.7f;                // Σταθερό μπλε

    // Σχεδίαση του αρχικού χεριού
    drawHand(x, y, length, thickness, brush);

    // Υπολογισμός νέου μήκους και πάχους για τα επόμενα "χέρια"
    float newLength = length * 0.6f;
    float newThickness = thickness * 0.6f;
```

```
        // Δύο νέα "χέρια" εκτείνονται από το τέλος του τρέχοντος χεριού προς τα αριστερά και δεξιά
        for (int i = -1; i <= 1; i += 2) {
            float newAngle = angle + i * 45;  // Γωνία περιστροφής: -45 ή +45 μοίρες
            float newX = x + length * cos(angle * M_PI / 180);  // Νέα x συντεταγμένη για αναδρομή
            float newY = y + length * sin(angle * M_PI / 180);  // Νέα y συντεταγμένη για αναδρομή

            // Αλλαγή προσανατολισμού και αναδρομική σχεδίαση του επόμενου χεριού
            graphics::setOrientation(newAngle);
            drawFractalHands(newX, newY, newLength, newThickness, depth - 1, newAngle);
        }

        // Επαναφορά της περιστροφής
        graphics::resetPose();
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη σε κάθε καρέ.
     *        Αυτή η συνάρτηση καθαρίζει τον καμβά και σχεδιάζει το fractal χεριών
     *        από το κέντρο του παραθύρου.
     */
    void draw() {
        // Καθαρισμός καμβά με μαύρο χρώμα
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης για σχεδίαση του fractal hands ξεκινώντας από το κέντρο του παραθύρου
```

```
        drawFractalHands(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 5,
MAX_DEPTH, 0);
    }


}
namespace FractalMountains {
    // Σταθερές για τη διαμόρφωση παραθύρου και fractal παραμέτρων
    const int WINDOW_WIDTH = 800;          // Πλάτος παραθύρου
    const int WINDOW_HEIGHT = 600;         // Ύψος παραθύρου
    const int MAX_DEPTH = 8;               // Το μέγιστο βάθος αναδρομής για το fractal
    const float DISPLACEMENT = 80.0f;      // Αρχική τυχαία μετατόπιση του ύψους για τα
βουνά


    /**
     * @brief Σχεδιάζει ένα τμήμα του fractal βουνού.
     *        Κάθε τμήμα αναλύεται σε μικρότερα τμήματα με τυχαία μετατόπιση ύψους στο μέσο
σημείο.
     * @param x1 Η αρχική συντεταγμένη x του τμήματος.
     * @param y1 Η αρχική συντεταγμένη y του τμήματος.
     * @param x2 Η τελική συντεταγμένη x του τμήματος.
     * @param y2 Η τελική συντεταγμένη y του τμήματος.
     * @param depth Το τρέχον βάθος αναδρομής.
     * @param displacement Η τυχαία μετατόπιση ύψους για το τρέχον επίπεδο βάθους.
     */
    void drawFractalMountain(float x1, float y1, float x2, float y2, int depth, float displacement) {
        if (depth == 0) {
            // Όταν το βάθος είναι 0, σχεδιάζουμε μια γραμμή που συνδέει τα δύο σημεία (x1, y1) και
(x2, y2)
            graphics::Brush brush;
            brush.fill_color[0] = 0.4f;  // Χρώμα για το βουνό (σκούρο καφέ-γκρι)
            brush.fill_color[1] = 0.3f;
            brush.fill_color[2] = 0.3f;
            graphics::drawLine(x1, y1, x2, y2, brush);  // Σχεδίαση γραμμής
```

```
        }
        else {
            // Υπολογισμός του μέσου σημείου με τυχαία μετατόπιση ύψους
            float midX = (x1 + x2) / 2;   // Μέσο σημείο στην x
            float midY = (y1 + y2) / 2 + (rand() % int(displacement * 2)) - displacement;  // Μέσο σημείο στην y με τυχαία μετατόπιση

            // Αναδρομική κλήση για τα δύο νέα τμήματα με μειωμένη μετατόπιση
            drawFractalMountain(x1, y1, midX, midY, depth - 1, displacement / 2);
            drawFractalMountain(midX, midY, x2, y2, depth - 1, displacement / 2);
        }
    }

    /**
     * @brief Η κύρια συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
     *        Σχεδιάζει το φόντο του ουρανού και στη συνέχεια το fractal βουνό.
     */
    void draw() {
        // Καθαρισμός φόντου (ουρανός)
        graphics::Brush skyBrush;
        skyBrush.fill_color[0] = 0.5f;  // Ανοιχτό γαλάζιο χρώμα
        skyBrush.fill_color[1] = 0.7f;
        skyBrush.fill_color[2] = 1.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, skyBrush);

        // Αρχικές συντεταγμένες για τη γραμμή του βουνού, από την αριστερή προς την δεξιά άκρη του παραθύρου
        float x1 = 0;
        float y1 = WINDOW_HEIGHT / 2;  // Αρχικό ύψος του βουνού στο κέντρο του παραθύρου
        float x2 = WINDOW_WIDTH;
        float y2 = WINDOW_HEIGHT / 2;
```

```
        // Κλήση της αναδρομικής συνάρτησης για σχεδίαση του fractal βουνού
        drawFractalMountain(x1, y1, x2, y2, MAX_DEPTH, DISPLACEMENT);
    }

}
namespace PentagonFractal {
    // Σταθερές παραμέτρων παραθύρου και fractal
    const int WINDOW_WIDTH = 800;      // Πλάτος παραθύρου
    const int WINDOW_HEIGHT = 600;     // Ύψος παραθύρου
    const int MAX_DEPTH = 5;           // Μέγιστο βάθος αναδρομής του fractal

    /**
     * @brief Σχεδιάζει ένα πεντάγωνο κεντραρισμένο στις συντεταγμένες (x, y).
     * @param x Η συντεταγμένη x του κέντρου του πενταγώνου.
     * @param y Η συντεταγμένη y του κέντρου του πενταγώνου.
     * @param size Το μέγεθος (μήκος πλευράς) του πενταγώνου.
     * @param brush Το χρώμα της γραμμής για τη σχεδίαση.
     */
    void drawPentagon(float x, float y, float size, graphics::Brush& brush) {
        // Το πεντάγωνο έχει γωνίες κάθε 72 μοίρες (2π/5 ακτίνια)
        float angleIncrement = 2 * M_PI / 5;
        float angle = M_PI / 2;  // Ξεκινάμε από την κορυφή

        // Αρχικές συντεταγμένες για την πρώτη κορυφή
        float prevX = x + size * cos(angle);
        float prevY = y - size * sin(angle);

        // Σχεδίαση των πλευρών του πενταγώνου
        for (int i = 1; i <= 5; i++) {
            angle += angleIncrement;
            float newX = x + size * cos(angle);
            float newY = y - size * sin(angle);
```

```
        graphics::drawLine(prevX, prevY, newX, newY, brush);
        prevX = newX;
        prevY = newY;
    }
}

/**
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του fractal pentagon.
 * @param x Η συντεταγμένη x του κέντρου του πενταγώνου.
 * @param y Η συντεταγμένη y του κέντρου του πενταγώνου.
 * @param size Το μέγεθος (μήκος πλευράς) του πενταγώνου.
 * @param depth Το τρέχον βάθος αναδρομής (μειώνεται σε κάθε κλήση).
 */
void drawFractalPentagon(float x, float y, float size, int depth) {
    if (depth == 0) return;  // Βάση αναδρομής: όταν depth = 0, τερματίζουμε την αναδρομή

    // Ορισμός χρώματος που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f * depth;  // Ρύθμιση χρώματος ανά επίπεδο βάθους
    brush.fill_color[1] = 0.2f * depth;
    brush.fill_color[2] = 0.3f * depth;

    // Σχεδίαση του κεντρικού πενταγώνου
    drawPentagon(x, y, size, brush);

    // Τοποθέτηση και αναδρομική κλήση για μικρότερα πεντάγωνα στις γωνίες
    float angleIncrement = 2 * M_PI / 5;
    float angle = M_PI / 2;

    for (int i = 0; i < 5; i++) {
        float newX = x + size * cos(angle) / 2.5;
        float newY = y - size * sin(angle) / 2.5;
```

```
            // Κλήση για το νέο πεντάγωνο με μειωμένο μέγεθος και βάθος
            drawFractalPentagon(newX, newY, size / 2.5, depth - 1);
            angle += angleIncrement;
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
     *        Σχεδιάζει το φόντο και το fractal pentagon στο κέντρο του παραθύρου.
     */
    void draw() {
        // Καθαρισμός φόντου με μαύρο χρώμα
        graphics::Brush bgBrush;
        bgBrush.fill_color[0] = 0.0f;
        bgBrush.fill_color[1] = 0.0f;
        bgBrush.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bgBrush);

        // Κλήση της συνάρτησης σχεδίασης fractal pentagon
        drawFractalPentagon(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, MAX_DEPTH);
    }

}
namespace ThueMorseCurve {
    // Σταθερές παραμέτρων παραθύρου και βάθους
    const int WINDOW_WIDTH = 800;     // Πλάτος παραθύρου
    const int WINDOW_HEIGHT = 600;    // Ύψος παραθύρου
    const int DEPTH = 15;             // Βάθος της ακολουθίας Thue-Morse
```

```
/**
 * @brief Γεννάει την ακολουθία Thue-Morse μέχρι το δεδομένο βάθος.
 * @param depth Το βάθος για την ακολουθία Thue-Morse.
 * @return Επιστρέφει έναν vector<int> με την ακολουθία Thue-Morse.
 */
std::vector<int> generateThueMorseSequence(int depth) {
    std::vector<int> sequence = { 0 };  // Αρχική ακολουθία με το πρώτο στοιχείο
    for (int i = 0; i < depth; i++) {
        // Δημιουργία αντιγράφου της ακολουθίας και αντιστροφή ψηφίων
        std::vector<int> temp = sequence;
        for (int j = 0; j < temp.size(); j++) {
            temp[j] = 1 - temp[j];  // Αντιστροφή του ψηφίου (0 -> 1, 1 -> 0)
        }
        sequence.insert(sequence.end(), temp.begin(), temp.end());  // Συνένωση ακολουθιών
    }
    return sequence;
}

/**
 * @brief Σχεδιάζει την καμπύλη Thue-Morse βασισμένη στην ακολουθία περιστροφών που δημιουργείται.
 * @param startX Αρχική συντεταγμένη x.
 * @param startY Αρχική συντεταγμένη y.
 * @param length Το μήκος κάθε γραμμής.
 * @param angle Η αρχική γωνία σχεδίασης.
 * @param sequence Η ακολουθία Thue-Morse που καθορίζει τις περιστροφές.
 */
void drawThueMorseCurve(float startX, float startY, float length, float angle, const std::vector<int>& sequence) {
    // Θέση και γωνία για τη σχεδίαση
    float x = startX;
    float y = startY;
```

```
        float currentAngle = angle;

        // Ορισμός χρώματος γραμμής
        graphics::Brush brush;
        brush.fill_color[0] = 0.0f; // Μαύρο χρώμα
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;

        // Σχεδίαση της καμπύλης σύμφωνα με την ακολουθία
        for (int i = 0; i < sequence.size(); i++) {
            // Υπολογισμός των συντεταγμένων του επόμενου σημείου
            float newX = x + length * cos(currentAngle * M_PI / 180.0f);
            float newY = y + length * sin(currentAngle * M_PI / 180.0f);
            graphics::drawLine(x, y, newX, newY, brush);  // Σχεδίαση γραμμής από το (x, y) στο (newX, newY)

            // Αναβάθμιση θέσης για το επόμενο σημείο
            x = newX;
            y = newY;

            // Καθορισμός περιστροφής σύμφωνα με την ακολουθία
            if (sequence[i] == 1) {
                currentAngle += 90;  // Περιστροφή δεξιά κατά 90 μοίρες
            }
            else {
                currentAngle -= 90;  // Περιστροφή αριστερά κατά 90 μοίρες
            }
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ.
```

```
     *       Σχεδιάζει το φόντο και την καμπύλη Thue-Morse.
     */
    void draw() {
        // Καθαρισμός φόντου
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f; // Λευκό φόντο
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Δημιουργία ακολουθίας Thue-Morse
        std::vector<int> thueMorseSequence = generateThueMorseSequence(DEPTH);

        // Σχεδίαση της καμπύλης από την αριστερή πλευρά του παραθύρου
        drawThueMorseCurve(WINDOW_WIDTH / 4, WINDOW_HEIGHT / 2, 10, 0, thueMorseSequence);
    }

}
namespace JellyfishFractal {
    // Ρυθμίσεις για το παράθυρο
    const int WINDOW_WIDTH = 800;      // Πλάτος παραθύρου
    const int WINDOW_HEIGHT = 700;     // Ύψος παραθύρου

    // Παράμετροι για το fractal jellyfish
    const float INITIAL_RADIUS = 80.0f;    // Αρχική ακτίνα του σώματος της μέδουσας
    const float RADIUS_DECAY = 0.7f;       // Συντελεστής μείωσης ακτίνας για κάθε επανάληψη
    const float TENTACLE_LENGTH = 100.0f;  // Αρχικό μήκος πλοκαμιών
    const float TENTACLE_DECAY = 0.7f;     // Συντελεστής μείωσης μήκους για κάθε τμήμα του πλοκαμιού
    const int NUM_TENTACLES = 6;           // Αριθμός πλοκαμιών
    const int MAX_DEPTH = 6;               // Μέγιστο βάθος αναδρομής
```

