POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Wprowadzenie do grafiki komputerowej

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 2

TEMAT ĆWICZENIA Open GL - podstawy

Wykonał:	**Marcel Guzik**
Termin:	**PN/P 7:30-10:00**
Data wykonania ćwiczenia:	**11-10-2021**
Data oddania sprawozdania:	**18-10-2021**
Ocena:	

Uwagi prowadzącego:

1 Wprowadzenie

Celem ćwiczenia jest narysowanie dywanu Sierpińskiego za pomocą biblioteki OpenGL.

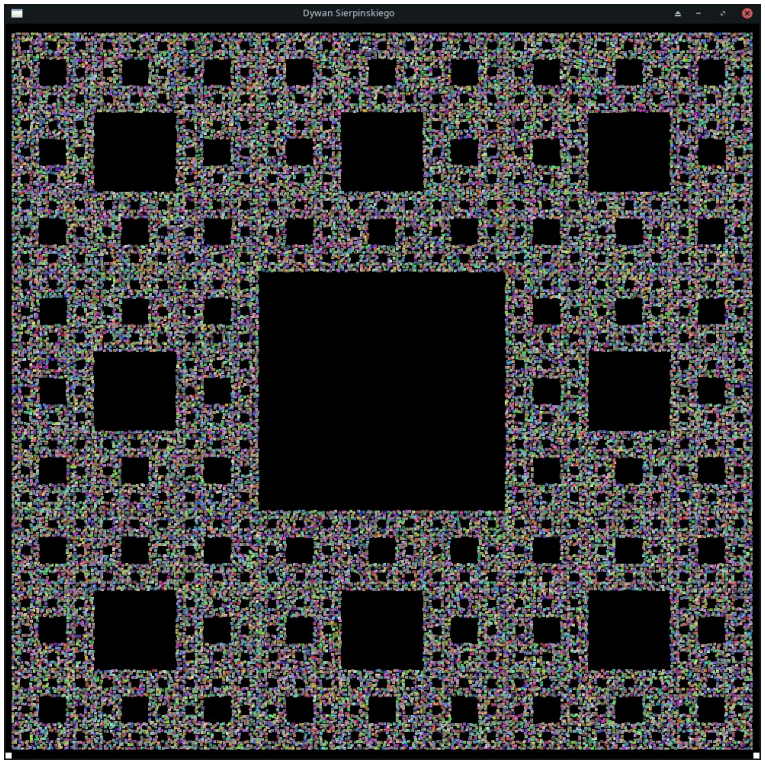


Figure 1: Dywan sierpińskiego – zrzut ekranu okna programu

Dywan Sierpińskiego to fraktal otrzymany z kwadratu za pomocą podzielenia go na dziewięć (3x3) mniejszych kwadratów, usunięcia środkowego kwadratu i ponownego rekurencyjnego zastosowania tej samej procedury do każdego z pozostałych ośmiu kwadratów. Nazwa pochodzi od nazwiska Wacława Sierpińskiego.

2 Wstęp do OpenGL

2.1 Szkielet aplikacji OpenGL

Do zrealizowania zadania używana jest biblioteka graficzna OpenGL (Open Graphics Library), a także biblioteka FreeGLUT (GL Utility Toolkit) abstrahująca API systemu operacyjnego i udostępniająca funkcje do tworzenia okienek i przechwytywania zdarzeń.

Rysowanie odbywa się poprzez wywołanie funkcji `glBegin`, następnie wywoływanie funkcji `glVertex2*` definiującej położenie kolejnych punktów, a następnie zakończenie procedury rysowania poprzez wywołanie funkcji `glEnd`. Ten sposób rysowania oraz związany z nimi zbiór funkcji nosi nazwę Fixed Function Pipeline, ponieważ transformacje/przekształcenia używane w procesie rysowania (function) są statycznie zdefiniowane i nie mogą zostać zmienione (fixed), zatem ten sposób rysowania nie zezwala na elastyczność jaką można uzyskać stosując np. shadery. Ten sposób rysowania oraz związany z nim zbiór funkcji został w 2008 zdeprecjonowany i jego używanie nie jest

aktualnie rekomendowane. Zamiast tego rekomendowane jest m.in. stosowanie shaderów oraz zarządzanie stosem macierzy ręcznie[1].

