<table>
<tr><td>POLITECHNIKA WROCŁAWSKA

Filia w Jeleniej Górze</td><td><u>Autor</u>
Maciej Myśków
Indeks: 272794</td><td>Wydział informatyki i telekomunikacji
Rok: 2024/2025
Rok akadem.: 3</td></tr>
<tr><td colspan="3">Grafika komputerowa i komunikacja człowiek-komputer</td></tr>
<tr><td>Data ćwiczenia:
09.10.2024</td><td rowspan="2"><u>Temat ćwiczenia laboratoryjnego:</u>
Podstawy OpenGL, grafika 2D</td><td>Ocena:</td></tr>
<tr><td>Nr ćwiczenia:
1</td><td>Prowadzący:
Dr. Inż. arch. Tomasz Zamojski</td></tr>
</table>

Wstęp

Podczas zajęć wykonywano rysowanie oraz przekształcenia figur w różny sposób – w zależności od wykonywanego zadania, tak aby spełnić kryteria danego ćwiczenia. Rysowanie oraz przekształcenia figur oparte było na podstawie materiałów instruktażowych (film oraz pomoc prowadzącego) w tym również swój własny sposób interpretacji zadania.

Przed wykonywaniem poszczególnych zadań niezbędne było zapoznanie się z podstawowymi operacjami oraz typami obiektów – prymitywami biblioteki OpenGL. Niezbędne było również zainstalowanie niezbędnych komponentów (python + biblioteka OpenGL), które umożliwiły wykonywanie omawianych zadań.

Opis zadań realizowanych podczas zajęć

Zadanie 1

1. Cel ćwiczenia
2. Realizacja celu
3. Omówienie kodu oraz jego działania
4. Efekt wykonanej pracy

Zadanie 2

1. Cel ćwiczenia
2. Realizacja celu
3. Omówienie kodu oraz jego działania
4. Efekt wykonanej pracy

Zadanie 3

1. Cel ćwiczenia
2. Realizacja celu
3. Omówienie kodu oraz jego działania
4. Efekt wykonanej pracy

Zadanie 1

Cele ćwiczenia

Celem zadania pierwszego było odpowiednie skonfigurowanie środowiska programistycznego w oparciu o prezentacje oraz wytyczne prowadzącego. Następnym elementem (praktycznym) było narysowanie za pomocą prymitywów (w tym przypadku wierzchołków) trójkąta. Ostatnim elementem zadania było odpowiednie pokolorowanie jego wierzchołków – tak aby każdy wierzchołek był innego koloru.

Realizacja celu

Zadanie zostało zrealizowane w pełni, to znaczy zostało postawione i skonfigurowane środowisko jako krok pierwszy. W następnym kroku stworzono trójkąt za pomocą trzech prymitywów – wierzchołków. W ostatnim kroku pokolorowano wytworzony obiekt (trójkąt) w sposób taki, aby każdy z jego wierzchołków bym w innym kolorze zapisanym za pomocą modelu barw RGB.

Omówienie kodu oraz jego działania

Funkcja **startup()** jest wywoływana podczas uruchamiania programu i jej zadaniem jest zainicjowanie parametrów początkowych. Używa ona funkcji **update_viewport()**, aby ustawić przestrzeń rysowania, a także funkcji **glClearColor(0.5, 0.5, 0.5, 1.0)**, która definiuje kolor tła (szary) używany do czyszczenia ekranu. Funkcja **shutdown()** jest przewidziana na zamknięcie programu, ale w tym przypadku nie wykonuje żadnych czynności.