```
/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει ένα τμήμα από το πλοκάμι της μέδουσας.
 * @param x Η αρχική συντεταγμένη x.
 * @param y Η αρχική συντεταγμένη y.
 * @param length Το μήκος του πλοκαμιού.
 * @param angle Η γωνία περιστροφής του πλοκαμιού.
 * @param depth Το υπόλοιπο βάθος αναδρομής.
 */
void drawTentacle(float x, float y, float length, float angle, int depth) {
    if (depth == 0) return; // Βάση της αναδρομής

    // Υπολογισμός των συντεταγμένων του τέλους της γραμμής
    float endX = x + length * cos(angle * M_PI / 180.0f);
    float endY = y + length * sin(angle * M_PI / 180.0f);

    // Σχεδίαση της γραμμής με χρώμα που αλλάζει ανάλογα με το βάθος
    graphics::Brush brush;
    brush.fill_color[0] = 0.4f + 0.1f * depth;  // Ροζ/μωβ απόχρωση
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.8f - 0.1f * depth;
    graphics::drawLine(x, y, endX, endY, brush);

    // Αναδρομικές κλήσεις για τα επόμενα τμήματα του πλοκαμιού
    drawTentacle(endX, endY, length * TENTACLE_DECAY, angle + 15.0f, depth - 1);
    drawTentacle(endX, endY, length * TENTACLE_DECAY, angle - 15.0f, depth - 1);
}

/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το σώμα της μέδουσας και τα πλοκάμια.
 * @param x Η αρχική συντεταγμένη x του κέντρου του σώματος.
 * @param y Η αρχική συντεταγμένη y του κέντρου του σώματος.
```

```
     * @param radius Η ακτίνα του σώματος.
     * @param depth Το υπόλοιπο βάθος αναδρομής.
     */
    void drawJellyfish(float x, float y, float radius, int depth) {
        if (depth == 0) return; // Βάση της αναδρομής

        // Σχεδίαση του σώματος της μέδουσας ως δίσκος
        graphics::Brush brush;
        brush.fill_color[0] = 0.9f;  // Απαλό ροζ-μωβ χρώμα
        brush.fill_color[1] = 0.6f;
        brush.fill_color[2] = 0.9f;
        graphics::drawDisk(x, y, radius, brush);

        // Σχεδίαση των πλοκαμιών γύρω από το σώμα
        for (int i = 0; i < NUM_TENTACLES; ++i) {
            float angle = 360.0f / NUM_TENTACLES * i;  // Κατανομή πλοκαμιών γύρω από τον κύκλο
            drawTentacle(x, y + radius, TENTACLE_LENGTH, angle, MAX_DEPTH);
        }

        // Αναδρομή για τη σχεδίαση του επόμενου μικρότερου σώματος της μέδουσας
        drawJellyfish(x, y, radius * RADIUS_DECAY, depth - 1);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται σε κάθε καρέ και καθορίζει το φόντο.
     */
    void draw() {
        // Καθαρισμός παραθύρου με μαύρο φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;  // Μαύρο φόντο
        bg.fill_color[1] = 0.0f;
```

```
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κεντρική κλήση για το fractal jellyfish
        drawJellyfish(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 3, INITIAL_RADIUS, MAX_DEPTH);
    }

}
namespace GosperIslandFractal {
    // Καθορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;

    // Γωνία περιστροφής σε ακτίνια (60 μοίρες)
    const float ANGLE = 60.0f * M_PI / 180.0f;

    /**
     * @brief Δομή για την "χελώνα", η οποία διατηρεί την τρέχουσα θέση και προσανατολισμό για τη σχεδίαση.
     */
    struct Turtle {
        float x, y;         ///< Θέση x, y της χελώνας
        float angle;        ///< Γωνία προσανατολισμού
        bool penDown;       ///< Κατάσταση στυλό (ενεργό ή όχι)
        graphics::Brush brush;

        /**
         * @brief Κατασκευαστής για την αρχικοποίηση της χελώνας με θέση και χρώμα.
         * @param startX Αρχική θέση x.
         * @param startY Αρχική θέση y.
         */
```

```
    Turtle(float startX, float startY) : x(startX), y(startY), angle(0), penDown(true) {
        brush.fill_color[0] = 0.0f; // Μαύρο χρώμα για τη σχεδίαση
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;
    }

    /**
     * @brief Μετακίνηση της χελώνας προς τα εμπρός, αφήνοντας γραμμή αν το στυλό είναι κάτω.
     * @param distance Απόσταση μετακίνησης.
     */
    void forward(float distance) {
        float newX = x + distance * cos(angle);
        float newY = y + distance * sin(angle);
        if (penDown) {
            graphics::drawLine(x, y, newX, newY, brush);
        }
        x = newX;
        y = newY;
    }

    /**
     * @brief Στροφή της χελώνας προς τα αριστερά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnLeft(float radians) {
        angle -= radians;
    }

    /**
     * @brief Στροφή της χελώνας προς τα δεξιά.
     * @param radians Γωνία στροφής σε ακτίνια.
```

```
     */
    void turnRight(float radians) {
        angle += radians;
    }
};

/**
 * @brief Συνάρτηση για τη δημιουργία του L-System για την καμπύλη Gosper.
 * @param iterations Αριθμός επαναλήψεων για την επέκταση της ακολουθίας.
 * @return Η ακολουθία εντολών που δημιουργήθηκε για την καμπύλη Gosper.
 */
std::string generateGosperCurve(int iterations) {
    std::string result = "A"; // Αρχικό αξίωμα της ακολουθίας
    for (int i = 0; i < iterations; ++i) {
        std::string next;
        for (char c : result) {
            if (c == 'A') {
                next += "A-B--B+A++AA+B-"; // Κανόνας για το "A"
            }
            else if (c == 'B') {
                next += "+A-BB--B-A++A+B"; // Κανόνας για το "B"
            }
            else {
                next += c; // Διατήρηση του χαρακτήρα αν δεν είναι 'A' ή 'B'
            }
        }
        result = next;
    }
    return result;
}

/**
```

```
 * @brief Σχεδίαση της καμπύλης Gosper με χρήση της δομής χελώνας και εντολών L-System.
 * @param turtle Αντικείμενο χελώνας για τη σχεδίαση.
 * @param instructions Η ακολουθία εντολών που θα ακολουθήσει η χελώνα.
 * @param length Μήκος κάθε βήματος της χελώνας.
 */
void drawGosperCurve(Turtle& turtle, const std::string& instructions, float length) {
    for (char command : instructions) {
        if (command == 'A' || command == 'B') {
            turtle.forward(length); // Μετακίνηση προς τα εμπρός
        }
        else if (command == '+') {
            turtle.turnLeft(ANGLE); // Στροφή προς τα αριστερά
        }
        else if (command == '-') {
            turtle.turnRight(ANGLE); // Στροφή προς τα δεξιά
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που εκτελείται σε κάθε καρέ.
 */
void draw() {
    // Καθαρισμός παραθύρου με μαύρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 0.0f;
    bg.fill_color[1] = 0.0f;
    bg.fill_color[2] = 0.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

    // Αρχικοποίηση της χελώνας και των οδηγιών για την καμπύλη Gosper
```

```
        Turtle turtle(WINDOW_WIDTH / 4, WINDOW_HEIGHT / 2);  // Αρχική θέση της χελώνας
        std::string instructions = generateGosperCurve(4);  // Επίπεδο επανάληψης της καμπύλης
        drawGosperCurve(turtle, instructions, 5.0f); // Μήκος κάθε βήματος
    }

}
namespace PentadendriteFractal {
    // Ρυθμίσεις διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;

    // Γωνία περιστροφής για κάθε νέο κλάδο (72 μοίρες)
    const float ANGLE = 72.0f * M_PI / 180.0f;

    /**
     * @brief Δομή για την "χελώνα", που διατηρεί την τρέχουσα θέση και γωνία της για τη σχεδίαση.
     */
    struct Turtle {
        float x, y;         ///< Συντεταγμένες της θέσης της χελώνας
        float angle;        ///< Γωνία προσανατολισμού
        graphics::Brush brush;

        /**
         * @brief Κατασκευαστής για την αρχικοποίηση της χελώνας με θέση και χρώμα.
         * @param startX Αρχική θέση x.
         * @param startY Αρχική θέση y.
         */
        Turtle(float startX, float startY) : x(startX), y(startY), angle(0) {
            brush.fill_color[0] = 0.0f; // Μαύρο χρώμα για τη σχεδίαση
            brush.fill_color[1] = 0.0f;
```

```
        brush.fill_color[2] = 0.0f;
    }

    /**
     * @brief Προχωράει τη χελώνα προς τα εμπρός αφήνοντας μια γραμμή.
     * @param distance Απόσταση μετακίνησης.
     */
    void forward(float distance) {
        float newX = x + distance * cos(angle);
        float newY = y + distance * sin(angle);
        graphics::drawLine(x, y, newX, newY, brush);
        x = newX;
        y = newY;
    }

    /**
     * @brief Περιστροφή της χελώνας προς τα αριστερά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnLeft(float radians) {
        angle -= radians;
    }

    /**
     * @brief Περιστροφή της χελώνας προς τα δεξιά.
     * @param radians Γωνία στροφής σε ακτίνια.
     */
    void turnRight(float radians) {
        angle += radians;
    }
};
```

```
/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του fractal Pentadendrite.
 * @param turtle Η χελώνα που καθοδηγεί τη σχεδίαση.
 * @param length Μήκος τρέχοντος κλάδου.
 * @param depth Βάθος αναδρομής.
 */
void drawPentadendrite(Turtle& turtle, float length, int depth) {
    if (depth == 0) {
        // Τελικό σημείο αναδρομής: απλή μετακίνηση προς τα εμπρός
        turtle.forward(length);
        return;
    }

    // Σχεδίαση του αρχικού τμήματος του κεντρικού κλάδου
    turtle.forward(length / 3);

    // Σχεδίαση πέντε υποκαταστημάτων γύρω από το τμήμα του κεντρικού κλαδιού
    for (int i = 0; i < 5; ++i) {
        // Αποθήκευση της τρέχουσας θέσης και γωνίας για επαναφορά μετά τον κλάδο
        float oldX = turtle.x;
        float oldY = turtle.y;
        float oldAngle = turtle.angle;

        // Περιστροφή της χελώνας για τη σχεδίαση του επόμενου κλάδου
        turtle.turnLeft(ANGLE * i);

        // Αναδρομή για σχεδίαση του υποκαταστήματος
        drawPentadendrite(turtle, length / 2, depth - 1);

        // Επαναφορά της θέσης και γωνίας της χελώνας
        turtle.x = oldX;
        turtle.y = oldY;
```

```
            turtle.angle = oldAngle;
        }

        // Σχεδίαση του τελικού τμήματος του κεντρικού κλαδιού
        turtle.forward(length / 3);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από την SGG Library σε κάθε καρέ.
     */
    void draw() {
        // Καθαρισμός παραθύρου με μαύρο φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Αρχικοποίηση της χελώνας στο κέντρο του παραθύρου
        Turtle turtle(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2);

        // Σχεδίαση του Pentadendrite fractal με αρχικό μήκος κλαδιού και βάθος
        drawPentadendrite(turtle, 200.0f, 4);
    }

}
namespace PinwheelTilingFractal {
    // Ρυθμίσεις παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;
    const float PI = 3.14159265358979323846;
```

```
/**
 * @brief Δομή που αντιπροσωπεύει ένα τρίγωνο.
 */
struct Triangle {
    float x1, y1;  ///< Συντεταγμένες της πρώτης κορυφής
    float x2, y2;  ///< Συντεταγμένες της δεύτερης κορυφής
    float x3, y3;  ///< Συντεταγμένες της τρίτης κορυφής
};

/**
 * @brief Σχεδιάζει ένα τρίγωνο χρησιμοποιώντας την βιβλιοθήκη γραφικών SGG.
 * @param triangle Το τρίγωνο που θα σχεδιαστεί.
 * @param brush Το πινέλο για τη ρύθμιση του χρώματος της γραμμής.
 */
void drawTriangle(const Triangle& triangle, const graphics::Brush& brush) {
    // Σχεδίαση των πλευρών του τριγώνου ως γραμμές
    graphics::drawLine(triangle.x1, triangle.y1, triangle.x2, triangle.y2, brush);
    graphics::drawLine(triangle.x2, triangle.y2, triangle.x3, triangle.y3, brush);
    graphics::drawLine(triangle.x3, triangle.y3, triangle.x1, triangle.y1, brush);
}

/**
 * @brief Αναδρομική συνάρτηση για τη διαίρεση ενός τριγώνου σε μικρότερα τρίγωνα, δημιουργώντας το fractal "Pinwheel".
 * @param t Το τρίγωνο που θα διαιρεθεί.
 * @param depth Το τρέχον βάθος αναδρομής. Όσο μεγαλύτερο το βάθος, τόσο περισσότερη λεπτομέρεια.
 * @param brush Το πινέλο που καθορίζει το χρώμα των γραμμών.
 */
void pinwheelDivide(Triangle t, int depth, const graphics::Brush& brush) {
    if (depth == 0) {
```

