

GRAFIKA KOMPUTEROWA

SPRAWOZDANIE

Damian Koper, 241292

24 października 2019

Spis treści

1	OpenGL - podstawy	3
1.1	Bazowa aplikacja	3
1.2	Dywan Sierpińskiego	4

Spis rysunków

1	Dywan Sierpińskiego po 6 iteracjach.	4
2	Częściowo narysowany dywan Sierpińskiego po zakończeniu 30 gałęzi rekurencji. . .	5
3	Dywan Sierpińskiego narysowany w całości.	5
4	Dywan Sierpińskiego po 3 iteracjach z losowym kolorem zielonym oraz perturbacjami. .	7

Spis listingów

1	Bazowy program wyświetlający czarne okno.	3
2	Funkcja rysująca dywan Sierpińskiego rekurencyjnie. Pominęto niektóre wywołania. .	5
3	Struktura danych przechowująca aktualny stan rysowania.	6
4	Funkcja rysująca dywan Sierpińskiego iteracyjnie. Pominęto niektóre wywołania. .	6

1 OpenGL - podstawy

OpenGL (*Open Graphics Library*) stanowi otwarty i uniwersalny interfejs umożliwiający renderowanie grafiki 2D i 3D. Obliczenia realizowane są dzięki bezpośredniej interakcji z GPU, który, mając zaimplementowane potrzebne operacje, może szybko przeprowadzać obliczenia. Natura abstrakcyjnego interfejsu jakim jest OpenGL pozwala tworzyć przenośne programy renderujące grafikę bez zważania na platformę uruchomienia, gdzie obliczenia mogą być realizowane programowo jak i sprzętowo.

1.1 Bazowa aplikacja

Biblioteką, która tworzy środowisko uruchomieniowe dla wyświetlania wyrenderowanej grafiki i realizuje operacje wejścia-wyjścia jest GLUT (*OpenGL Utility Toolkit*). Prosty program wyświetlający okno można stworzyć niewielkim nakładem pracy.

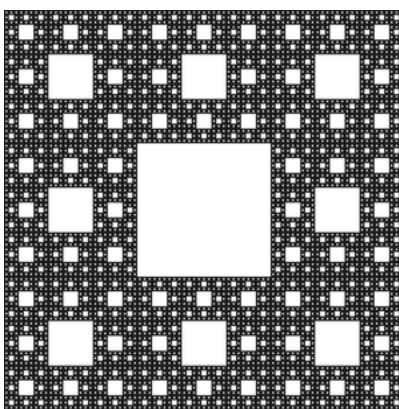
```
1 #include <GL/glut.h>
2 void draw()
3 {
4     glClearColor(0, 0, 0, 1);
5     glClear(GL_COLOR_BUFFER_BIT);
6     glFlush();
7 }
8
9 int main(int argc, char **argv)
10 {
11     glutInit(&argc, argv);
12     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
13     glutInitWindowPosition(50, 50);
14     glutInitWindowSize(800, 800);
15     glutCreateWindow("Lab GK");
16     glutDisplayFunc(draw);
17     glutMainLoop();
18     return 0;
19 }
```

Listing 1: Bazowy program wyświetlający czarne okno.

W funkcji `main` w pierwszej kolejności inicjalizowana jest sama aplikacja, a następnie wszystkie jej parametry. Następnie do biblioteki GLUT przekazywana poprzez wskaźnik jest funkcja `draw`, która jest odpowiedzialna za bezpośrednią interakcję z API OpenGL, a zatem za rysowanie właściwych elementów w wyświetlonym oknie. W tej wersji funkcja `draw` czyści okno kolorem czarnym i opróżnia bufor przekazując dane na ekran.

1.2 Dywan Sierpińskiego

Dywan Sierpińskiego jest fraktalem otrzymanym z kwadratu podzielonego na 9 mniejszych kwadratów, z których usuwany jest środkowy. Procedura ta jest rekurencyjnie powtarzana dla pozostałych ośmiu kwadratów.



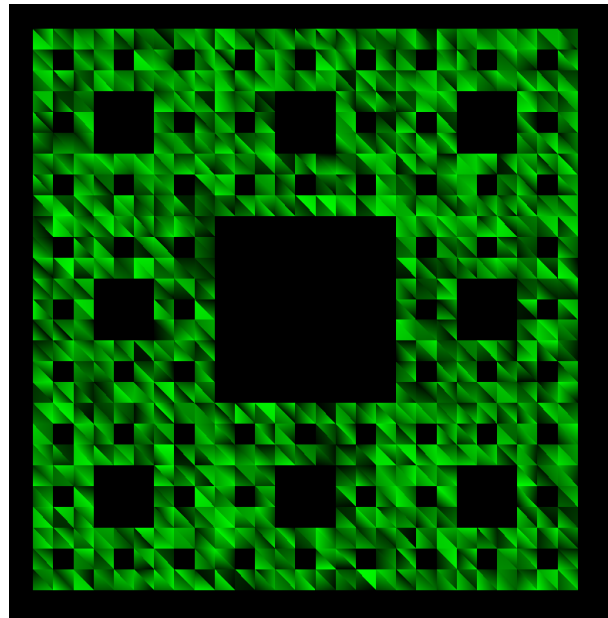
Rysunek 1: Dywan Sierpińskiego po 6 iteracjach.