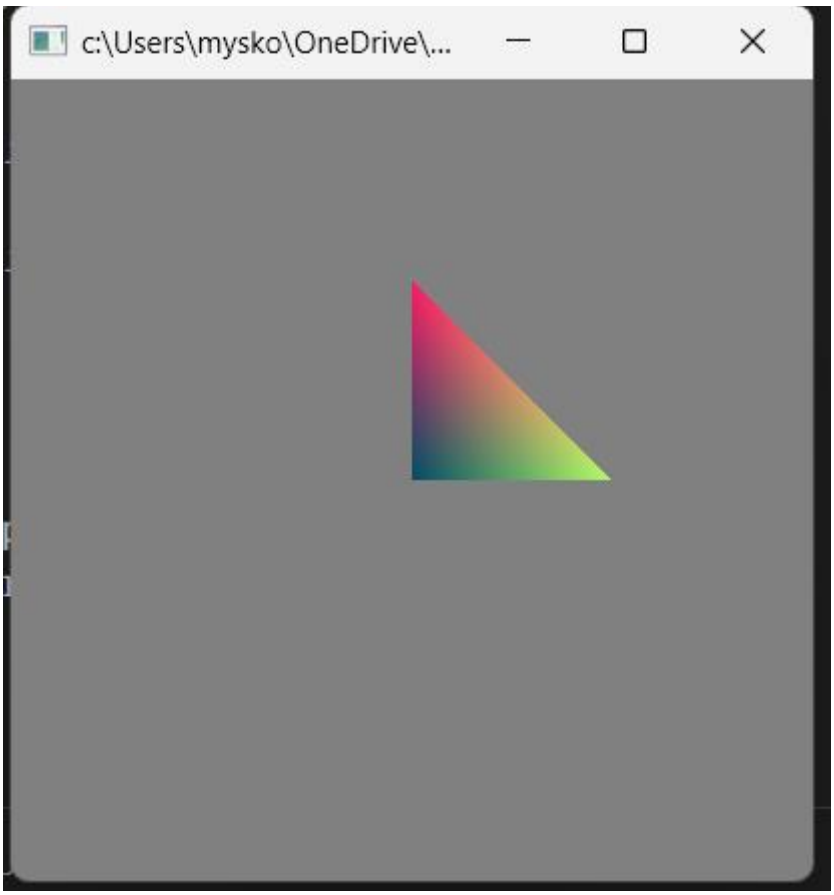
Rysowanie trójkąta odbywa się w funkcji **render(time)**. Za pomocą **glClear(GL_COLOR_BUFFER_BIT)** ekran jest czyszczony, a następnie w ramach bloku **glBegin(GL_TRIANGLES)** i **glEnd()** OpenGL rysuje trójkąt. Wierzchołki trójkąta są definiowane przez współrzędne 2D, podane w funkcjach **glVertex2f()**. Każdy wierzchołek ma przypisany kolor, który jest ustawiany za pomocą **glColor3f()**. Po zdefiniowaniu wierzchołków i kolorów, trójkąt jest rysowany na ekranie. Funkcja **glFlush()** wymusza wyświetlenie narysowanych elementów na ekranie.

Funkcja **update_viewport(window, width, height)** odpowiada za zarządzanie przestrzenią rysowania, szczególnie w sytuacjach, gdy zmienia się rozmiar okna. Gdy szerokość lub wysokość okna wynosi zero, ustawiana jest minimalna wartość 1, aby uniknąć dzielenia przez zero. **Funkcja glViewport()** definiuje obszar wyświetlania, a **glOrtho()** ustawia projekcję ortogonalną, co oznacza, że elementy rysowane w programie są widoczne w 2D. Ostatecznie funkcja ta przekształca przestrzeń współrzędnych w zależności od proporcji okna.

Funkcja **main()** uruchamia główną pętlę programu. Najpierw inicjalizowana jest biblioteka GLFW przy pomocy **glfwInit()**. Następnie tworzone jest okno o wymiarach 400x400 pikseli za pomocą **glfwCreateWindow()**. Funkcja **glfwMakeContextCurrent(window)** ustawia kontekst graficzny dla okna, co oznacza, że wszystkie operacje rysowania będą wykonywane właśnie w tym oknie. Funkcja **glfwSetFramebufferSizeCallback(window, update_viewport)** ustawia reakcję programu na zmianę rozmiaru okna.