```
        // Βάση της αναδρομής: σχεδιάζουμε το τρίγωνο
        drawTriangle(t, brush);
        return;
    }

    // Υπολογισμός μέσων σημείων για κάθε πλευρά του τριγώνου
    float mx1 = (t.x1 + t.x2) / 2;
    float my1 = (t.y1 + t.y2) / 2;
    float mx2 = (t.x2 + t.x3) / 2;
    float my2 = (t.y2 + t.y3) / 2;
    float mx3 = (t.x3 + t.x1) / 2;
    float my3 = (t.y3 + t.y1) / 2;

    // Δημιουργία νέων μικρότερων τριγώνων
    Triangle t1 = { t.x1, t.y1, mx1, my1, mx3, my3 };
    Triangle t2 = { mx1, my1, t.x2, t.y2, mx2, my2 };
    Triangle t3 = { mx3, my3, mx2, my2, t.x3, t.y3 };
    Triangle t4 = { mx1, my1, mx2, my2, mx3, my3 };

    // Αναδρομή για κάθε νέο τρίγωνο με μειωμένο βάθος
    pinwheelDivide(t1, depth - 1, brush);
    pinwheelDivide(t2, depth - 1, brush);
    pinwheelDivide(t3, depth - 1, brush);
    pinwheelDivide(t4, depth - 1, brush);
}

/**
 * @brief Σχεδιάζει το βασικό τριγωνικό μοτίβο και ξεκινά την αναδρομική διαίρεση για το Pinwheel fractal.
 */
void drawPinwheelFractal() {
    graphics::Brush brush;
```

```
        brush.fill_color[0] = 0.1f;  // Κόκκινη απόχρωση
        brush.fill_color[1] = 0.5f;  // Πράσινη απόχρωση
        brush.fill_color[2] = 0.7f;  // Μπλε απόχρωση

        // Ορισμός του αρχικού τριγώνου για το fractal
        Triangle baseTriangle = { 400, 200, 600, 600, 200, 600 };

        // Εκκίνηση αναδρομικής διαίρεσης
        pinwheelDivide(baseTriangle, 5, brush);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για να σχεδιάσει το fractal στο παράθυρο.
     */
    void draw() {
        // Καθαρισμός του παραθύρου με μαύρο φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Σχεδίαση του fractal "Pinwheel"
        drawPinwheelFractal();
    }

}
namespace VicsekCrossFractal {
    // Ρυθμίσεις παραθύρου
    const float WINDOW_WIDTH = 800;
```

```
const float WINDOW_HEIGHT = 600;


/**
 * @brief Σχεδιάζει ένα τετράγωνο με κέντρο τις συντεταγμένες (x, y) και συγκεκριμένη πλευρά.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param side Το μήκος της πλευράς του τετραγώνου.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα του τετραγώνου.
 */
void drawSquare(float x, float y, float side, const graphics::Brush& brush) {
    graphics::drawRect(x, y, side, side, brush);
}


/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Vicsek Cross Fractal.
 *        Σε κάθε επίπεδο αναδρομής, το τετράγωνο χωρίζεται σε πέντε μικρότερα
 *        τετράγωνα στη μορφή σταυρού.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param side Το μήκος της πλευράς του τετραγώνου.
 * @param depth Το τρέχον βάθος της αναδρομής. Αν το depth είναι 0, σταματά η αναδρομή.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα των τετραγώνων.
 */
void drawVicsekCross(float x, float y, float side, int depth, const graphics::Brush& brush) {
    // Βάση της αναδρομής: αν το βάθος είναι 0, σχεδιάζουμε το τετράγωνο και επιστρέφουμε
    if (depth == 0) {
        drawSquare(x, y, side, brush);
        return;
    }

    // Υπολογισμός νέου μήκους πλευράς για τα μικρότερα τετράγωνα
```

```
    float newSide = side / 3.0f;

    // Αναδρομική κλήση για τα πέντε τετράγωνα στη μορφή του σταυρού
    drawVicsekCross(x, y, newSide, depth - 1, brush);                    // Κεντρικό τετράγωνο
    drawVicsekCross(x - newSide, y - newSide, newSide, depth - 1, brush);  // Άνω αριστερό τετράγωνο
    drawVicsekCross(x + newSide, y - newSide, newSide, depth - 1, brush);  // Άνω δεξί τετράγωνο
    drawVicsekCross(x - newSide, y + newSide, newSide, depth - 1, brush);  // Κάτω αριστερό τετράγωνο
    drawVicsekCross(x + newSide, y + newSide, newSide, depth - 1, brush);  // Κάτω δεξί τετράγωνο
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για τη δημιουργία του παραθύρου.
 *        Καθαρίζει την οθόνη και καλεί τη συνάρτηση για τη σχεδίαση του Vicsek Cross Fractal.
 */
void draw() {
    // Καθαρισμός του παραθύρου με άσπρο φόντο
    graphics::Brush bg;
    bg.fill_color[0] = 1.0f;  // Άσπρο φόντο
    bg.fill_color[1] = 1.0f;
    bg.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

    // Ρύθμιση πινέλου για το fractal
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f;  // Μπλε απόχρωση
    brush.fill_color[1] = 0.3f;
    brush.fill_color[2] = 0.8f;
```

```
            // Κεντρική κλήση της συνάρτησης αναδρομής για το Vicsek Cross Fractal
            drawVicsekCross(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 400, 4, brush);
        }

    }
    namespace HexagonTilingFractal {
        // Διαστάσεις παραθύρου
        const float WINDOW_WIDTH = 800.0f;
        const float WINDOW_HEIGHT = 600.0f;

        /**
         * @brief Δομή Hexagon που αναπαριστά ένα εξάγωνο και περιέχει συναρτήσεις για τον υπολογισμό των κορυφών του και τη σχεδίασή του.
         */
        struct Hexagon {
            float x, y;  // Συντεταγμένες του κέντρου του εξαγώνου
            float size;  // Μέγεθος της πλευράς του εξαγώνου

            /**
             * @brief Σχεδιάζει το εξάγωνο συνδέοντας τις γειτονικές κορυφές.
             * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα και τις γραμμές του εξαγώνου.
             */
            void drawHexagon(const graphics::Brush& brush) const {
                // Επαναληπτική διαδικασία για τις 6 πλευρές του εξαγώνου
                for (int i = 0; i < 6; ++i) {
                    // Υπολογισμός γωνιών για τα δύο άκρα κάθε πλευράς
                    float angle1 = M_PI / 3 * i;
                    float angle2 = M_PI / 3 * (i + 1);
                    float x1 = x + size * cos(angle1);
                    float y1 = y + size * sin(angle1);
                    float x2 = x + size * cos(angle2);
```

```
            float y2 = y + size * sin(angle2);

            // Σχεδίαση πλευράς του εξαγώνου
            graphics::drawLine(x1, y1, x2, y2, brush);
        }
    }
};

/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Hexagonal Tiling Fractal.
 *       Δημιουργεί ένα εξάγωνο στο κέντρο και στη συνέχεια καλεί την ίδια τη συνάρτηση για τα 6 γειτονικά εξάγωνα.
 * @param x Η συντεταγμένη x του κέντρου του εξαγώνου.
 * @param y Η συντεταγμένη y του κέντρου του εξαγώνου.
 * @param size Το μέγεθος της πλευράς του εξαγώνου.
 * @param depth Το βάθος αναδρομής. Αν το depth είναι 0, η συνάρτηση σταματά.
 * @param brush Το πινέλο που χρησιμοποιείται για το χρώμα και τις γραμμές του εξαγώνου.
 */
void drawHexagonalTilingFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: αν το βάθος είναι 0, δεν σχεδιάζουμε περαιτέρω εξάγωνα
    if (depth == 0) return;

    // Δημιουργία του κεντρικού εξαγώνου
    Hexagon hex = { x, y, size };
    hex.drawHexagon(brush);

    // Παράγοντες για τον υπολογισμό των συντεταγμένων των 6 γειτονικών εξαγώνων
    const float dx[] = { 1.5f, 0.75f, -0.75f, -1.5f, -0.75f, 0.75f };
    const float dy[] = { 0.0f, 1.3f, 1.3f, 0.0f, -1.3f, -1.3f };

    // Αναδρομική κλήση για τα 6 γειτονικά εξάγωνα
```

```
        for (int i = 0; i < 6; ++i) {
            // Υπολογισμός νέων συντεταγμένων για τα γειτονικά εξάγωνα
            float newX = x + dx[i] * size * 2;
            float newY = y + dy[i] * size * 2;

            // Κλήση αναδρομής για το γειτονικό εξάγωνο
            drawHexagonalTilingFractal(newX, newY, size / 2, depth - 1, brush);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών SGG. Σχεδιάζει το fractal των εξαγώνων.
     */
    void draw() {
        // Ρύθμιση του πινέλου σχεδίασης
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f;  // Απόχρωση μπλε-πράσινου για το fractal
        brush.fill_color[1] = 0.6f;
        brush.fill_color[2] = 0.8f;
        brush.outline_opacity = 1.0f;

        // Κλήση της αναδρομικής συνάρτησης για τη σχεδίαση του Hexagonal Tiling Fractal
        drawHexagonalTilingFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 4, brush);
    }

}
namespace SquareFractal {
    // Διαστάσεις παραθύρου
    const float WINDOW_WIDTH = 800.0f;
    const float WINDOW_HEIGHT = 600.0f;
```

```
/**
 * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός fractal με τετράγωνα.
 *        Σχεδιάζει ένα κεντρικό τετράγωνο και στη συνέχεια δημιουργεί μικρότερα τετράγωνα στις γωνίες του.
 * @param x Η συντεταγμένη x του κέντρου του τετραγώνου.
 * @param y Η συντεταγμένη y του κέντρου του τετραγώνου.
 * @param size Το μέγεθος της πλευράς του τετραγώνου.
 * @param depth Το βάθος αναδρομής. Η συνάρτηση σταματά αν το βάθος φτάσει στο 0.
 * @param brush Το πινέλο που χρησιμοποιείται για τον χρωματισμό των τετραγώνων.
 */
void drawSquareFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: όταν το βάθος είναι 0, σταματάμε τη σχεδίαση
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του νέου μεγέθους για τα επόμενα τετράγωνα
    float newSize = size / 2;

    // Αναδρομική σχεδίαση τετραγώνων στις τέσσερις γωνίες του τρέχοντος τετραγώνου
    drawSquareFractal(x - newSize, y - newSize, newSize, depth - 1, brush); // Πάνω αριστερά γωνία
    drawSquareFractal(x + newSize, y - newSize, newSize, depth - 1, brush); // Πάνω δεξιά γωνία
    drawSquareFractal(x - newSize, y + newSize, newSize, depth - 1, brush); // Κάτω αριστερά γωνία
    drawSquareFractal(x + newSize, y + newSize, newSize, depth - 1, brush); // Κάτω δεξιά γωνία
}

/**
```

```
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ της οθόνης.
     *        Καθορίζει το πινέλο και ξεκινά τη σχεδίαση του fractal από το κέντρο της οθόνης.
     */
    void draw() {
        // Ρύθμιση του πινέλου σχεδίασης
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f;  // Απόχρωση μπλε-πράσινου για το fractal
        brush.fill_color[1] = 0.5f;
        brush.fill_color[2] = 0.8f;
        brush.fill_opacity = 0.7f;   // Ημιδιαφανές χρώμα

        // Κλήση της συνάρτησης drawSquareFractal από το κέντρο του παραθύρου
        drawSquareFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200, 4, brush);
    }

}
namespace CarpetFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600.0f;
    const float WINDOW_HEIGHT = 600.0f;

    /**
     * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός Carpet Fractal.
     *        Η συνάρτηση δημιουργεί ένα κεντρικό τετράγωνο και στη συνέχεια
     *        το διαιρεί σε μικρότερα τετράγωνα γύρω από το κέντρο του.
     * @param x Η συντεταγμένη x του κέντρου του τρέχοντος τετραγώνου.
     * @param y Η συντεταγμένη y του κέντρου του τρέχοντος τετραγώνου.
     * @param size Το μέγεθος της πλευράς του τρέχοντος τετραγώνου.
     * @param depth Το βάθος αναδρομής. Αν το βάθος φτάσει στο 0, σταματάμε τη διαδικασία.
     * @param brush Το πινέλο που χρησιμοποιείται για τη σχεδίαση των τετραγώνων.
     */
```

```
void drawCarpetFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: αν το βάθος είναι 0, τερματίζουμε την αναδρομή
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου για το τρέχον επίπεδο
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του νέου μεγέθους για τα υποτετράγωνα στο επόμενο επίπεδο αναδρομής
    float newSize = size / 3;

    // Αναδρομική κλήση για κάθε μία από τις 8 θέσεις γύρω από το κεντρικό τετράγωνο
    for (int dx = -1; dx <= 1; ++dx) {
        for (int dy = -1; dy <= 1; ++dy) {
            // Παράκαμψη του κεντρικού τετραγώνου για να αφήσουμε κενό το κέντρο
            if (dx == 0 && dy == 0) continue;