1.2.1 Rysowanie rekurencyjne

Rysowanie dywanu Sierpińskiego zrealizowane za pomocą rekurencji zakłada rekurencyjne wywołanie funkcji dla każdego z ośmiu zewnętrznych kwadratów fraktalu. Poziomów wywoływań następuje tyle ile zostało zdefiniowane dla pierwszego wywołania funkcji. Gdy zostanie spełniony warunek zakończenia rekurencji, rysowany jest kwadrat. Funkcja po raz pierwszy zostaje wywołana w metodzie `draw` zaraz po wyczyszczeniu okna.



Rysunek 2: Częściowo narysowany dywan Sierpińskiego po zakończeniu 30 gałęzi rekurencji.



Rysunek 3: Dywan Sierpińskiego narysowany w całości.

```

1 void carpet(int levels, float a, float dx = 0, float dy = 0)
2 {
3     if (levels != 0)
4     {
5         a = a / 3;
6         carpet(levels - 1, a, dx - a, dy - a);
7         ...
8         carpet(levels - 1, a, dx + a, dy + a);
9     }
10    else
11    {
12        //Pomocnicza funkcja rysujaca kwadrat
13        rect(dx, dy, a);
14    }
15 }

```

Listing 2: Funkcja rysująca dywan Sierpińskiego rekurencyjnie. Pominęto niektóre wywołania.

1.2.2 Rysowanie iteracyjne

Rysowanie iteracyjne możliwe jest do osiągnięcia przez matematyczne sprawdzenie czy dany kwadrat ma być wypełniony czy nie, albo zamianę wersji rekurencyjnej do wersji iteracyjnej z wykorzystaniem kolejki i struktury danych opisującej aktualny poziom rysowania.

```
1 struct CarpetLevelData
2 {
3     int level;
4     float a;
5     float dx = 0;
6     float dy = 0;
7 };
```

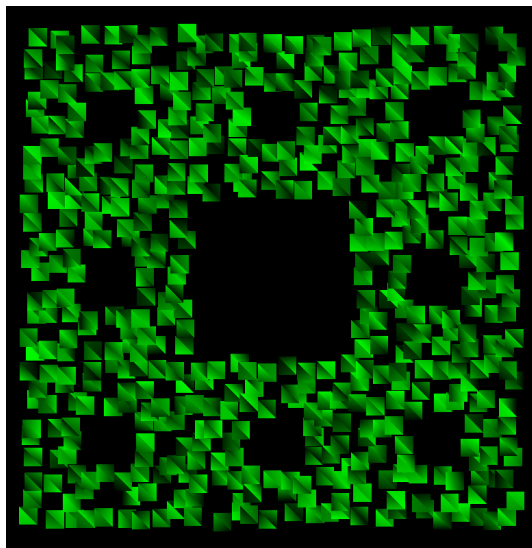
Listing 3: Struktura danych przechowująca aktualny stan rysowania.

```
1 void carpetIt(int levels, float a)
2 {
3     queue<CarpetLevelData> q = queue<CarpetLevelData>();
4     q.push({levels, a});
5     while (!q.empty())
6     {
7         CarpetLevelData data = q.front();
8         if (data.level > 0)
9         {
10             data.a = data.a / 3;
11             q.push({data.level - 1, data.a, data.dx - data.a, data.dy - data.a});
12             ...
13             q.push({data.level - 1, data.a, data.dx + data.a, data.dy + data.a});
14         }
15         else
16         {
17             rect(data.dx, data.dy, data.a);
18         }
19         q.pop();
20     }
```

Listing 4: Funkcja rysująca dywan Sierpińskiego iteracyjnie. Pominięto niektóre wywołania.

1.2.3 Losowe kolory i perturbacje

Losowe kolory osiągnięto poprzez wywołanie funkcji `glColor3ub(0, randChar(), 0)` przed wywołaniem funkcji interfejsu OpenGL definiującą składowy wierzchołek trójkąta tworzącego kwadrat, gdzie funkcja `randChar` zwraca liczbę z przedziału $< 0; 255 >$. Mając kontrolę nad rysowaniem każdego kwadratu można wprowadzić również losowe jego przesunięcie w osi X i Y.



Rysunek 4: Dywan Sierpińskiego po 3 iteracjach z losowym kolorem zielonym oraz perturbacjami.