Silnik 3D

Stworzony na laboratorium podstawy grafiki komputerowej w roczniku 2024/205

Autorstwa:
Szymon Żuber
Michał Lesiak
Wiktor Baranowicz

Stworzony z pomocą OpenGl4/freeglut w C++

PODZIAŁ PRACY:

Wpływ każdego z autorów nie był ograniczony jedynie do podanych dalej plików większość z nich była robiona wspólnie lub wspólnie poprzez iterację, lecz podział pracy po plikach wychodzi mniej więcej w taki sposób:

Szymon Zuber:

DisplayManager.h
KeyboardHandler.h
MouseHandler.h
BitmapHandler.h
Engine.h
ObjectManager.h
Renderer.h
ShaderHandler.

Michał Lesiak:

Line.h
Point.h
Poliline.h
Quads.h
QuadsTextured.h
Triangle.h
TriangleFan.h
TriangleStrip.h
Cube.h
CubeTextured.h
FigureE.h

Wiktor Baranowicz:

DirectDraw.h

EngineObject.h
Figures.h
IndiceDraw.h
Observer.h
Player.h
Refreshable.h
Resizable.h
Textured.h
TransformableFigure.h

SPIS TREŚCI:

OBSŁUGA SILNIKA:

1. Inicjalizacja silnika w programie

Aby podłączyć silnik do tworzonego programu należy do przynajmniej jednego pliku na którym się pracuje i jest połączony ze wszystkimi innymi, w których będą używane funkcje silnika załączyć dopisek:
#include “Engine.h”
Po wcześniejszym dołączeniu wszystkich plików silnika do katalogu z plikami użytkownikami.
W tworzonym projekcie należy również umieścić następujące makro związane z obsługą bitmap w jednym miejscu, najpewniej tam gdzie jest funkcja main całego programu.
#define STB_IMAGE_IMPLEMENTATION

Do poprawnego działania silnika potrzebne są instancje następujących klas lub klasy po ich dziedziczące:
ObjectManager - klasa zajmująca się obiektami

Renderer - klasa odpowiedzialna za rysowanie widoku
DisplayManager - klasa odpowiadająca za działanie wyświetlacza
Engine - główna klasa silnika
Następnie można dodać wywołania mające na celu włączenie opcjonalnych funkcji silnika co będzie omówione w następnych rozdziałach.
W przeciwnym wypadku, jeśli nie jest używana myszka i klawiatura, należy stworzyć pustą funkcję i po stworzeniu obiektu klasy Engine wywołać następujące funkcję:
toggleMouse(false, std::bind(nullfunc));
toggleKeyboard(false, std::bind(nullfunc));
Na samym końcu należy wywołać funkcję glutMainLoop() rozpoczynającą główną pętle programu.

Aby poprawnie działał program należy obiekt klasy Engine stworzyć po wcześniejszym stworzeniu obiektów klasy Renderer i DisplayManager.

2. Obsługa klawiatury i myszy

Opcjonalnie można dodać klasy odpowiadające za wejście klawiatury i myszki lub bardziej prawdopodobniejsze, klasy po nich dziedziczące, zawierające implementację różnych efektów po naciśnięciu poszczególnych klawiszy:
KeyboardHandler i MouseHandler.
W razie chęci użycia myszki i klawatury należy dodatkowo po stworzeniu instancji klas zajmujących się nimi dodać po stworzeniu obiektu klasy Engine następujące wywołania:
toggleMouse(true, std::bind(-wskaźnik na funkcję-, -wskaźnik na obiekt-));
toggleKeyboard(true, std::bind(-wskaźnik na funkcję-, -wskaźnik na obiekt-));
gdzie wskaźnikami na funkcje są wskaźniki na specjalnie stworzone funkcje, które obsługują naciśnięcia przycisków i klawiszy - sam efekt naciśnięcia poszczególnych klawiszy, a wskaźnik na obiekt to wskaźnik na obiekt klas które to obsługują.

Klasa KeyboardHandler korzysta dodatkowo z pomocniczej struktury KeyStates zawierająca stany poszczególnych klawiszy:
naciśnięty - pressing, nie naciśnięty - notClicked oraz naciśnięty ale do usunięcia w następnej klatce - removeInFuture.
Z perspektywy tworzącego program jest to kompletnie niewidoczne.
Użytkownik powinien skorzystać z następujących funkcji w swojej klasie:

- refresh()- funkcja odświeża stan klawiatury, co powinno być wywołane pod sam koniec spersonalizowanej funkcji obsługi klawiszy
- setIfShouldRefresh(przycisk, true/false) - funkcja ustawia czy dany klawisz powinien być odświeżany z każdą klatką gry, co zapobiega kliknięciu danego przycisku parę razy podczas przytrzymywania klawisza
- checkIfPressed - funkcja sprawdzająca czy dany klawisz został naciśnięty(zgodnie z tabelą ascii)

Dostępne są również funkcje keyDown i keyUp odpowiadające za zachowanie podczas naciśnięcia/odpuszczenia klawisza, które są domyślnie zaimplementowane, lecz można je odłączać/podłączać stosując metody freeglut takie jak glutKeyboardFunc, w celu osiągnięcia chcianego efektu - nie sprawdzanie naciskania klawiszy podczas pisania w konsoli np. Funkcja obsługująca klawisze należy przekazać używając toggleKeyboard i wtedy będzie ona wykonywana co klatkę rozgrywki.