            // Αναδρομική κλήση για σχεδίαση υποτετραγώνου στη θέση (dx, dy)
            drawCarpetFractal(x + dx * newSize, y + dy * newSize, newSize, depth - 1, brush);
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για κάθε καρέ της οθόνης.
 *        Καθορίζει το χρώμα του fractal και ξεκινά τη σχεδίαση του Carpet Fractal από το κέντρο του παραθύρου.
 */
void draw() {
    // Ρύθμιση πινέλου για το μπλε χρώμα του fractal
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
```

```
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 1.0f;  // Μπλε χρώμα
        brush.fill_opacity = 0.7f;   // Ημιδιαφανές πινέλο

        // Έναρξη της αναδρομικής σχεδίασης του fractal από το κέντρο του παραθύρου
        drawCarpetFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 600, 4, brush);
    }

}
namespace CirclePackingFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Αναδρομική συνάρτηση για τη δημιουργία του Circle Packing Fractal.
     *        Σχεδιάζει έναν κεντρικό κύκλο και μικρότερους κύκλους γύρω από αυτόν σε καθορισμένες θέσεις.
     * @param x Η x συντεταγμένη του κέντρου του τρέχοντος κύκλου.
     * @param y Η y συντεταγμένη του κέντρου του τρέχοντος κύκλου.
     * @param radius Η ακτίνα του τρέχοντος κύκλου.
     * @param depth Το βάθος της αναδρομής. Αν είναι 0 ή η ακτίνα είναι πολύ μικρή, η αναδρομή σταματά.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ του κύκλου.
     */
    void drawCirclePackingFractal(float x, float y, float radius, int depth, const graphics::Brush& brush) {
        // Βάση αναδρομής: Αν το βάθος είναι 0 ή η ακτίνα είναι πολύ μικρή, σταματάμε
        if (depth == 0 || radius < 1) return;

        // Σχεδίαση του κύριου κύκλου για το τρέχον επίπεδο
        graphics::drawDisk(x, y, radius, brush);
```

```
        // Μείωση ακτίνας για τους υποκύκλους στο επόμενο επίπεδο
        float newRadius = radius / 2;

        // Αναδρομικές κλήσεις για να δημιουργήσουμε υποκύκλους γύρω από τον κεντρικό κύκλο
        drawCirclePackingFractal(x + newRadius, y, newRadius, depth - 1, brush);  // Δεξιά
        drawCirclePackingFractal(x - newRadius, y, newRadius, depth - 1, brush);  // Αριστερά
        drawCirclePackingFractal(x, y + newRadius, newRadius, depth - 1, brush);  // Κάτω
        drawCirclePackingFractal(x, y - newRadius, newRadius, depth - 1, brush);  // Πάνω
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ της οθόνης.
     *        Καθορίζει το χρώμα του fractal και καλεί τη συνάρτηση `drawCirclePackingFractal` από το κέντρο του παραθύρου.
     */
    void draw() {
        // Ρύθμιση πινέλου με ανοιχτό μπλε χρώμα και ημιδιαφανές αποτέλεσμα
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f;
        brush.fill_color[1] = 0.6f;
        brush.fill_color[2] = 1.0f;  // Ανοιχτό μπλε χρώμα
        brush.fill_opacity = 0.8f;

        // Ξεκινάμε την αναδρομική σχεδίαση του fractal από το κέντρο του παραθύρου με αρχική ακτίνα 200
        drawCirclePackingFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 200, 5, brush);
    }

}
namespace HTreeFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600;
```

```
const float WINDOW_HEIGHT = 600;


/**
 * @brief Αναδρομική συνάρτηση για τη δημιουργία του H-Tree Fractal.
 *        Σχεδιάζει ένα "H" και στη συνέχεια μικρότερα "H" στα 4 άκρα του τρέχοντος σχήματος.
 * @param x H x συντεταγμένη του κέντρου του τρέχοντος "H".
 * @param y H y συντεταγμένη του κέντρου του τρέχοντος "H".
 * @param length Το μήκος των γραμμών του "H".
 * @param depth Το βάθος της αναδρομής. Όταν φτάνει το 0, η αναδρομή σταματά.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ του "H".
 */
void drawHTree(float x, float y, float length, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: Αν το βάθος είναι 0 ή το μήκος των γραμμών είναι πολύ μικρό, σταματάμε
    if (depth == 0 || length < 2) return;

    // Υπολογισμός των μισών και του ενός τετάρτου του μήκους των γραμμών
    float halfLength = length / 2;
    float quarterLength = length / 4;

    // Σχεδίαση των δύο κάθετων γραμμών του "H"
    graphics::drawLine(x - halfLength, y - quarterLength, x - halfLength, y + quarterLength, brush);
    graphics::drawLine(x + halfLength, y - quarterLength, x + halfLength, y + quarterLength, brush);

    // Σχεδίαση της οριζόντιας γραμμής του "H"
    graphics::drawLine(x - halfLength, y, x + halfLength, y, brush);

    // Αναδρομική κλήση για να σχεδιάσουμε μικρότερα "H" στα 4 άκρα του τρέχοντος "H"
    drawHTree(x - halfLength, y - quarterLength, length / 2, depth - 1, brush); // Αριστερά πάνω
    drawHTree(x + halfLength, y - quarterLength, length / 2, depth - 1, brush); // Δεξιά πάνω
```

```
        drawHTree(x - halfLength, y + quarterLength, length / 2, depth - 1, brush); // Αριστερά κάτω
        drawHTree(x + halfLength, y + quarterLength, length / 2, depth - 1, brush); // Δεξιά κάτω
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ της οθόνης.
     *        Καθορίζει το χρώμα του fractal και καλεί τη συνάρτηση `drawHTree` από το κέντρο του παραθύρου.
     */
    void draw() {
        graphics::Brush brush;
        brush.fill_color[0] = 0.0f;
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;  // Μαύρο χρώμα για το fractal
        brush.outline_opacity = 1.0f;

        // Εκκίνηση της σχεδίασης του H-Tree από το κέντρο του παραθύρου με αρχικό μήκος 300
        drawHTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 300, 5, brush);
    }

}
namespace GasketOfTrianglesFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 700;

    /**
     * @brief Σχεδιάζει ένα τρίγωνο χρησιμοποιώντας τις τρεις κορυφές του.
     * @param x1 Η x συντεταγμένη της πρώτης κορυφής.
     * @param y1 Η y συντεταγμένη της πρώτης κορυφής.
     * @param x2 Η x συντεταγμένη της δεύτερης κορυφής.
```

```
     * @param y2 Η y συντεταγμένη της δεύτερης κορυφής.
     * @param x3 Η x συντεταγμένη της τρίτης κορυφής.
     * @param y3 Η y συντεταγμένη της τρίτης κορυφής.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
    void drawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, const graphics::Brush& brush) {
        // Σχεδίαση των τριών πλευρών του τριγώνου
        graphics::drawLine(x1, y1, x2, y2, brush);
        graphics::drawLine(x2, y2, x3, y3, brush);
        graphics::drawLine(x3, y3, x1, y1, brush);
    }

    /**
     * @brief Αναδρομική συνάρτηση για τη δημιουργία του fractal Gasket of Triangles.
     *        Σχεδιάζει ένα τρίγωνο και το διαιρεί αναδρομικά σε τρία μικρότερα τρίγωνα.
     * @param x1 Η x συντεταγμένη της πρώτης κορυφής του τριγώνου.
     * @param y1 Η y συντεταγμένη της πρώτης κορυφής του τριγώνου.
     * @param x2 Η x συντεταγμένη της δεύτερης κορυφής του τριγώνου.
     * @param y2 Η y συντεταγμένη της δεύτερης κορυφής του τριγώνου.
     * @param x3 Η x συντεταγμένη της τρίτης κορυφής του τριγώνου.
     * @param y3 Η y συντεταγμένη της τρίτης κορυφής του τριγώνου.
     * @param depth Το βάθος της αναδρομής. Όταν φτάνει το 0, σταματάει η αναδρομή.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
    void drawGasket(float x1, float y1, float x2, float y2, float x3, float y3, int depth, const graphics::Brush& brush) {
        // Βάση αναδρομής: αν το βάθος είναι 0, σχεδιάζουμε το τρίγωνο και επιστρέφουμε
        if (depth == 0) {
            drawTriangle(x1, y1, x2, y2, x3, y3, brush);
            return;
        }
```

```
    // Υπολογισμός των μεσαίων σημείων των πλευρών
    float mx1 = (x1 + x2) / 2;
    float my1 = (y1 + y2) / 2;
    float mx2 = (x2 + x3) / 2;
    float my2 = (y2 + y3) / 2;
    float mx3 = (x1 + x3) / 2;
    float my3 = (y1 + y3) / 2;

    // Αναδρομική κλήση για τα τρία περιφερειακά τρίγωνα
    drawGasket(x1, y1, mx1, my1, mx3, my3, depth - 1, brush);
    drawGasket(mx1, my1, x2, y2, mx2, my2, depth - 1, brush);
    drawGasket(mx3, my3, mx2, my2, x3, y3, depth - 1, brush);
}

/**
 * @brief Κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ της οθόνης.
 *        Ρυθμίζει το χρώμα του fractal και καλεί τη συνάρτηση `drawGasket` για την εκκίνηση της σχεδίασης.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_opacity = 1.0f;
    brush.outline_opacity = 0.5f;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f; // Μαύρο χρώμα για το fractal

    // Εκκίνηση σχεδίασης του Gasket of Triangles με το αρχικό τρίγωνο και βάθος 5
    drawGasket(400, 50, 50, 650, 750, 650, 5, brush);
}
```

```
}
namespace ZenoParadoxFractal {

    // Διαστάσεις παραθύρου σχεδίασης
    const float WINDOW_WIDTH = 900;
    const float WINDOW_HEIGHT = 400;

    /**
     * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Zeno's Paradox Fractal.
     *        Σχεδιάζει ένα τετράγωνο και καλεί αναδρομικά τη συνάρτηση για το μισού μεγέθους τετράγωνο, που τοποθετείται στα δεξιά.
     * @param x Η x συντεταγμένη του τετραγώνου.
     * @param y Η y συντεταγμένη του τετραγώνου.
     * @param size Το μέγεθος του τετραγώνου (πλάτος και ύψος).
     * @param depth Το βάθος αναδρομής. Όταν φτάνει το 0, σταματά η αναδρομή.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
    void drawZenoFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
        // Βάση αναδρομής: Αν το βάθος είναι 0, σταματάμε
        if (depth == 0) return;

        // Σχεδίαση τετραγώνου
        graphics::drawRect(x, y, size, size, brush);

        // Αναδρομική κλήση για το επόμενο μικρότερο τετράγωνο
        // Το νέο τετράγωνο τοποθετείται δίπλα από το τρέχον, έχει μέγεθος στο μισό και βάθος μειωμένο κατά 1
        drawZenoFractal(x + size, y, size / 2, depth - 1, brush);
    }

    /**
```

```
     * @brief Κύρια συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
     *        Καθορίζει τις παραμέτρους σχεδίασης για το fractal και εκκινεί τη διαδικασία.
     */
    void draw() {
        graphics::Brush brush;
        brush.fill_color[0] = 0.0f;  // Μαύρο χρώμα γέμισμα για τα τετράγωνα
        brush.fill_color[1] = 0.0f;
        brush.fill_color[2] = 0.0f;
        brush.outline_opacity = 0.5f;  // Διαφάνεια περιγράμματος

        // Εκκίνηση του fractal από το σημείο (200, κεντρικό ύψος) με αρχικό μέγεθος 300 και βάθος αναδρομής 20
        drawZenoFractal(200, WINDOW_HEIGHT / 2 - 25, 300, 20, brush);
    }

}
namespace CarpetTilingFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το Carpet Tiling Fractal.
     *        Σε κάθε αναδρομικό επίπεδο, το κεντρικό τετράγωνο διαιρείται σε εννέα μικρότερα τετράγωνα.
     *        Η συνάρτηση επαναλαμβάνει αυτή τη διαδικασία μόνο για τα περιφερειακά τετράγωνα.
     * @param x Η x-συντεταγμένη του κεντρικού σημείου του τετραγώνου.
     * @param y Η y-συντεταγμένη του κεντρικού σημείου του τετραγώνου.
     * @param size Το μέγεθος της πλευράς του τετραγώνου.
     * @param depth Το βάθος αναδρομής, το οποίο μειώνεται σε κάθε κλήση.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
```

```
void drawCarpetFractal(float x, float y, float size, int depth, const graphics::Brush& brush) {
    // Βάση αναδρομής: Όταν το βάθος φτάσει στο 0, σταματά η αναδρομή
    if (depth == 0) return;

    // Σχεδίαση του κεντρικού τετραγώνου στο τρέχον επίπεδο
    graphics::drawRect(x, y, size, size, brush);

    // Υπολογισμός του μεγέθους για τα υποτετράγωνα, το οποίο είναι το 1/3 του τρέχοντος τετραγώνου
    float newSize = size / 3;