2.2 System współrzędnych

Pozycje wierzchołków przekazywane do funkcji `glVertex2f` nie oznaczają pozycji pikseli w których te wierzchołki mają się znajdować, lecz pozycję w ciągłej przestrzeni (lub powierzchni w wypadku 2D), którą możemy transformować korzystając z przeznaczonych do tego celu macierzy oraz funkcji do manipulacji nimi, np. `glOrtho`, `glScale`, `glTranslate`, etc.

W domyślnym stanie tych macierzy, tzn. bez modyfikacji ich żadnymi z powyższych funkcji, OpenGL używa systemu współrzędnych zwanego Normalized Device Coordinates (znormalizowane współrzędne urządzenia):

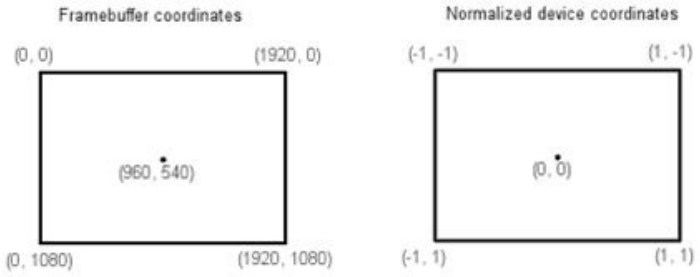


Figure 2: Pozycja punktu leżącego na środku okienka o rozmiarach 1920x1080; po lewej - klasycznie w pikselach, po prawej - w znormalizowanych współrzędnych urządzenia

Dzięki takiemu podejściu nie musimy znać rozmiaru okienka w pikselach, a nasza grafika będzie wypełniać zarówno małe, jak i duże okienka. Co prawda gdy okienko będzie o wiele dłuższe w jednej z osi, jego zawartość będzie znacząco zniekształcona, jednak OpenGL udostępnia wiele mechanizmów do kontrolowania naszej powierzchni rysowania.

Pierwszym z nich jest funkcja `glViewport`, która pozwala określić rozmiar ekranu (viewportu) używanego przez OpenGL do rysowania niezależnie od faktycznego rozmiaru okienka ustalonego przez użytkownika. Jako argumenty przyjmuje ona przesunięcie od punktu (0, 0) (lewego górnego rogu okienka) oraz rozmiar obszaru rysowania w pikselach.