Klasa MouseHandler korzysta analogicznie dodatkowo z pomocniczej struktury KeyStates oraz struktury mouseButtons nadające nazwy przyciskom myszki - leftButton, rightButton, scrollButton.
Użytkownik powinien skorzystać z następujących funkcji w swojej klasie:

- refresh()- funkcja odświeża stan myszki, co powinno być wywołane pod sam koniec spersonalizowanej funkcji obsługi myszki
- setIfShouldRefresh(przycisk, true/false) - funkcja ustawia czy dany przycisk powinien być odświeżany z każdą klatką gry, co zapobiega kliknięciu danego przycisku parę razy podczas przytrzymywania przycisku
- checkIfPressed - funkcja sprawdzająca czy dany przycisk został naciśnięty(można zastosować nazwy ze struktury mouseButtons)

Dostępne są również funkcje buttonHandle i mouseCallback odpowiadające za zachowanie podczas naciśnięcia/odpuszczenia przycisku oraz ruchu myszki, które są domyślnie zaimplementowane, lecz można je odłączać/podłączać stosując metody freeglut takie jak glutMouseFunc, w celu osiągnięcia chcianego efektu.
Klasa zawiera również dodatkowe funkcje do obsługiwania klasy observer(kamery).
Funkcja obsługująca klawisze należy przekazać używając toggleMouse i wtedy będzie ona wykonywana co klatkę rozgrywki.

3. Obsługa kamery

Domyślnie w programie kamera będzie statyczna. Aby zmienić to należy stworzyć obiekt klasy Observer, po wcześniejszej inicjalizacji klas Engine i Renderer przekazując w konstruktorze obiekt klasy Shader z renderera używając getShader(). Następnie wystarczy w obiekcie klasy ObjectManager i MouseHandler(lub dziedziczącym) wywołać setCamera() przekazując wskaźnik na obiekt klasy Observer.

Klasa observer metody pozwalające na manualną kontrolę widoku, lecz nie jest to potrzebne do działania programu.

4. Obsługa wyświetlacza

Klasa odpowiedzialna za wyświetlacz do DisplayManager. Domyślnie do poprawnego działania wystarczy stworzenie instancji tej klasy i przekazanie ją do instancji klasy Engine, lecz można wykorzystać dodatkowe funkcje to większej personalizacji. Do konstruktora można przekazać szerokość i wysokość okna oraz czy obraz ma być na pełnym ekranie co można później zmienić odpowiednimi funkcjami, lecz również można zmienić dwie rzeczy, które są potem nie do zmienienia podczas działania programu, ustawić tytuł całego programu

oraz czy program powinien wykorzystywać podwójne buforowanie. Domyślnie tytuł będzie “Engine” oraz podwójne buforowanie będzie wykorzystywane.

5. Obsługa renderowania

Klasą odpowiedzialną za renderowanie jest Renderer. Do stworzenie instancji tej klasy potrzebne jest wcześniejsze stworzenie klasy ObjectManager co zostanie omówione później. Podczas tworzenia instancji obiektu tej klasy można dodatkowo również podać argumenty odpowiadające za kolor czyszczenia obrazu(tła), czy powinien być używany ZBufor oraz czy powinien być używany widok ortogonalny czy perspektywistyczny, lecz wszystkie te zmienne można zmienić potem podczas działania programu. Klasa ta również odpowiada za poprawne działanie shaderów używając pliki: flatShader.frag/vert oraz shader.frag/vert, lecz jest to ukryte przed samym użytkownikiem.

Dodatkowo można ustawić pewne parametry dotyczące shaderów za pomocą paru funkcji, m.in. zmieniając siłę poszczególnych oświetleń, pozycję światła oświetlającego scenę, zwiększenie kontrastu koloru, przełączanie się między płaskim cieniowaniem, a gładkim oraz przełączanie się między modelem oświetlenia globalnego, a “latarkowego”.

6. Obsługa obiektów

Obsługa silnika dzieje się za pomocą instancji klasy ObjectManager. Za jej pomocą można dodawać/usuwać obiekty do rysowania typów, IndiceDraw, IndiceDraw(oteksturowane) oraz DirectDraw. Dodatkowo można ustawić w niej kamerę(w celu użycia jej potem przez klasę Renderer).
Aby gracz był wyświetlany należy go również dodać do tej klasy - domyślnie przez addIndicedDrawable.
Klasa również dodaje/usuwa poszczególne obiekty do buforów do rysowania związanymi z opengl, a więc nie musi się tym przejmować użytkownik. Niejawnie przechowywane są też dane związane z tym, które wierzchołki należą do której figury i w jakiej kolejności je łączyć w przypadku rysowania indeksowego.