    // Αναδρομική σχεδίαση των 8 περιφερειακών τετραγώνων γύρω από το κεντρικό
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            // Παράλειψη του κεντρικού τετραγώνου (όταν dx και dy είναι 0)
            if (dx != 0 || dy != 0) {
                // Σχεδίαση υποτετραγώνου στη θέση που υποδεικνύεται από dx και dy
                drawCarpetFractal(x + dx * newSize, y + dy * newSize, newSize, depth - 1, brush);
            }
        }
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
 *        Καθορίζει τις παραμέτρους για το fractal και ξεκινά τη διαδικασία σχεδίασης.
 */
void draw() {
    graphics::Brush brush;
    brush.fill_color[0] = 0.1f;  // Σκούρο γκρι χρώμα για το fractal
    brush.fill_color[1] = 0.1f;
    brush.fill_color[2] = 0.1f;
```

```
        brush.outline_opacity = 0.5f;  // Ελαφρώς ημιδιαφανές περίγραμμα


        // Εκκίνηση της σχεδίασης από το κέντρο του παραθύρου με μέγεθος 1/3 του παραθύρου και βάθος 5
        drawCarpetFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH / 3, 5, brush);
    }


}
namespace IceFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;


    /**
     * @brief Αναδρομική συνάρτηση για τη σχεδίαση του Ice Fractal.
     *        Σε κάθε επανάληψη, η γραμμή χωρίζεται σε τρία τμήματα, ενώ προστίθεται ένα επιπλέον σημείο
     *        για να σχηματίσει γωνία στο κέντρο της γραμμής, δίνοντας την εμφάνιση παγοκρυστάλλου.
     *
     * @param x1 Η x-συντεταγμένη του αρχικού σημείου της γραμμής.
     * @param y1 Η y-συντεταγμένη του αρχικού σημείου της γραμμής.
     * @param x2 Η x-συντεταγμένη του τελικού σημείου της γραμμής.
     * @param y2 Η y-συντεταγμένη του τελικού σημείου της γραμμής.
     * @param depth Το τρέχον βάθος αναδρομής, που καθορίζει το επίπεδο λεπτομέρειας του fractal.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
    void drawIceFractal(float x1, float y1, float x2, float y2, int depth, const graphics::Brush& brush) {
        // Βάση της αναδρομής: αν το βάθος φτάσει στο 0, σχεδιάζεται μια ευθεία γραμμή και η αναδρομή σταματά
        if (depth == 0) {
```

```
        graphics::drawLine(x1, y1, x2, y2, brush);
        return;
    }

    // Υπολογισμός μετατόπισης για τα τμήματα της γραμμής
    float dx = x2 - x1;
    float dy = y2 - y1;

    // Σημεία διαίρεσης της γραμμής σε τρία μέρη
    float xA = x1 + dx / 3;
    float yA = y1 + dy / 3;
    float xB = x1 + 2 * dx / 3;
    float yB = y1 + 2 * dy / 3;

    // Υπολογισμός του σημείου που σχηματίζει την γωνία της γραμμής στο κέντρο
    float xPeak = (xA + xB) / 2 - (yB - yA) * std::sqrt(3) / 2;
    float yPeak = (yA + yB) / 2 + (xB - xA) * std::sqrt(3) / 2;

    // Αναδρομική κλήση για σχεδίαση των τεσσάρων τμημάτων που δημιουργούνται
    drawIceFractal(x1, y1, xA, yA, depth - 1, brush);         // Πρώτο τμήμα
    drawIceFractal(xA, yA, xPeak, yPeak, depth - 1, brush);   // Δεύτερο τμήμα με γωνία
    drawIceFractal(xPeak, yPeak, xB, yB, depth - 1, brush);   // Τρίτο τμήμα με γωνία
    drawIceFractal(xB, yB, x2, y2, depth - 1, brush);         // Τέταρτο τμήμα
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG σε κάθε καρέ.
 *        Θέτει το αρχικό χρώμα και το αρχικό μέγεθος της γραμμής, και καλεί τη συνάρτηση για να σχεδιάσει το Ice Fractal.
 */
void draw() {
    graphics::Brush brush;
```

```
        brush.fill_color[0] = 0.0f;  // Χρώμα μπλε
        brush.fill_color[1] = 0.5f;
        brush.fill_color[2] = 1.0f;

        // Ορισμός αρχικής ευθείας γραμμής στο κέντρο του παραθύρου
        float startX = WINDOW_WIDTH / 4;
        float startY = WINDOW_HEIGHT / 2;
        float endX = 3 * WINDOW_WIDTH / 4;
        float endY = WINDOW_HEIGHT / 2;

        // Σχεδίαση Ice Fractal με βάθος αναδρομής 4 για τη δημιουργία λεπτομερών σχηματισμών
        drawIceFractal(startX, startY, endX, endY, 4, brush);
    }

}
namespace ArchimedesSpiralFractal {
    // Ορισμός διαστάσεων παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Συνάρτηση για τη σχεδίαση της σπείρας του Αρχιμήδη,
     *        χρησιμοποιώντας έναν αριθμό από μικρούς κύκλους σε κάθε σημείο της σπείρας.
     *
     * @param centerX Η x-συντεταγμένη του κέντρου της σπείρας.
     * @param centerY Η y-συντεταγμένη του κέντρου της σπείρας.
     * @param iterations Ο αριθμός των σημείων/επανάληψεων που σχεδιάζονται στη σπείρα.
     * @param a Ο αρχικός παράγοντας απόστασης από το κέντρο (αρχικό μέγεθος της σπείρας).
     * @param b Ο παράγοντας αύξησης της απόστασης για κάθε βήμα (ταχύτητα ανάπτυξης της σπείρας).
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
```

```
void drawArchimedeanSpiral(float centerX, float centerY, int iterations, float a, float b, const graphics::Brush& brush) {
    // Αρχική τιμή γωνίας για την ανάπτυξη της σπείρας
    float angle = 0.0f;

    // Έναρξη επανάληψης για τον υπολογισμό και σχεδίαση των σημείων της σπείρας
    for (int i = 0; i < iterations; i++) {
        // Υπολογισμός της ακτίνας και των συντεταγμένων του νέου σημείου
        float r = a + b * angle;  // Ακτίνα με βάση τους παράγοντες a και b
        float x = centerX + r * cos(angle); // Υπολογισμός x-συντεταγμένης
        float y = centerY + r * sin(angle); // Υπολογισμός y-συντεταγμένης

        // Σχεδίαση ενός μικρού κύκλου στο υπολογισμένο σημείο της σπείρας
        graphics::drawDisk(x, y, 5, brush);

        // Αύξηση της γωνίας για το επόμενο σημείο, ελέγχοντας την απόσταση των σημείων
        angle += 0.2f; // Ελέγχει την απόσταση μεταξύ των σημείων στη σπείρα
    }
}

/**
 * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
 *        Ρυθμίζει το χρώμα και το μέγεθος της σπείρας και καλεί τη συνάρτηση σχεδίασης.
 */
void draw() {
    // Ρύθμιση του πινέλου για την εμφάνιση της σπείρας
    graphics::Brush brush;
    brush.fill_color[0] = 0.2f; // Ρύθμιση μπλε-πράσινης απόχρωσης για το fractal
    brush.fill_color[1] = 0.5f;
    brush.fill_color[2] = 0.7f;

    // Ορισμός του κέντρου της σπείρας στο κέντρο του παραθύρου
```

```
        float centerX = WINDOW_WIDTH / 2;
        float centerY = WINDOW_HEIGHT / 2;

        // Κλήση της συνάρτησης σχεδίασης της σπείρας του Αρχιμήδη
        // Παράμετροι:
        // - 300 σημεία/επανάληψεις
        // - α = 5 (αρχικό μέγεθος σπείρας)
        // - β = 2 (ρυθμός ανάπτυξης)
        drawArchimedeanSpiral(centerX, centerY, 300, 5, 2, brush);
    }

}
namespace CubicFractal {
    // Ορισμός των διαστάσεων του παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;
    const int MAX_DEPTH = 4; // Ορισμός του μέγιστου βάθους αναδρομής για το fractal

    /**
     * @brief Συνάρτηση σχεδίασης ενός τετραγώνου με συγκεκριμένες συντεταγμένες και μέγεθος.
     *
     * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου.
     * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου.
     * @param side Το μήκος κάθε πλευράς του τετραγώνου.
     * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
     */
    void drawSquare(float x, float y, float side, const graphics::Brush& brush) {
        // Σχεδίαση τετραγώνου χρησιμοποιώντας την κεντρική θέση x, y και πλευρά side
        graphics::drawRect(x, y, side, side, brush);
    }
```

```
/**
 * @brief Αναδρομική συνάρτηση που δημιουργεί το "Cubic Fractal" σχεδιάζοντας τετράγωνα.
 *
 * @param x Η x-συντεταγμένη του κέντρου του τετραγώνου.
 * @param y Η y-συντεταγμένη του κέντρου του τετραγώνου.
 * @param side Το μήκος κάθε πλευράς του τετραγώνου για το τρέχον επίπεδο.
 * @param depth Το τρέχον επίπεδο βάθους αναδρομής.
 * @param brush Το πινέλο που καθορίζει το χρώμα και το στυλ σχεδίασης.
 */
void drawCubicFractal(float x, float y, float side, int depth, const graphics::Brush& brush) {
    if (depth == 0) return; // Διακοπή αναδρομής όταν φτάσουμε στο μέγιστο βάθος

    // Σχεδίαση του κεντρικού τετραγώνου για το τρέχον επίπεδο
    drawSquare(x, y, side, brush);

    // Υπολογισμός νέου μεγέθους για τα υποτετράγωνα
    float newSide = side / 3.0f;
    int nextDepth = depth - 1;

    // Σχεδίαση 8 περιφερειακών τετραγώνων γύρω από το κεντρικό τετράγωνο
    drawCubicFractal(x - newSide, y - newSide, newSide, nextDepth, brush); // Αριστερά-πάνω
    drawCubicFractal(x, y - newSide, newSide, nextDepth, brush);           // Κέντρο-πάνω
    drawCubicFractal(x + newSide, y - newSide, newSide, nextDepth, brush); // Δεξιά-πάνω

    drawCubicFractal(x - newSide, y, newSide, nextDepth, brush);           // Αριστερά-κέντρο
    drawCubicFractal(x + newSide, y, newSide, nextDepth, brush);           // Δεξιά-κέντρο

    drawCubicFractal(x - newSide, y + newSide, newSide, nextDepth, brush); // Αριστερά-κάτω
    drawCubicFractal(x, y + newSide, newSide, nextDepth, brush);           // Κέντρο-κάτω
    drawCubicFractal(x + newSide, y + newSide, newSide, nextDepth, brush); // Δεξιά-κάτω
}
```

```
    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
     *        Καθορίζει το κεντρικό τετράγωνο και ξεκινά την αναδρομική σχεδίαση του fractal.
     */
    void draw() {
        // Ρύθμιση του πινέλου σχεδίασης με μπλε-πράσινη απόχρωση
        graphics::Brush brush;
        brush.fill_color[0] = 0.1f;
        brush.fill_color[1] = 0.4f;
        brush.fill_color[2] = 0.7f;

        // Τοποθέτηση του κεντρικού τετραγώνου στο κέντρο του παραθύρου
        float centerX = WINDOW_WIDTH / 2;
        float centerY = WINDOW_HEIGHT / 2;
        float initialSize = 400.0f; // Αρχικό μέγεθος του κεντρικού τετραγώνου

        // Κλήση της αναδρομικής συνάρτησης σχεδίασης fractal
        drawCubicFractal(centerX, centerY, initialSize, MAX_DEPTH, brush);
    }

}
namespace TorusKnotFractal {
    // Ορισμός παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;
    const int MAX_ITERATIONS = 5;  // Αριθμός επαναλήψεων για τη σχεδίαση του fractal

    /**
     * @brief Συνάρτηση που σχεδιάζει τον κόμπο Torus Knot σε σπειροειδές fractal μοτίβο.
     *
     * @param centerX Η x-συντεταγμένη του κέντρου.
```

```
* @param centerY Η y-συντεταγμένη του κέντρου.
* @param radius Η αρχική ακτίνα του κόμβου.
* @param iterations Το βάθος αναδρομής για το fractal.
* @param lineWidth Το πάχος της γραμμής για τη σχεδίαση.
*/
void drawTorusKnot(float centerX, float centerY, float radius, int iterations, float lineWidth) {
    if (iterations <= 0) return;  // Βάση αναδρομής: σταματάμε όταν οι επαναλήψεις μηδενιστούν

    // Ρύθμιση πινέλου με ελαφριά αδιαφάνεια και μεταβαλλόμενο χρώμα
    graphics::Brush brush;
    brush.fill_opacity = 0.8f;
    brush.fill_color[0] = 0.5f + 0.1f * iterations;  // Ενίσχυση κόκκινου χρώματος ανά επανάληψη
    brush.fill_color[1] = 0.3f;
    brush.fill_color[2] = 0.8f - 0.1f * iterations;  // Μείωση μπλε απόχρωσης ανά επανάληψη

    // Παράμετροι του κόμβου Torus Knot
    float p = 3;       // Αριθμός κυρίων καμπυλών
    float q = 2;       // Πλήθος περιστροφών γύρω από τον τόρο
    int points = 100;  // Αριθμός σημείων που σχηματίζουν τον κόμπο