1 https://www.khronos.org/opengl/wiki/Legacy_OpenGL

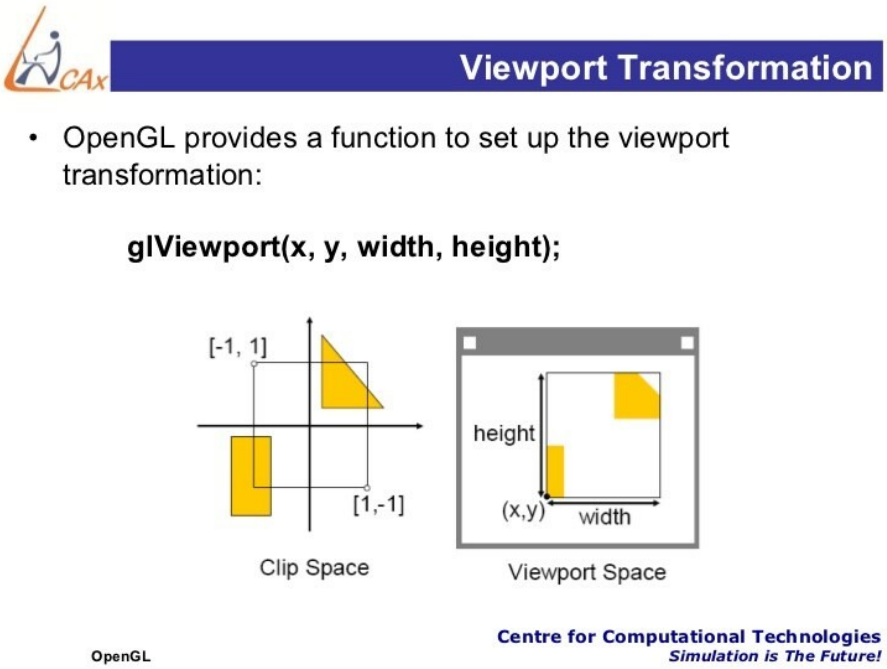


Figure 3: Zaprezentowanie działania funkcji `glViewport`

Następnie mamy do dyspozycji przekształcenia macierzowe, które określają jak transformowany jest system współrzędnych w naszym viewporcie. Oprócz funkcji `glScale`, `glTranslate`, `glRotate`, realizujących podstawowe przekształcenia przestrzeni (lub, w naszym wypadku, powierzchni), mamy także funkcje `glOrtho` oraz `glFrustrum`, które opisują odpowiednio macierze rzutu prostokątnego oraz perspektywistycznego, używane w procesie transformowania współrzędnych trójwymiarowych na współrzędne dwuwymiarowe, potrzebne do rysowania na dwuwymiarowej powierzchni viewportu.

W przykładowym kodzie źródłowym znajdującym się na stronie internetowej tego ćwiczenia, funkcja `glOrtho` została użyta do skalowania jednej z osi powierzchni, tak aby pomimo niekwadratowego okienka, rysowany dywan zachowywał wymiary kwadratu. W niniejszej realizacji zadania, to zachowanie przeniesiono do funkcji glViewport, aby zagwarantować kwadratowe wymiary viewportu i przez to zminimalizować zniekształcenia powstałe z transformacji niekwadratowej przestrzeni ciągłej na kwadratową przestrzeń dyskretną. Więcej w sekcji 4.1. Wywołanie funkcji glOrtho nie zostało jednak usunięte i pozostało ono aby zmienić system współrzędnych z ((-1, -1), (1,1)) na ((-100, -100), (100, 100)) oraz dla zaprezentowania zrozumienia jej działania przez osobę wykonującą ćwiczenie.