W głównej pętli program działa, dopóki okno nie zostanie zamknięte. Wewnątrz tej pętli funkcja **render(glfwGetTime())** jest wywoływana, aby rysować trójkąt. Funkcja **glfwSwapBuffers(window)** zamienia bufor wyświetlania, aby obraz został zaktualizowany na ekranie, a **glfwPollEvents()** obsługuje zdarzenia użytkownika, takie jak kliknięcia myszą czy naciśnięcia klawiszy. Po zakończeniu działania programu wywoływana jest funkcja **shutdown()** oraz **glfwTerminate()**, aby zamknąć kontekst i okno.

Efekt wykonanej pracy



Rys. 1 – zadanie 1 efekt końcowy

Zadanie 2

Cele ćwiczenia

Celem zadania drugiego było narysowanie prostokąta o zadanym punkcie położenia początkowego – u mnie współrzędne środka prostokąta, wraz z odpowiednio obliczonymi punktami narożnymi (cztery wierzchołki). Prostokąt miał być zbudowany z dwóch trójkątów (prymitywów) połączonych ze sobą w odpowiedni sposób.

Realizacja celu

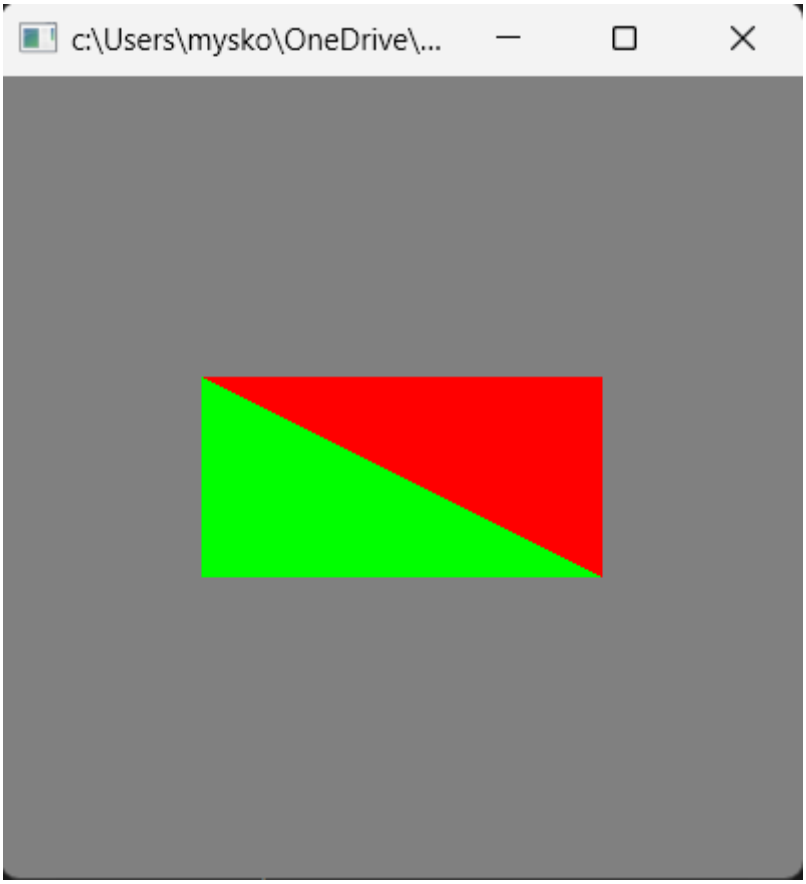
Zadanie zostało zrealizowane w większości podobnie jak zadanie powyżej (Zadanie 1), jednakże tutaj skonstruowano własną funkcję rysującą omawiany w zadaniu obiekt. Dodatkowo przed ostateczną implementacją wykonałem kilka prostych matematycznych obliczeń (na kartce), które umożliwiły mi wyznaczenie punktów wierzchołkowych (połówkowe) oraz środek w układzie 2D, które są zawarte w kodzie.

Omówienie kodu oraz jego działania

Funkcje **startup(), shutdown(), update_viewport()** oraz **main()** nie różnią się w zasadzie niczym od poprzedniego kodu z zadania 1, który został opisany powyżej. Jedyna drobna różnica dotyczy funkcji **render(time)**, która polega na tym, że wywołujemy w niej własną funkcję **rysuj_prostokat(x, y, a, b)** – która to zostanie opisana w następnym akapicie.