    // Σχεδίαση του κόμβου Torus Knot
    for (int i = 0; i < points; ++i) {
        // Γωνία τρέχοντος σημείου και υπολογισμός συντεταγμένων (x, y)
        float angle = 2 * M_PI * i / points;
        float x = centerX + radius * cos(p * angle) * cos(angle);
        float y = centerY + radius * cos(p * angle) * sin(angle);

        // Υπολογισμός επόμενου σημείου για τη σχεδίαση γραμμής
        float nextAngle = 2 * M_PI * (i + 1) / points;
        float nextX = centerX + radius * cos(p * nextAngle) * cos(nextAngle);
```

```
            float nextY = centerY + radius * cos(p * nextAngle) * sin(nextAngle);

            // Σχεδίαση γραμμής μεταξύ των σημείων
            graphics::drawLine(x, y, nextX, nextY, brush);
        }

        // Αναδρομική κλήση για το επόμενο επίπεδο του fractal, με μικρότερη ακτίνα και πάχος γραμμής
        drawTorusKnot(centerX, centerY, radius * 0.6f, iterations - 1, lineWidth * 0.7f);
    }

    /**
     * @brief Συνάρτηση σχεδίασης για το παράθυρο SGG.
     *        Ορίζει το κέντρο και την αρχική ακτίνα και καλεί τη συνάρτηση σχεδίασης του κόμβου.
     */
    void draw() {
        // Ορισμός κέντρου και αρχικής ακτίνας
        float centerX = WINDOW_WIDTH / 2;
        float centerY = WINDOW_HEIGHT / 2;
        float initialRadius = 300.0f;

        // Κλήση της συνάρτησης σχεδίασης του Torus Knot Fractal
        drawTorusKnot(centerX, centerY, initialRadius, MAX_ITERATIONS, 2.0f);
    }

}
namespace KnotTheoryFractal{
    // Ορισμός των διαστάσεων του παραθύρου και του μέγιστου αριθμού επαναλήψεων
const float WINDOW_WIDTH = 600;
const float WINDOW_HEIGHT = 600;
const int MAX_ITERATIONS = 6;  // Μέγιστος αριθμός επαναλήψεων για το fractal
```

```
/**
 * @brief Αναδρομική συνάρτηση που σχεδιάζει το fractal κόμβου με περιστροφή και σμίκρυνση.
 *
 * @param centerX Η x-συντεταγμένη του κέντρου.
 * @param centerY Η y-συντεταγμένη του κέντρου.
 * @param radius Η ακτίνα του αρχικού κύκλου.
 * @param iterations Το βάθος αναδρομής του fractal.
 * @param rotationAngle Η γωνία περιστροφής για κάθε επίπεδο.
 */
void drawKnotFractal(float centerX, float centerY, float radius, int iterations, float rotationAngle) {
    // Τερματίζουμε την αναδρομή όταν φτάσουμε σε μηδενικές επαναλήψεις
    if (iterations <= 0) return;

    // Ρύθμιση πινέλου και χρώματος με ελαφριά αδιαφάνεια
    graphics::Brush brush;
    brush.fill_opacity = 0.7f;
    brush.fill_color[0] = 0.3f + 0.1f * iterations;  // Χρωματική προσαρμογή για κάθε επίπεδο
    brush.fill_color[1] = 0.2f;
    brush.fill_color[2] = 0.5f + 0.1f * iterations;

    // Ορισμός των σημείων της καμπύλης
    int points = 100;

    // Σχεδίαση της καμπύλης που αποτελεί τον κόμβο
    for (int i = 0; i < points; ++i) {
        // Υπολογισμός της γωνίας του τρέχοντος σημείου και των συντεταγμένων (x, y)
        float angle = 2 * M_PI * i / points;
        float x = centerX + radius * cos(3 * angle) * cos(angle + rotationAngle);
        float y = centerY + radius * cos(3 * angle) * sin(angle + rotationAngle);

        // Υπολογισμός της επόμενης γωνίας και των επόμενων συντεταγμένων (nextX, nextY)
        float nextAngle = 2 * M_PI * (i + 1) / points;
```

```
        float nextX = centerX + radius * cos(3 * nextAngle) * cos(nextAngle + rotationAngle);
        float nextY = centerY + radius * cos(3 * nextAngle) * sin(nextAngle + rotationAngle);

        // Σχεδίαση γραμμής μεταξύ των τρεχόντων και των επόμενων σημείων
        graphics::drawLine(x, y, nextX, nextY, brush);
    }

    // Αναδρομική κλήση για το επόμενο, μικρότερο επίπεδο του fractal με περιστροφή
    drawKnotFractal(centerX, centerY, radius * 0.7f, iterations - 1, rotationAngle + M_PI / 4);
}

/**
 * @brief Συνάρτηση σχεδίασης για το παράθυρο SGG.
 *        Καλεί την `drawKnotFractal` για τη σχεδίαση του fractal κόμβου.
 */
void draw() {
    // Αρχικές συντεταγμένες για το κέντρο του fractal και η ακτίνα του
    float centerX = WINDOW_WIDTH / 2;
    float centerY = WINDOW_HEIGHT / 2;
    float initialRadius = 200.0f;

    // Κλήση της συνάρτησης για τη σχεδίαση του κόμβου
    drawKnotFractal(centerX, centerY, initialRadius, MAX_ITERATIONS, 0);
}

    }
    namespace ButterflyEffectFractal {
        // Ορισμός διαστάσεων παραθύρου
        const float WINDOW_WIDTH = 800;
        const float WINDOW_HEIGHT = 600;

        /**
```

```
* @brief Αναδρομική συνάρτηση για τη σχεδίαση του Butterfly Effect Fractal.
*
* @param x Η x-συντεταγμένη του αρχικού σημείου.
* @param y Η y-συντεταγμένη του αρχικού σημείου.
* @param length Το μήκος της γραμμής που θα σχεδιαστεί σε κάθε επίπεδο.
* @param angle Η γωνία με την οποία θα σχεδιαστεί η γραμμή.
* @param depth Το βάθος της αναδρομής που απομένει.
* @param brush Το χρώμα της γραμμής.
*/
void drawButterflyEffectFractal(float x, float y, float length, float angle, int depth, const graphics::Brush& brush) {
    if (depth == 0) return; // Βάση της αναδρομής: αν το βάθος είναι 0, σταματάμε

    // Υπολογισμός συντεταγμένων για το επόμενο σημείο της γραμμής
    float newX = x + length * cos(angle);
    float newY = y + length * sin(angle);

    // Σχεδίαση γραμμής από το τρέχον σημείο στο νέο σημείο
    graphics::drawLine(x, y, newX, newY, brush);

    // Παράμετροι για τη διακλάδωση του fractal
    float branchAngle = M_PI / 3; // Γωνία διακλάδωσης (60 μοίρες)
    float lengthReduction = 0.7f; // Μείωση μήκους γραμμής σε κάθε επίπεδο

    // Αναδρομικές κλήσεις για τα "φτερά" της πεταλούδας
    // Κλήση για το αριστερό "φτερό" με περιστροφή προς τα αριστερά
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle + branchAngle, depth - 1, brush);

    // Κλήση για το δεξί "φτερό" με περιστροφή προς τα δεξιά
    drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle - branchAngle, depth - 1, brush);
```

```
        // Πρόσθετες διακλαδώσεις για να δημιουργηθεί πιο πλούσια δομή "πεταλούδας"
        drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle + branchAngle / 2, depth - 1, brush);
        drawButterflyEffectFractal(newX, newY, length * lengthReduction, angle - branchAngle / 2, depth - 1, brush);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG για να ζωγραφίσει το fractal.
     *        Καθαρίζει το φόντο και καλεί τη `drawButterflyEffectFractal`.
     */
    void draw() {
        // Καθαρισμός φόντου σε μαύρο χρώμα
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Ορισμός χρώματος για το fractal
        graphics::Brush fractalBrush;
        fractalBrush.fill_color[0] = 0.0f;
        fractalBrush.fill_color[1] = 0.0f;
        fractalBrush.fill_color[2] = 1.0f; // Μπλε χρώμα για το fractal

        // Εκκίνηση της σχεδίασης από το κέντρο του παραθύρου
        drawButterflyEffectFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 0, 10, fractalBrush);
    }

}
```

```
namespace LorenzAttractorFractal{
    // Διαστάσεις του παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 750;

    // Μέγιστος αριθμός σημείων που θα σχεδιαστούν
    const int MAX_POINTS = 5000;

    // Παράμετροι Lorenz Attractor
    const float sigma = 10.0f;       // Παράμετρος σύζευξης των εξισώσεων
    const float rho = 28.0f;         // Παράμετρος που καθορίζει το χάος στο σύστημα
    const float beta = 8.0f / 3.0f;  // Παράμετρος απόσβεσης

    // Αρχικές συνθήκες για το σύστημα Lorenz
    float x = 0.1f, y = 0.0f, z = 0.0f; // Αρχικές συντεταγμένες
    float dt = 0.01f;                   // Βήμα χρόνου για την αριθμητική ολοκλήρωση

    // Δομή για την αποθήκευση των σημείων σε 2D
    std::vector<std::pair<float, float>> points;

    /**
     * @brief Υπολογισμός των σημείων του Lorenz Attractor.
     *
     * Αυτή η συνάρτηση υλοποιεί τις εξισώσεις Lorenz για να υπολογίσει
     * διαδοχικά σημεία του ελκυστή. Κάθε σημείο προστίθεται σε έναν πίνακα για
     * μεταγενέστερη σχεδίαση.
     */
    void calculateLorenzAttractor() {
        for (int i = 0; i < MAX_POINTS; ++i) {
            // Υπολογισμός παραγώγων σύμφωνα με τις εξισώσεις Lorenz
            float dx = sigma * (y - x) * dt;             // Παραγωγός για x
            float dy = (x * (rho - z) - y) * dt;         // Παραγωγός για y
```

```
            float dz = (x * y - beta * z) * dt;          // Παραγωγός για z

            // Ενημέρωση των συντεταγμένων x, y, z
            x += dx;
            y += dy;
            z += dz;

            // Προσθήκη του σημείου στην προβολή 2D για σχεδίαση
            points.emplace_back(WINDOW_WIDTH / 2 + x * 8, WINDOW_HEIGHT / 2 - z * 8);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
     *
     * Η συνάρτηση `draw` χρησιμοποιείται για να σχεδιάσει όλα τα σημεία
     * του ελκυστή Lorenz σε μια 2D προβολή.
     */
    void draw() {
        // Ορισμός του χρώματος σχεδίασης
        graphics::Brush brush;
        brush.fill_color[0] = 1.0f;  // Κόκκινο
        brush.fill_color[1] = 0.3f;  // Πράσινο
        brush.fill_color[2] = 0.0f;  // Μπλε
        brush.fill_opacity = 0.7f;   // Αδιαφάνεια

        // Σχεδίαση όλων των σημείων Lorenz Attractor
        for (auto& point : points) {
            graphics::drawDisk(point.first, point.second, 1.0f, brush); // Κάθε σημείο σχεδιάζεται ως μικρός κύκλος
        }
    }
```

```
}
namespace RingFractal {
    // Διαστάσεις παραθύρου γραφικών
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Αναδρομική συνάρτηση για τη σχεδίαση ενός fractal δακτυλίων.
     *
     * Σχεδιάζει έναν κύκλο και περιμετρικά του άλλους μικρότερους κύκλους.
     * Κάθε μικρότερος κύκλος αναπαράγει το ίδιο μοτίβο με μειωμένη ακτίνα.
     *
     * @param x Συντεταγμένη X του κεντρικού κύκλου
     * @param y Συντεταγμένη Y του κεντρικού κύκλου
     * @param radius Ακτίνα του κύκλου
     * @param depth Βάθος αναδρομής για το fractal
     */
    void drawRingsFractal(float x, float y, float radius, int depth) {
        if (depth <= 0) return;  // Βάση αναδρομής: σταματάμε αν το βάθος είναι 0

        // Ορισμός των ιδιοτήτων σχεδίασης του κύκλου
        graphics::Brush brush;
        brush.fill_opacity = 0.0f;        // Διαφανής πλήρωση
        brush.outline_opacity = 0.8f;     // Ημιδιαφανές περίγραμμα
        brush.outline_width = 2.0f;       // Πλάτος περιγράμματος

        // Σχεδίαση του κεντρικού κύκλου
        graphics::drawDisk(x, y, radius, brush);

        // Αριθμός μικρότερων κύκλων γύρω από τον κεντρικό
        int numCircles = 6;
```

```
        // Υπολογισμός γωνίας και τοποθέτηση μικρότερων κύκλων περιμετρικά του κεντρικού
        for (int i = 0; i < numCircles; i++) {
            float angle = (2 * M_PI / numCircles) * i;     // Γωνία τοποθέτησης σε ακτίνια
            float newX = x + radius * cos(angle);          // Νέα θέση X του μικρότερου κύκλου
            float newY = y + radius * sin(angle);          // Νέα θέση Y του μικρότερου κύκλου