3 Realizacja programu

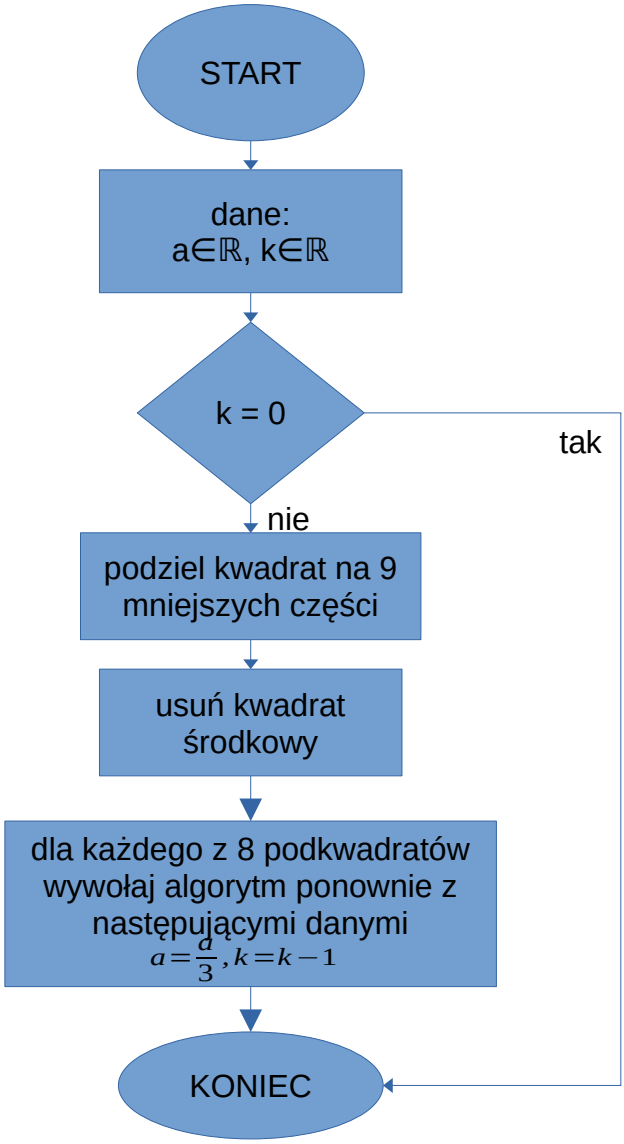
3.1 Algorytm rysowania dywanu

Algorytm generacji dywanu jest następujący:

1. Obiektem wyjściowym jest kwadrat o boku a
2. Kwadrat dzielimy na 9 mniejszych równych kwadratów o boku a/3 i usuwamy środkowy

3. Każdy z mniejszych kwadratów znów dzielimy na 9 części i z każdej części usuwamy część środkową
4. Powtarzmy dalsze kroki według wyżej opisanej zasady

Algorytm na wejście przyjmuje rozmiar kwadratu zdefiniowany przez zmienne `CARPET_START` i `CARPET_END` oraz liczbę poziomów rekurencji `LEVELS`.



Drawing 1: Diagram blokowy algorytmu rysowania dywanu Sierpińskiego

3.2 Perturbacje i inne efekty

Aby uzyskać efekt wizualny perturbacji, każdy rysowany kwadrat przesunięto o losowo wygenerowany offset. Maksymalną wartość przesunięcia w obu osiach określa stała `DISTORTION_MAX`.

Kolor kwadratu uzyskuje się w następujący sposób: każdemu wierzchołkowi kwadratu nadaje się losowo wygenerowany kolor, a następnie kolor w środku kwadratu jest mieszany z proporcjami określonymi odległościami punktu wewnątrz kwadratu od każdego z jego wierzchołków. Ten proces uzyskiwania koloru dla fragmentów leżących pomiędzy wierzchołkami nazywamy interpolacją.

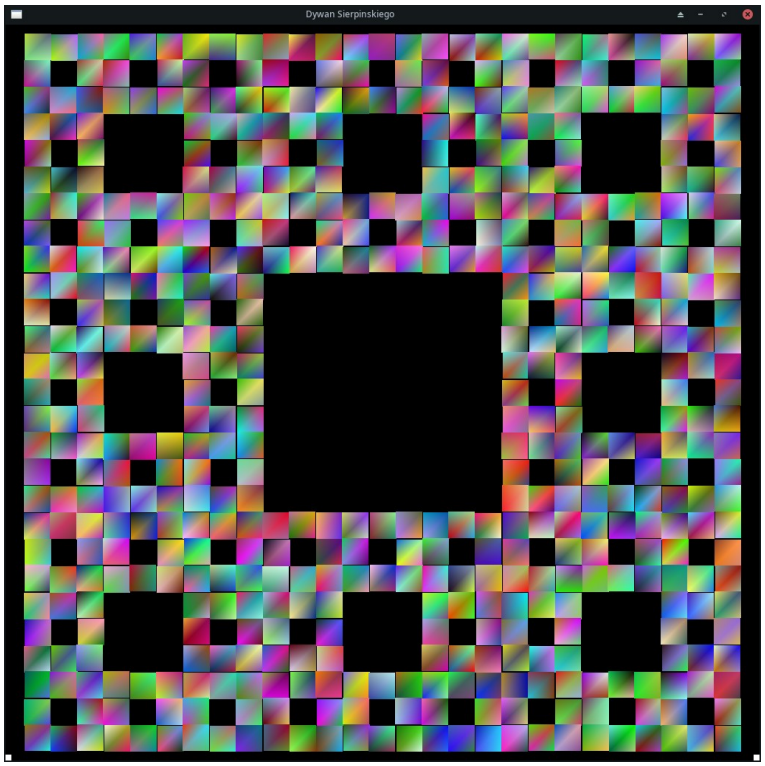
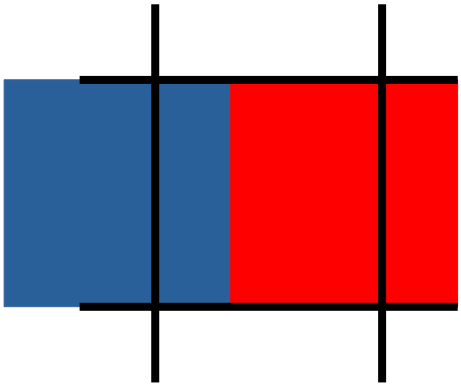