Funkcja **rysuj_prostokat()** przyjmuje współrzędne środka prostokąta oraz jego wymiary. Oblicza półwymiary (polowka_a i polowka_b), które są używane do określenia narożników prostokąta. Pierwszy trójkąt, który tworzy dolną część prostokąta, jest rysowany na zielono. Kolejne współrzędne wierzchołków definiują dolny lewy, dolny prawy i górny lewy róg. Drugi trójkąt tworzy górną część prostokąta i jest rysowany na czerwono. Współrzędne wierzchołków określają dolny prawy, górny prawy i górny lewy róg.

Efekt wykonanej pracy



Rys. 2 – zadanie 2 efekt końcowy

Zadanie 3

Cele ćwiczenia

Celem zadania trzeciego było wykorzystanie wcześniejszego kodu (zadanie 2) rysującego prostokąt z dwóch trójkątów do napisania logiki deformacji boków prostokąta oraz wprowadzenia losowości jego koloru.

Realizacja celu

W tym zadaniu napisałem funkcje, która bazuje na mojej poprzedniej funkcji rysującej prostokąt, jednakże tutaj musiałem uwzględnić logikę deformacji boków prostokąta. Wybrany sposób deformacji, to odpowiednie skalowanie boków a i b prostokąta. Efekt skalowania jest losowy, to znaczy, że parametr skalujący boki (u mnie d), jest generowany losowo przez moduł Pythona – random. Dodatkowo generowany jest losowo również kolor wypełnienia figury.

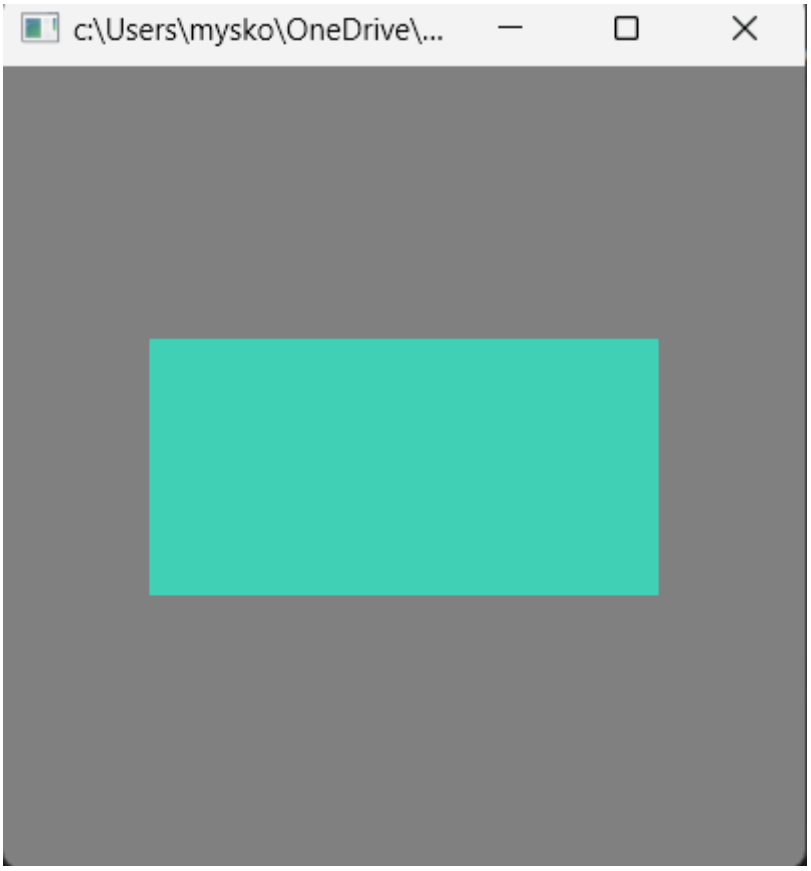
Omówienie kodu oraz jego działania

Funkcje **startup(), shutdown(), update_viewport()** oraz **main()** nie różnią się w zasadzie niczym od poprzedniego kodu z zadania 1, który został opisany powyżej. Jedyna drobna różnica dotyczy funkcji **render(time)**, która polega na tym, że wywołujemy w niej własną funkcję **rysuj_prostokat(0.0, 0.0, 100.0, 50.0, deformacja, kolor)** – która to zostanie opisana w następnym akapicie.