            // Αναδρομική κλήση για τη σχεδίαση του μικρότερου κύκλου με μειωμένη ακτίνα και βάθος
            drawRingsFractal(newX, newY, radius / 2, depth - 1);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών SGG.
     *
     * Η `draw` χρησιμοποιείται για τη σχεδίαση του κεντρικού φράκταλ στο κέντρο του παραθύρου.
     */
    void draw() {
        // Ορισμός του χρώματος φόντου
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;   // Μαύρο χρώμα
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης για τη σχεδίαση του fractal των δακτυλίων
        drawRingsFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, 4);  // Αρχική ακτίνα 150 και βάθος 4
    }
```

```
}
namespace PythagoreanTreeFractal {
    // Διαστάσεις παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;
    const float INITIAL_SIDE = 150.0f; // Πλευρά του αρχικού τετραγώνου

    /**
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το φράκταλ του Πυθαγορείου Δέντρου.
     *
     * Σε κάθε επίπεδο της αναδρομής, δύο μικρότερα τετράγωνα προστίθενται
     * στις πάνω γωνίες του τρέχοντος τετραγώνου, δημιουργώντας τη δομή του φράκταλ.
     *
     * @param x Συντεταγμένη x του κέντρου του τετραγώνου
     * @param y Συντεταγμένη y του κέντρου του τετραγώνου
     * @param side Η πλευρά του τετραγώνου
     * @param angle Γωνία περιστροφής του τετραγώνου
     * @param depth Το βάθος της αναδρομής
     */
    void drawPythagoreanTree(float x, float y, float side, float angle, int depth) {
        if (depth <= 0) return; // Βάση αναδρομής

        // Δημιουργία πινέλου για τη σχεδίαση των τετραγώνων
        graphics::Brush brush;
        brush.fill_opacity = 1.0f;
        brush.fill_color[0] = 0.0f; // Πράσινη απόχρωση
        brush.fill_color[1] = 0.5f + 0.5f * (float)depth / 10;
        brush.fill_color[2] = 0.0f;

        // Υπολογισμός των κορυφών του τετραγώνου με βάση το κέντρο και τη γωνία περιστροφής
        float halfSide = side / 2.0f;
        float rad = angle * M_PI / 180.0f;
```

```
float x0 = x - halfSide * cos(rad) - halfSide * sin(rad);
float y0 = y - halfSide * sin(rad) + halfSide * cos(rad);

float x1 = x + halfSide * cos(rad) - halfSide * sin(rad);
float y1 = y + halfSide * sin(rad) + halfSide * cos(rad);

float x2 = x + halfSide * cos(rad) + halfSide * sin(rad);
float y2 = y + halfSide * sin(rad) - halfSide * cos(rad);

float x3 = x - halfSide * cos(rad) + halfSide * sin(rad);
float y3 = y - halfSide * sin(rad) - halfSide * cos(rad);

// Σχεδίαση των γραμμών που σχηματίζουν το τετράγωνο
graphics::drawLine(x0, y0, x1, y1, brush);
graphics::drawLine(x1, y1, x2, y2, brush);
graphics::drawLine(x2, y2, x3, y3, brush);
graphics::drawLine(x3, y3, x0, y0, brush);

// Υπολογισμός νέου μεγέθους πλευράς για τα υπο-τετράγωνα και νέες γωνίες
float newSide = side * 0.707; // Μείωση μεγέθους κατά √2 / 2
float leftAngle = angle - 45;
float rightAngle = angle + 45;

// Συντεταγμένες για τα κέντρα των νέων τετραγώνων
float leftX = x3;
float leftY = y3;
float rightX = x2;
float rightY = y2;

// Αναδρομική κλήση για τη σχεδίαση του αριστερού και του δεξιού τετραγώνου
drawPythagoreanTree(leftX, leftY, newSide, leftAngle, depth - 1);
```

```
        drawPythagoreanTree(rightX, rightY, newSide, rightAngle, depth - 1);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
     *
     * Η συνάρτηση `draw` σχεδιάζει το φόντο και καλεί τη συνάρτηση `drawPythagoreanTree`
     * για να ξεκινήσει η αναδρομική σχεδίαση του Πυθαγόρειου Δέντρου από το κάτω κέντρο.
     */
    void draw() {
        // Ορισμός φόντου
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f; // Μαύρο φόντο
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης fractal από το κάτω μέρος του παραθύρου
        drawPythagoreanTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 100,
INITIAL_SIDE, 0, 10);
    }

}
namespace FlowerFractal {
    // Διαστάσεις παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το φράκταλ του λουλουδιού.
     *
```

```
     * Η συνάρτηση σχεδιάζει έναν κύκλο σε κάθε πέταλο και κατόπιν αναδρομικά
     * καλεί τον εαυτό της για κάθε νέο πέταλο, σχηματίζοντας μικρότερα πέταλα.
     *
     * @param x Η συντεταγμένη x του κέντρου του κύριου λουλουδιού
     * @param y Η συντεταγμένη y του κέντρου του κύριου λουλουδιού
     * @param radius Η ακτίνα του κύκλου (πέταλου)
     * @param depth Το βάθος της αναδρομής, καθορίζει το επίπεδο λεπτομέρειας
     * @param petals Ο αριθμός των πετάλων για κάθε επίπεδο αναδρομής
     */
    void drawFlowerFractal(float x, float y, float radius, int depth, int petals) {
        if (depth <= 0) return; // Βάση αναδρομής

        // Ορισμός πινέλου για τα πέταλα
        graphics::Brush petalBrush;
        petalBrush.fill_opacity = 0.6f;  // Διαφάνεια πετάλων
        petalBrush.fill_color[0] = 1.0f; // Χρώμα πετάλου (κόκκινο-ροζ απόχρωση που αλλάζει με το βάθος)
        petalBrush.fill_color[1] = 0.4f + 0.2f * depth / 5;
        petalBrush.fill_color[2] = 0.7f - 0.1f * depth;

        // Υπολογισμός γωνίας για κάθε πέταλο ώστε να κατανέμονται ομοιόμορφα γύρω από το κέντρο
        float angleIncrement = 360.0f / petals;
        for (int i = 0; i < petals; i++) {
            float angle = i * angleIncrement * M_PI / 180.0f;

            // Υπολογισμός θέσης του πετάλου με βάση την ακτίνα και τη γωνία
            float petalX = x + radius * cos(angle);
            float petalY = y + radius * sin(angle);

            // Σχεδίαση του πετάλου ως δίσκος
            graphics::drawDisk(petalX, petalY, radius * 0.5f, petalBrush);
```

```
            // Αναδρομική κλήση για το κάθε πέταλο, μειώνοντας το μέγεθος και το βάθος
            drawFlowerFractal(petalX, petalY, radius * 0.5f, depth - 1, petals);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
     *
     * Η συνάρτηση καθαρίζει το παράθυρο με λευκό φόντο και καλεί τη συνάρτηση
     * `drawFlowerFractal` για να σχεδιάσει το φράκταλ του λουλουδιού.
     */
    void draw() {
        // Καθαρισμός του παραθύρου με λευκό φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 1.0f; // Λευκό
        bg.fill_color[1] = 1.0f;
        bg.fill_color[2] = 1.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης σχεδίασης του Flower Fractal
        drawFlowerFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 5, 6);
    }

}
namespace CrystalFractal {
    // Διαστάσεις παραθύρου
    const float WINDOW_WIDTH = 600;
    const float WINDOW_HEIGHT = 600;

    /**
```

```
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το Crystal Fractal
     *
     * Η συνάρτηση δημιουργεί ένα ακτινικό fractal όπου κάθε κλάδος δημιουργεί
     * μικρότερους κλάδους σε προκαθορισμένες γωνίες, σχηματίζοντας έτσι έναν
     * συμμετρικό σχηματισμό.
     *
     * @param x Η συντεταγμένη x του κέντρου του fractal
     * @param y Η συντεταγμένη y του κέντρου του fractal
     * @param length Το μήκος κάθε κλάδου του fractal
     * @param depth Το βάθος της αναδρομής, καθορίζει το επίπεδο λεπτομέρειας
     * @param branches Ο αριθμός των κλάδων που εκτείνονται από κάθε σημείο
     */
    void drawCrystalFractal(float x, float y, float length, int depth, int branches) {
        if (depth <= 0) return;  // Βάση αναδρομής

        // Ορισμός πινέλου για την σχεδίαση του κλάδου
        graphics::Brush crystalBrush;
        crystalBrush.fill_color[0] = 0.5f + 0.5f * (depth % 2);  // Εναλλαγή χρώματος ανά επίπεδο
        crystalBrush.fill_color[1] = 0.8f - 0.2f * depth / 5;
        crystalBrush.fill_color[2] = 1.0f - 0.1f * depth;

        // Υπολογισμός γωνίας ανάμεσα στους κλάδους ώστε να τοποθετηθούν ομοιόμορφα γύρω από το κέντρο
        float angleIncrement = 360.0f / branches;

        // Σχεδίαση και αναδρομική κλήση για κάθε κλάδο
        for (int i = 0; i < branches; i++) {
            // Υπολογισμός γωνίας και νέων συντεταγμένων για κάθε κλάδο
            float angle = i * angleIncrement * M_PI / 180.0f;
            float newX = x + length * cos(angle);
            float newY = y + length * sin(angle);
```

```
            // Σχεδίαση του κλάδου ως γραμμή από το σημείο (x, y) στο (newX, newY)
            graphics::drawLine(x, y, newX, newY, crystalBrush);

            // Αναδρομική κλήση για την δημιουργία μικρότερων κλάδων στο τέλος κάθε γραμμής
            drawCrystalFractal(newX, newY, length * 0.5f, depth - 1, branches);
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών
     *
     * Η συνάρτηση καθαρίζει το παράθυρο με μαύρο φόντο και καλεί τη συνάρτηση
     * `drawCrystalFractal` για να ξεκινήσει η σχεδίαση του fractal από το κέντρο του παραθύρου.
     */
    void draw() {
        // Καθαρισμός του παραθύρου με μαύρο φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης σχεδίασης του Crystal Fractal
        drawCrystalFractal(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, 5, 6);
    }

}
namespace FibonacciSpiralFractal {
    // Ορισμός διαστάσεων παραθύρου και χρυσής αναλογίας
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;
```

```
const float GOLDEN_RATIO = 1.618f;

/**
 * @brief Υπολογισμός της ακολουθίας Fibonacci
 *
 * Η συνάρτηση αυτή δημιουργεί έναν πίνακα με τιμές Fibonacci, οι οποίες χρησιμοποιούνται
 * ως διαδοχικά μήκη πλευράς για τη σχεδίαση των τετραγώνων και τόξων του fractal.
 *
 * @param terms Ο αριθμός των όρων της ακολουθίας Fibonacci που θα υπολογιστούν.
 * @param scaleFactor Συντελεστής κλίμακας για το μήκος των πλευρών.
 * @return Ένας πίνακας με τις τιμές Fibonacci προσαρμοσμένες στον συντελεστή κλίμακας.
 */
std::vector<float> generateFibonacci(int terms, float scaleFactor = 1.0f) {
    std::vector<float> fibonacci = { scaleFactor, scaleFactor * GOLDEN_RATIO };
    for (int i = 2; i < terms; i++) {
        fibonacci.push_back(fibonacci[i - 1] + fibonacci[i - 2]);
    }
    return fibonacci;
}

/**
 * @brief Αναδρομική συνάρτηση για την σχεδίαση της σπείρας Fibonacci.
 *
 * Η συνάρτηση χρησιμοποιεί την ακολουθία Fibonacci για να σχεδιάσει διαδοχικά τετράγωνα
 * και τόξα, τα οποία διατάσσονται σε μια σπείρα ακολουθώντας την αναλογία του χρυσού αριθμού.
 *
 * @param x Η αρχική συντεταγμένη x της σπείρας.
 * @param y Η αρχική συντεταγμένη y της σπείρας.
 * @param terms Ο αριθμός των όρων της ακολουθίας Fibonacci που θα χρησιμοποιηθούν.
 * @param initialLength Το αρχικό μήκος πλευράς για το πρώτο τετράγωνο.
 */
```

```
void drawFibonacciSpiral(float x, float y, int terms, float initialLength) {
    // Δημιουργία της ακολουθίας Fibonacci
    std::vector<float> fibonacci = generateFibonacci(terms, initialLength);

    float angle = 0;  // Αρχική γωνία
    graphics::Brush spiralBrush;
    spiralBrush.fill_color[0] = 0.1f;
    spiralBrush.fill_color[1] = 0.6f;
    spiralBrush.fill_color[2] = 1.0f;

    // Σχεδίαση της σπείρας
    for (int i = 0; i < terms; i++) {
        float length = fibonacci[i];  // Μήκος πλευράς για κάθε όρο
        float nextX = x + length * cos(angle * M_PI / 180.0f);  // Νέα θέση x
        float nextY = y + length * sin(angle * M_PI / 180.0f);  // Νέα θέση y

        // Σχεδίαση τετραγώνου Fibonacci
        graphics::drawRect(
            x + length / 2 * cos(angle * M_PI / 180.0f),
            y + length / 2 * sin(angle * M_PI / 180.0f),
            length, length, spiralBrush
        );

        // Σχεδίαση τόξου με γωνία 90° για κάθε όρο της ακολουθίας
        graphics::drawSector(x, y, length, length * 1.1f, angle, angle + 90, spiralBrush);