Figure 4: Dywan narysowany z mniejszym poziomem rekurencji, dla zaprezentowania koloru kwadratów

4 Błędy i inne trudności

4.1 Granice kwadratów nie leżące idealnie na granicach pikseli

Z powodu wykładniczo zmniejszającego się rozmiaru kwadratów, już od relatywnie niskiej ilości poziomów, następujemy na zasadniczy problem: rozmiar pojedynczych kwadratów staje się mniejszy niż rozmiar pojedynczego piksela. Tzn. zakładając niekwadratowy viewport, oraz macierz projekcji mająca przeskalować kwadrat narysowany na tym viewporcie tak, by został on narysowany jako kwadrat, mamy do czynienia ze zniekształceniami jeżeli elementy które rysujemy są mniejsze niż jeden piksel.



Drawing 2: Przykład dwóch różnych kwadratów zajmujących przestrzeń jednego piksela. Jakiego koloru powinien być piksel?

W takiej sytuacji mamy do dyspozycji następujące rozwiązania:

- zwiększyć rozdzielczość ekranu (nierealne rozwiązanie)
- jeżeli dany kwadrat jest mniejszy niż piksel, przejść do alternatywnego trybu rysowania w którym nie rysujemy już kwadratów pod sobą, lecz dla kolejnych poziomów obliczamy pole które odejmujemy od kwadratu wielkości piksela. Wraz z odejmowaniem pola, wygaszamy dany piksel, tzn. im mniejsze pole nam pozostaje, tym dalej koloru kwadratu a bliżej koloru tła, będzie dany piksel
- zastosować generalny algorytm typu anti-aliasing
- zablokować viewport tak, by zawsze był kwadratem, a jego rozmiar był wielokrotnością ilości kwadratów leżących na boku dywanu; jest on prosty do zrealizowania, jednak rozwiązując problem ten sposób od pewnej ilości poziomów nie będą widoczne różnice, efektywnie nie będą rysowane żadne kolejne detale

Do rozwiązania problemu została wybrana opcja nr 4. Niestety, dla wyższych wartości poziomu rekurencji, nie jest możliwe rysowanie kolejnych poziomów kiedy zakładamy że każdy kwadrat będzie co najmniej wielkości piksela. W takim wypadku aby pozbyć się większych zniekształceń, a jednocześnie móc rysować małe kwadraty, rozmiar boku kwadratowego viewportu określa się za pomocą wzoru 3^n , gdzie *n* oznacza ilość poziomów rekurencji do której należy zagwarantować wielokrotność liczby pikseli względem liczby kwadratów.

5 Podsumowanie

Program rysuje dywan Sierpińskiego z wystarczającą precyzją. Dodatkowo jakiekolwiek zniekształcenia są pomijalnie małe w porównaniu do ustawionego poziomu dyspersji (który jest podobny do dyspersji przykładowego programu załączonego na stronie). Sposób rysowania jest kontrolowany przez zdefiniowane na początku programu stałe. Niestety bez użycia bardziej zaawansowanych narzędzi udostępnianych przez OpenGL nie jest możliwe generalne rysowanie elementów mniejsze niż piksel, zatem każda próba reprezentacji struktur fraktalnych będzie skutkowała obrazem w najlepszym razie niekompletnym, a w najgorszym razie wyraźnie zniekształconym.