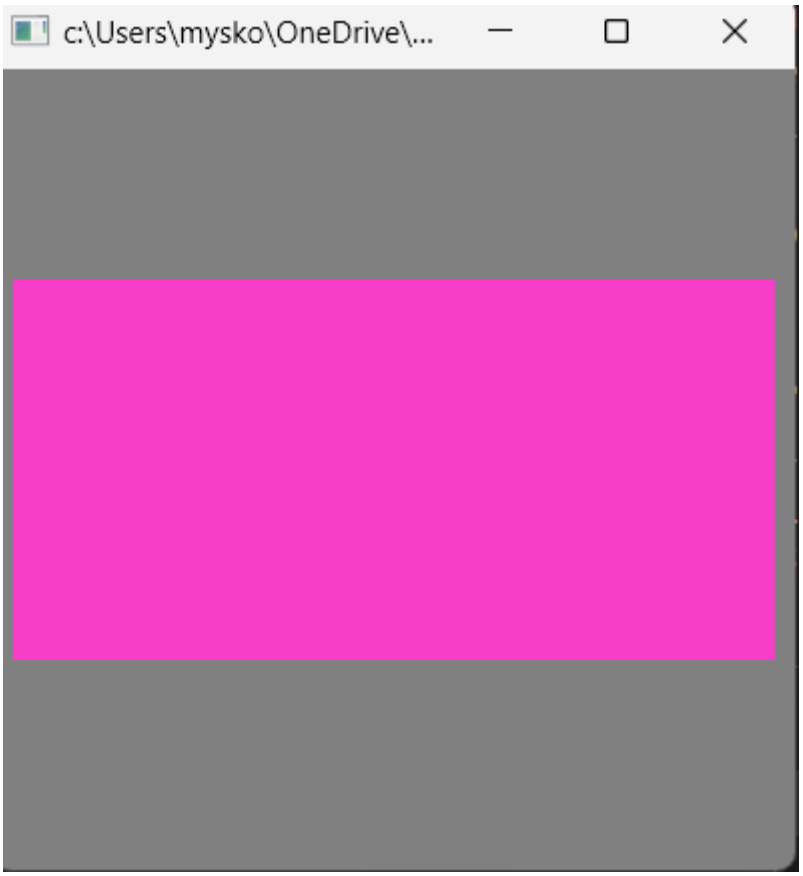
Funkcja **rysuj_prostokat(0.0, 0.0, 100.0, 50.0, deformacja, kolor)** ma za zadanie obliczenie odpowiednich współrzędnych wierzchołków prostokąta oraz narysowanie. Przyjmuje pięć argumentów: x i y to współrzędne środka prostokąta, a i b to długości boków prostokąta, a d to współczynnik deformacji, który domyślnie ustawiony jest na 1.0. Dodatkowo przekazujemy kolor, który definiuje kolor prostokąta. Na początku funkcji obliczane są przeskalowane wymiary prostokąta na podstawie współczynnika deformacji. Zmienne **przeskalowany_bok_a** oraz **przeskalowany_bok_b** to długości boków prostokąta po zastosowaniu deformacji. Oznacza to, że gdy d wynosi mniej niż 1.0, prostokąt będzie się zmniejszał, a gdy większy niż 1.0, będzie się powiększał. Następnie funkcja oblicza współrzędne czterech wierzchołków prostokąta, korzystając z przeskalowanych wymiarów. Współrzędne te są obliczane w taki sposób, że prostokąt jest wyśrodkowany w punkcie (x, y). Wartości x1, y1, x2, y2, x3, y3, x4, y4 reprezentują dolny lewy, dolny prawy, górny prawy oraz górny lewy róg prostokąta, odpowiednio. Dzięki temu rysowany prostokąt ma odpowiednie wymiary i jest umiejscowiony w odpowiednim miejscu w układzie współrzędnych. Kolor również jest wybierany losowo za pomocą takiej samej koncepcji jak w przypadku współczynnika skalowania. Za strukturę przechowywania koloru w Pythonie wybrałem krotkę.