        // Ενημέρωση συντεταγμένων και γωνίας για το επόμενο τετράγωνο
        x = nextX;
        y = nextY;
        angle += 90;  // Περιστροφή 90° για επόμενη σπείρα
    }
}
```

```
    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη γραφικών.
     *
     * Η συνάρτηση αυτή καθαρίζει το φόντο και ξεκινά την αναδρομική σχεδίαση
     * της σπείρας Fibonacci από το κέντρο του παραθύρου.
     */
    void draw() {
        // Καθαρισμός του παραθύρου με λευκό φόντο
        graphics::Brush bg;
        bg.fill_color[0] = 1.0f;
        bg.fill_color[1] = 1.0f;
        bg.fill_color[2] = 1.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Κλήση της συνάρτησης σχεδίασης Fibonacci σπείρας από το κέντρο της οθόνης
        drawFibonacciSpiral(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 6, 12);
    }

}
namespace TriangularSpiralFractal {
    // Ορισμός παραμέτρων παραθύρου
    const float WINDOW_WIDTH = 700;
    const float WINDOW_HEIGHT = 700;

    /**
     * @brief Σχεδιάζει ένα ισόπλευρο τρίγωνο με βάση το σημείο εκκίνησης, το μήκος πλευράς και τη γωνία.
     *
     * @param x Συντεταγμένη x για το σημείο εκκίνησης.
     * @param y Συντεταγμένη y για το σημείο εκκίνησης.
```

```
     * @param sideLength Το μήκος πλευράς του τριγώνου.
     * @param angle Η γωνία περιστροφής του τριγώνου σε ακτίνια.
     * @param brush Το αντικείμενο πινέλου για την απόδοση του τριγώνου.
     */
    void drawTriangle(float x, float y, float sideLength, float angle, const graphics::Brush& brush)
{
        // Υπολογισμός ύψους ισόπλευρου τριγώνου
        float height = sideLength * sqrt(3) / 2;

        // Υπολογισμός των τριών κορυφών του τριγώνου με βάση τη γωνία
        float x1 = x + sideLength * cos(angle);
        float y1 = y + sideLength * sin(angle);

        float x2 = x + sideLength * cos(angle + 120 * M_PI / 180.0f);
        float y2 = y + sideLength * sin(angle + 120 * M_PI / 180.0f);

        // Σχεδίαση του τριγώνου με τις τρεις πλευρές
        graphics::drawLine(x, y, x1, y1, brush);
        graphics::drawLine(x1, y1, x2, y2, brush);
        graphics::drawLine(x2, y2, x, y, brush);
    }

    /**
     * @brief Σχεδιάζει το Triangular Spiral Fractal.
     *
     * Η σπείρα σχηματίζεται από μια αλληλουχία τριγώνων που περιστρέφονται και αυξάνονται σε μέγεθος,
     * δημιουργώντας ένα fractal σπειροειδούς μορφής.
     *
     * @param startX Συντεταγμένη x για το σημείο εκκίνησης της σπείρας.
     * @param startY Συντεταγμένη y για το σημείο εκκίνησης της σπείρας.
     * @param depth Ο αριθμός των τριγώνων στη σπείρα.
```

```
     * @param initialSideLength Το αρχικό μήκος πλευράς του πρώτου τριγώνου.
     * @param angleIncrement Η γωνία περιστροφής ανά τρίγωνο σε μοίρες.
     */
    void drawTriangularSpiral(float startX, float startY, int depth, float initialSideLength, float angleIncrement) {
        graphics::Brush brush;
        brush.fill_color[0] = 0.2f;
        brush.fill_color[1] = 0.6f;
        brush.fill_color[2] = 1.0f;

        float currentAngle = 0;          // Αρχική γωνία περιστροφής
        float currentX = startX;         // Αρχική θέση x
        float currentY = startY;         // Αρχική θέση y
        float sideLength = initialSideLength; // Αρχικό μήκος πλευράς

        // Αναδρομική σχεδίαση τριγώνων για τη δημιουργία της σπείρας
        for (int i = 0; i < depth; i++) {
            // Σχεδίαση του τρέχοντος τριγώνου
            drawTriangle(currentX, currentY, sideLength, currentAngle, brush);

            // Υπολογισμός νέας θέσης και γωνίας για το επόμενο τρίγωνο
            currentX += sideLength * cos(currentAngle);
            currentY += sideLength * sin(currentAngle);

            // Αύξηση πλευράς και περιστροφή γωνίας για τη σπείρα
            sideLength += initialSideLength * 0.1f;
            currentAngle += angleIncrement * M_PI / 180.0f;
        }
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
```

```
     *
     * Καθαρίζει το φόντο και ξεκινά τη σχεδίαση του Triangular Spiral Fractal
     * από το κέντρο του παραθύρου.
     */
    void draw() {
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f;  // Μαύρο φόντο
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH, WINDOW_HEIGHT, bg);

        // Εκκίνηση σχεδίασης της σπείρας από το κέντρο του παραθύρου
        drawTriangularSpiral(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 400, 2, 15);
    }

}
namespace BinaryTreeFractal {
    // Ορισμός παραμέτρων παραθύρου
    const float WINDOW_WIDTH = 800;
    const float WINDOW_HEIGHT = 600;

    /**
     * @brief Αναδρομική συνάρτηση που σχεδιάζει το δυαδικό δέντρο fractal.
     *
     * Η συνάρτηση σχεδιάζει ένα κλαδί, στη συνέχεια καλείται αναδρομικά για να σχεδιάσει δύο
     * μικρότερα κλαδιά στα άκρα, καθένα σε γωνία απόκλισης από το προηγούμενο.
     *
     * @param x Συντεταγμένη x για την αρχή του κλαδιού.
     * @param y Συντεταγμένη y για την αρχή του κλαδιού.
     * @param length Το μήκος του τρέχοντος κλαδιού.
     * @param angle Η γωνία με την οποία εκτείνεται το κλαδί.
```

```
     * @param depth Το βάθος της αναδρομής, καθορίζει το πλήθος των επιπέδων του δέντρου.
     * @param brush Το πινέλο που χρησιμοποιείται για τη σχεδίαση του κλαδιού.
     */
    void drawBinaryTree(float x, float y, float length, float angle, int depth, const graphics::Brush& brush) {
        if (depth == 0) return;  // Βάση της αναδρομής - σταματάμε όταν το βάθος είναι 0

        // Υπολογισμός των νέων συντεταγμένων του άκρου του κλαδιού
        float newX = x + length * cos(angle);
        float newY = y - length * sin(angle);  // Αφαιρούμε για να σχεδιάσουμε προς τα πάνω

        // Σχεδίαση του κλαδιού
        graphics::drawLine(x, y, newX, newY, brush);

        // Αναδρομικές κλήσεις για τα δύο κλαδιά με αλλαγές στη γωνία και το μήκος
        float branchAngle = M_PI / 6;           // Γωνία διακλάδωσης (30 μοίρες)
        float branchLengthFactor = 0.7f;        // Μείωση του μήκους του κλαδιού σε κάθε επίπεδο

        // Κλήση για το αριστερό και δεξί κλαδί
        drawBinaryTree(newX, newY, length * branchLengthFactor, angle - branchAngle, depth - 1, brush);
        drawBinaryTree(newX, newY, length * branchLengthFactor, angle + branchAngle, depth - 1, brush);
    }

    /**
     * @brief Συνάρτηση σχεδίασης που καλείται από τη βιβλιοθήκη SGG.
     *
     * Καθαρίζει το φόντο και ξεκινά τη σχεδίαση του Binary Tree Fractal
     * από το κέντρο της βάσης του παραθύρου.
     */
    void draw() {
        // Καθαρισμός του φόντου
```

```
        graphics::Brush bg;
        bg.fill_color[0] = 0.0f; // Μαύρο φόντο
        bg.fill_color[1] = 0.0f;
        bg.fill_color[2] = 0.0f;
        graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bg);

        // Δημιουργία του πινέλου για το δέντρο
        graphics::Brush treeBrush;
        treeBrush.fill_color[0] = 0.0f;
        treeBrush.fill_color[1] = 0.5f;
        treeBrush.fill_color[2] = 0.0f;

        // Κλήση της συνάρτησης για σχεδίαση του fractal δέντρου
        drawBinaryTree(WINDOW_WIDTH / 2, WINDOW_HEIGHT - 100, 100, M_PI / 2, 20,
treeBrush);
    }

  }
#endif // FRACTALS_H
```

Δείγμα main που καλεί fractals.

```
#include "sgg/graphics.h"
#include <cmath>
#include "fractals.h"

// Ορισμός παραμέτρων παραθύρου
const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 700;

/**
 * @brief Κύρια συνάρτηση που εκκινεί το πρόγραμμα.
 *
 * Δημιουργεί το παράθυρο και καθορίζει τη συνάρτηση σχεδίασης για να απεικονιστεί το
fractal.
 *
 * @return int Επιστρέφει 0 για επιτυχημένη εκτέλεση.
 */
int main() {
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Sierpinski Triangle
Fractal");
    //graphics::createWindow(VicsekFractal::windowWidth, VicsekFractal::windowHeight,
"");
    // Ορισμός της συνάρτησης σχεδίασης
    //graphics::setDrawFunction(SierpinskiTriangle::draw);
    //graphics::setDrawFunction(SierpinskiCarpet::draw);
    //graphics::setDrawFunction(BarnsleyFern::draw);
    //graphics::setDrawFunction(DragonCurve::draw);
    //graphics::setDrawFunction(CantorSet::draw);
    //graphics::setDrawFunction(ApollonianGasket::draw);
    //graphics::setDrawFunction(PeanoCurve::draw);
    //graphics::setDrawFunction(HilbertCurve::draw);
    //graphics::setDrawFunction(TSquare::draw);
    //graphics::setDrawFunction(HFractal::draw);
    //graphics::setDrawFunction(VicsekFractal::draw);
    //graphics::setDrawFunction(MengerSponge::draw);
    //graphics::setDrawFunction(Hexaflake::draw);
    //graphics::setDrawFunction(GosperCurve::draw);
    //graphics::setDrawFunction(LevyCCurve::draw);
    //graphics::setDrawFunction(Pentaflake::draw);
    //graphics::setDrawFunction(GoldenDragon::draw);
    //graphics::setDrawFunction(ButterflyFractal::draw);
    //graphics::setDrawFunction(PlasmaFractal::draw);
    //graphics::setDrawFunction(OctahedronFractal::draw);
    //graphics::setDrawFunction(SnowflakeCurve::draw);
    //graphics::setDrawFunction(LSystemFractal::draw);
    //graphics::setDrawFunction(VortexFractal::draw);
    //graphics::setDrawFunction(DurerPentagonFractal::draw);
    //graphics::setDrawFunction(BarnsleyFernFractal::draw);
    //graphics::setDrawFunction(SierpinskiHexagonFractal::draw);
    //graphics::setDrawFunction(CrystalGrowthFractal::draw);
    //graphics::setDrawFunction(FractalHands::draw);
    //graphics::setDrawFunction(FractalMountains::draw);
    //graphics::setDrawFunction(PentagonFractal::draw);
    //graphics::setDrawFunction(ThueMorseCurve::draw);
    //graphics::setDrawFunction(JellyfishFractal::draw);
    //graphics::setDrawFunction(GosperIslandFractal::draw);
    //graphics::setDrawFunction(PentadendriteFractal::draw);
    //graphics::setDrawFunction(PinwheelTilingFractal::draw);
    //graphics::setDrawFunction(VicsekCrossFractal::draw);
    //graphics::setDrawFunction(HexagonTilingFractal::draw);
    //graphics::setDrawFunction(SquareFractal::draw);
    //graphics::setDrawFunction(CarpetFractal::draw);
    //graphics::setDrawFunction(CirclePackingFractal::draw);
```

```
    //graphics::setDrawFunction(HTreeFractal::draw);
    //graphics::setDrawFunction(GasketOfTrianglesFractal::draw);
    //graphics::setDrawFunction(ZenoParadoxFractal::draw);
    //graphics::setDrawFunction(CarpetTilingFractal::draw);
    //graphics::setDrawFunction(IceFractal::draw);
    //graphics::setDrawFunction(ArchimedesSpiralFractal::draw);
    //graphics::setDrawFunction(CubicFractal::draw);
    //graphics::setDrawFunction(TorusKnotFractal::draw);
    //graphics::setDrawFunction(KnotTheoryFractal::draw);
    //graphics::setDrawFunction(ButterflyEffectFractal::draw);

    //LorenzAttractorFractal::calculateLorenzAttractor();
    //graphics::setDrawFunction(LorenzAttractorFractal::draw);

    //graphics::setDrawFunction(RingFractal::draw);
    //graphics::setDrawFunction(PythagoreanTreeFractal::draw);
    //graphics::setDrawFunction(FlowerFractal::draw);
    //graphics::setDrawFunction(CrystalFractal::draw);
    //graphics::setDrawFunction(FibonacciSpiralFractal::draw);
    //graphics::setDrawFunction(TriangularSpiralFractal::draw);
    //graphics::setDrawFunction(BinaryTreeFractal::draw);


    // Έναρξη του κύκλου μηνυμάτων
    graphics::startMessageLoop();
    return 0;
}
```