Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Grafika komputerowa i komunikacja człowiek-komputer

Laboratorium 2 – Podstawy OpenGL

---

semestr zimowy 2023/2024

Termin zajęć:
środa TN 14:15-17:15

Autor:
Eryk Mika 264451

Prowadzący:
Dr inż. arch. Tomasz Zamojski

1. Przygotowanie środowiska pracy

Na moim komputerze z systemem *Windows 10* był już zainstalowany interpreter języka *Python*, w którym pisane są skrypty na laboratoria z kursu. Musiałem zainstalować pakiety *PyOpenGL* oraz *glfw*, co zrobiłem za pomocą instalatora pakietów *pip*, wydając następujące polecenie w terminalu:

```
pip install PyOpenGL glfw
```

Spowodowało to zainstalowanie tych pakietów, dzięki czemu możliwe jest uruchamianie skryptów opisanych w tym sprawozdaniu. Skrypty uruchamiam z poziomu linii poleceń za pomocą polecenia *python <nazwa skryptu z .py>*.

2. Kod wspólny dla rozwiązań do wszystkich zadań

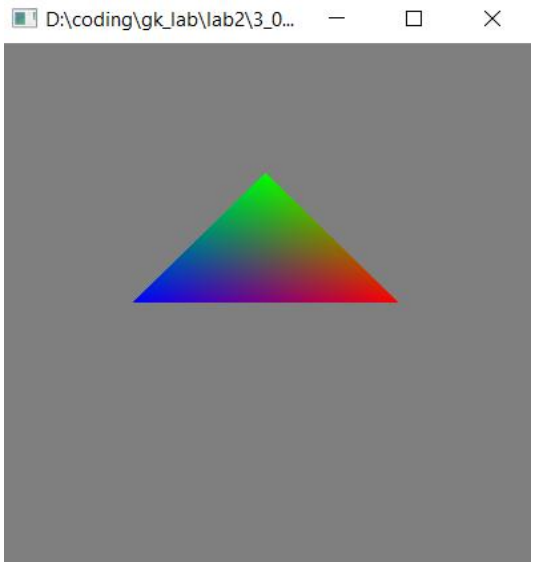
Do napisania rozwiązań problemów ze wszystkich zadań została wykorzystana część przykładowego kodu programu, który został przekazany przez Prowadzącego. Kod ten odpowiedzialny jest za utworzenie okna i renderowanie w nim grafiki 2D. Obejmuje on zaimportowanie odpowiednich modułów z pakietów OpenGL i glfw oraz funkcje:

- *startup()* - ustawia początkowe ustawienia dla programu. W tym przypadku, ustawia rozmiar widoku oraz kolor, do jakiego będzie czyszczony bufor,
- *shutdown()*,
- *render(time)* - zawiera specyficzny dla zadania kod do renderowania obrazu,
- *update_viewport(window, width, height)* - aktualizuje widok na scenie. Przyjmuje trzy argumenty: window (okno GLFW), width (szerokość) i height (wysokość). Ustawia odpowiednie parametry dla przeskalowania widoku,
- *main()* - zawiera logikę głównego programu.

3. Zadanie na ocenę 3.0 – skrypt *3_0.py*

W zadaniu tym należało narysować trójkąt z każdym wierzchołkiem innego koloru. Zrobiłem to poprzez modyfikację dostarczonego kodu. W funkcji *render()* zostawiłem tylko jedno wywołanie funkcji rysującej prymityw - trójkąt - *glBegin(GL_TRIANGLES)*. Następnie przed umieszczeniem każdego wierzchołka trójkąta w pamięci za pomocą funkcji *glVertex()* zmieniam kolor kreślonego prymitywu za pomocą funkcji *glColor3f()*, która pozwala na określenie barwy w modelu RGB i zakresie 0-1 dla każdej ze składowych modelu. Wywołanie funkcji *glEnd()* powoduje zakończenie

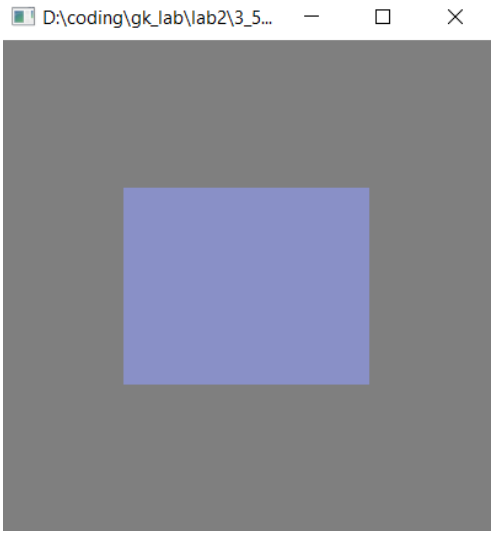
zapisywania prymitywu do pamięci, a wywołanie funkcji *glFlush()* powoduje przesłanie zawartości pamięci do wyświetlenia. Efekt działania skryptu został przedstawiony na zrzucie ekranu:



Rysunek 3.1 Efekt uruchomienia skryptu 3_0.py

4. Zadanie na ocenę 3.5 – skrypt *3_5.py*

W zadaniu tym należało napisać funkcję rysującą prostokąt w zadanym miejscu. Napisana przeze mnie funkcja *draw_rectangle(x, y, a, b)* przyjmuje 4 argumenty, które odpowiednio oznaczają współrzędne wierzchołka prostokąta (x, y) oraz długości jego boków. Prostokąt jest stworzony jako para trójkątów prostokątnych, których częścią wspólną są ich przeciwprostokątne. Ich współrzędne to odpowiednio $(\mathrm{x},\ \mathrm{y}), (\mathrm{x}+\mathrm{a},\ \mathrm{y}), (\mathrm{x},\ \mathrm{y}+\mathrm{b})$ dla pierwszego trójkąta oraz $(x+a,\ y+b), (x+a,\ y), (x, y+b)$ dla drugiego trójkąta. Funkcja *draw_rectangle()* jest wywoływana w funkcji *render()*. Efekt działania skryptu:



Rysunek 4.1 Efekt uruchomienia skryptu 3_5.py

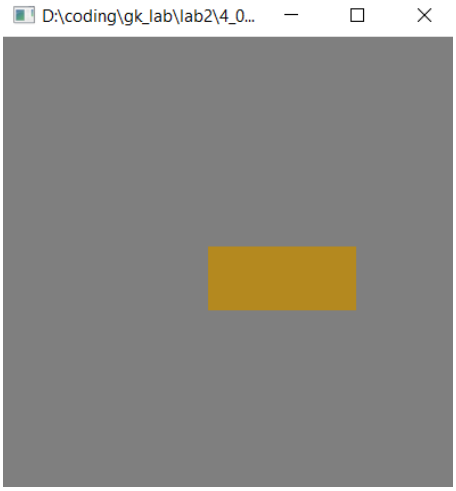
5. Zadanie na ocenę 4.0 – skrypt *4_0.py*

Zadanie to polegało na modyfikacji funkcji z poprzedniego zadania w taki sposób, aby wprowadzić losowe deformacje prostokąta i zmiany jego koloru. Do listy argumentów funkcji został dodany argument *d*, który determinuje tę losowość. Domyślnie przyjmuje on wartość 0. Do obsługi został wykorzystany moduł *random* zaimportowany poprzez *import random*. Argument d został użyty jako ziarno generatora liczb losowych. Uzyskane wartości są następnie użyte do modyfikacji parametrów prostokąta. W przypadku losowania koloru dzielenie wartości całkowitej z przedziału od 0 do 255 (włącznie) przez 255 gwarantuje przekazanie do funkcji wartości składowej modelu RGB w zakresie 0-1.

```
28      glColor3f(random.randint(0,255)/255, random.randint(0,255)/255, random.randint(0,255)/255)
29
30      a, b = random.randint(0, 100), random.randint(0, 100)
```

Rysunek 5.1 Najpierw losujemy kolor prostokąta, a następnie długości boków prostokąta.

W celu uruchomienia skryptu za pomocą terminala/wiersza poleceń (cmd) jako dodatkowy argument należy podać ziarno do generatora liczb losowych, na przykład ***„python 4_0.py 5”***. Efekt działania:



Rysunek 5.2 Uruchomienie skryptu 4_0.py z parametrem 5 w linii komend

6. Zadanie na ocenę 4.5 – skrypt *4_5.py*

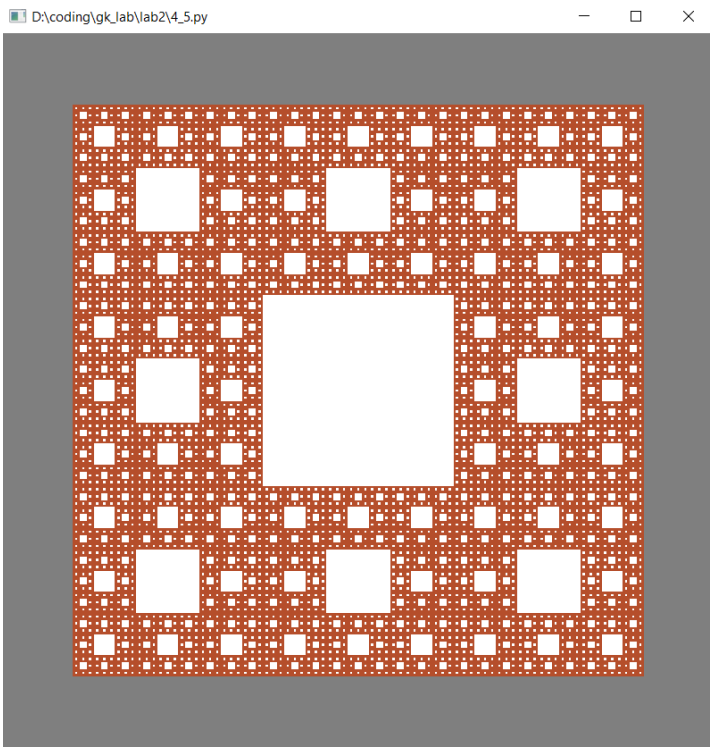
W zadaniu tym należało narysować fraktal – dywan Sierpińskiego. W celu rozwiązania tego problemu użyłem podejścia rekurencyjnego. Najpierw rysowany jest kolorowy prostokąt (kwadrat) przy użyciu funkcji z wcześniejszego zadania, a następnie za pomocą rekurencyjnych wywołań funkcji *draw_recursively*() tworzone są w tym kwadracie białe „wycięcia”. Funkcja jako argumenty przyjmuje, podobnie jak funkcja *draw_rectangle()*, wartości x, y, a, b. Dwie pierwsze z nich określają współrzędne prostokąta (kwadratu), a dwa kolejne długości jego boków. W środku tego kwadratu rysujemy biały kwadrat o bokach długości $^1/_3$ boków większego kwadratu.

Następnie, dzieląc wnętrze większego kwadratu wokół mniejszego kwadratu na 8 takich samych mniejszych kwadratów, wywołujemy funkcję rekurencyjnie, przekazując współrzędne wierzchołków tych kwadratów i długości ich boków do funkcji. Współrzędne te znajdują się w punktach o współrzędnych równych x lub y, bądź będących w $^1/_3$ lub $^2/_3$ części boku większego kwadratu.

Funkcja przyjmuje także argument *sim* o domyślnej wartości 1, który determinuje głębokość rekurencji a tym samym stopień podobieństwa. Parametr ten, podobnie jak we wcześniejszym zadaniu, jest podawany przy uruchamianiu skryptu z wiersza poleceń.

```
56      new_width = 0.3333 * a
57      new_height = 0.3333 * b
58
59      draw_rectangle(x + new_width, y + new_height, new_width, new_height)
60
61      # Wspolrzedne x to po kolei wspolrzedna x poczatku, 1/3 oraz 2/3 czesci boku wiekszego kwadratu
62
63      # row 1 / wiersz 1 - od gory - wierzcholki kwadratow o wspolrzednych y w 2/3 czesci boku pionowego (y + 2*0.3333*b)
64      draw_recursively(x,                 y + 2 * new_height, new_width, new_height, sim-1)
65      draw_recursively(x + new_width,     y + 2 * new_height, new_width, new_height, sim-1)
66      draw_recursively(x + 2 * new_width, y + 2 * new_height, new_width, new_height, sim-1)
67
68      # row 2 / wiersz 2 - wierzcholki kwadratow o wspolrzednych y w 1/3 czesci boku pionowego (y + 0.3333*b)
69      draw_recursively(x,                 y + new_height, new_width, new_height, sim-1)
70      # w srodku prostokat/kwadrat z poprzedniej iteracji
71      draw_recursively(x + 2* new_width, y + new_height, new_width, new_height, sim-1)
72
73      # row 3 / wiersz 3 - wierzcholki kwadratow o wspolrzednych y takich samych jak wiekszy kwadrat (y)
74      draw_recursively(x,                 y, new_width, new_height, sim-1)
75      draw_recursively(x + new_width,     y, new_width, new_height, sim-1)
76      draw_recursively(x + 2* new_width, y, new_width, new_height, sim-1)
```

Rysunek 6.1 Główna część rekurencyjnego algorytmu rysowania białych „wycięć”



Rysunek 6.2 Uruchomienie skryptu 4_5.py z parametrem 5