Efekt wykonanej pracy

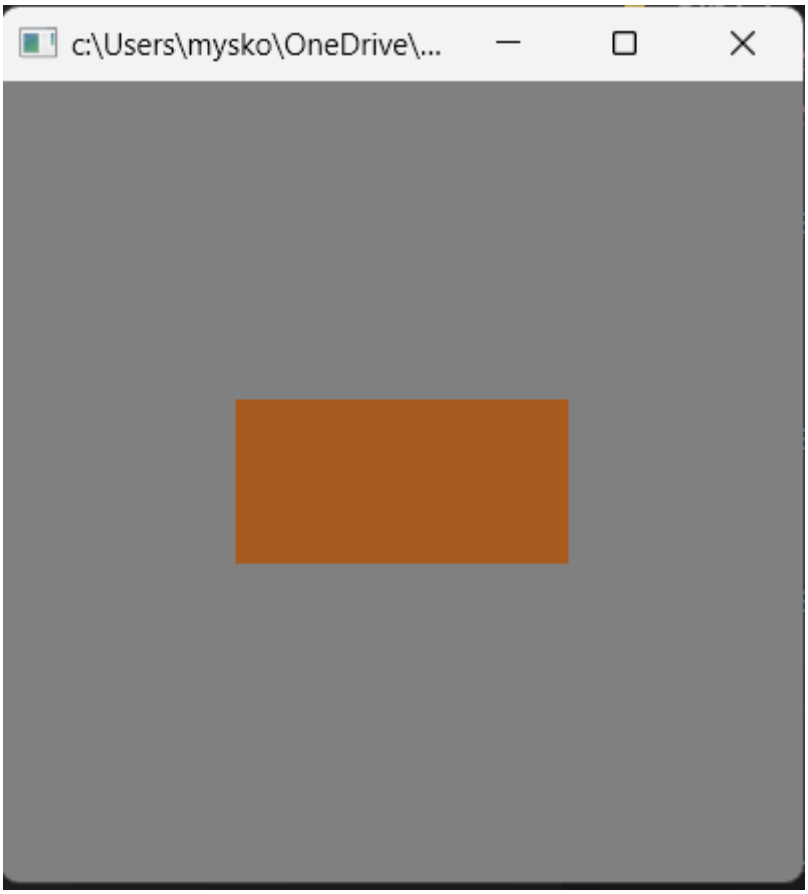
Poniżej zostały przedstawione trzy uruchomienia programu, tak aby pokazać pełne działanie kodu obsługującego logikę losowej deformacji prostokąta wraz z losowym kolorem jego wypełnienia.



Rys. 3 – zadanie 2 efekt końcowy: pierwsze losowanie parametrów



Rys. 4 – zadanie 2 efekt końcowy: drugie losowanie parametrów



Rys. 5 – zadanie 2 efekt końcowy: trzecie losowanie parametrów

Podsumowanie

Powyżej wykonane zadana dały mi silne podstawy z zakresu podstawowych definicji obiektów i funkcjonalności graficznych przy użyciu biblioteki OpenGL. Wykonywanie każdego z wyżej zaprezentowanych zadań wymagało z każdym etapem coraz więcej dokładania od siebie swoich pomysłów na realizację.

Wyżej zrealizowane zadania wymagały zrozumienia jak jest reprezentowana grafika w komputerze, jak są wykonywane obliczenia oraz jak jest prezentowana na ekranie.

Zadania również przybliżyły koncepcję modelowania graficznego za pomocą podstawowej jednostki, jaką jest tzw. prymityw (na przykład: wierzchołek, linia, trójkąt). Za pomocą prymitywów komputer tworzy bardziej zaawansowane wizualizację.

Powyżej wykonane zadania nie należały do trudnych ale były kluczowe w zrozumieniu wyżej wymienionych konceptów grafiki oraz zapoznały mnie ze środowiskiem programowania w OpenGL.