7. Obsługa bitmap

Obsługa bitmap jest za pomocą klasy BitmapHandler, zawierającej wyłącznie funkcję do usuwania danej bitmapy i ładowania ich z plików. W tym celu wykorzystywana jest krótka biblioteka stbi. W celu ustawienia danego obiektu na używanie tej klasy należy najpierw ją załadować za pomocą loadBitmap i przekazać jej id do zmiennej typu int, a następnie wskazać figurę którą chcemy oteksturować i użyć funkcji setTextured(jeśli dziedziczy ona po klasie Textured).
Klasy które domyślnie można teksturować mają końcówkę Textured. Aby klasa mogła być teksturowana wystarczy żeby dziedziczyła po textured i znajdowała się w vektorze w ObjectManager odpowiedzialnym za tego typu klasy. Warto jednak pamiętać że w takim samym formacie wpisuje się dane figury (3 glm::vec4 na wierzchołek), lecz 3 vektor w tym

przypadku to nie kolor, a wskazanie pozycji na teksturze. Program domyślnie pozwala na jednoczesne wykorzystanie jednej bitmapy.

8. Obsługa silnika

Główną klasą całego silnika jest klasa Engine. Aby poprawnie stworzyć program należy przed stworzeniem instancji tej klasy utworzyć instancję klasy Renderer i DisplayManager. Stwarzając instancję, przez konstruktor dodatkowo przekazać następujące dane: zmienne zawierające dane wejściowe całego programu oraz liczbę planowanych fps.
Klasa dodatkowo zawiera funkcje toggleKeyboard oraz toggleMouse, które włączają/wyłączają funkcjonalność myszki/klawiatury oraz funkcje pozwalające na ustawienie liczbę odświeżania na sobie wybraną. Ważnym do dodania jest że biblioteka freeglut nie pozwala na ustawienie limitu wyżej niż liczba częstotliwości odświeżania monitora w większości przypadków.

9. Obsługa gracza

Klasa będąca graczem to klasa Player. Domyślnie dziedziczy ona po klasie Cube, a więc obiekt gracza będzie sześcianem. Klasa ta pozwala na ustawienie prędkości gracza we wszystkich 3 osiach. Jeśli instancja klasa istnieje, co klatkę będzie wykonywany ruch obiektu, zgodnie z prędkościami, a następnie będą one zerowane.

10. Podstawowe klasy obiektów programu

Podstawową klasą wszystkich obiektów jest klasa EngineObject. Zawiera ona w sobie mechanizm ustawiania identyfikatorów oraz czy dany obiekt powinien być aktywny, lecz leży to po stronie programisty używającego silnika, aby ustawiać danym obiektom unikalne identyfikatory etc. Domyślnie to czy obiekt jest aktywny nie wpływa na nic, powinno to być zaimplementowane przez programistę.

Klasa Figures jest podstawową klasą wszystkich figur. Zawiera ona zmienną, która wskazuje czy dany obiekt jest teksturowany oraz vektor mający przechowywać wszystkie wierzchołki danej figury. Zapewnia ona również dostęp do tych zmiennych oraz dodatkowe funkcje obliczające liczbę wierzchołków danego obiektu oraz zwiększającą indeks przekazywany do funkcji o liczbę wierzchołków danej figury.

Klasa DirectDraw jest klasą wszystkich obiektów, które są rysowane za pomocą glDrawArrays. Zawiera ona w sobie wirtualną funkcję drawDirect.

Klasa IndiceDraw jest odpowiednikiem DirectDraw, lecz dla obiektów rysowanych za pomocą indeksów przy pomocą glDrawElements. Zawiera ona w sobie vektor indeksów oraz funkcję do dostępu i zmiany go.

11. Dodatkowe klasy modyfikujące obiekty

Silnik jest również wyposażony o dodatkowe klasy modyfikujące podstawowe klasy obiektów.

Klasa refreshable zawiera zmienną oraz metody związane ze zgłaszaniem, że bufor figur do rysowania powinien być odświeżony. Domyślnie wszystkie figury dzedziczą po niej.

Klasa resizable zawiera zmienną oraz metody związane ze zmienianiem poszczególnych rozmiarów figur. Nie tyczy się to jednak rozmiaru całej figury tylko bardzo szerokość linii w klasie Line i PoliLine oraz wielkość punktu w Point.

Klasa textured zawiera zmienną oraz metody związane z ustawianiem identyfikatorów tekstury. Domyślnie jedyne klasy po niej dziedziczące mają końcówkę Textured.

Klasa transformableFigure jest rozszerzeniem klasy Figures, pozwalającym wykonywanie na danym obiekcie transformacji, albo przez zdefiniowane metody translate, rotate, scale ułatwiające transformacje lub używając metodę freeTransform pozwalającą na zastosowanie dowolnej macierzy do transformacji. Domyślnie wszystkie klasy dostarczone przez nas pozwalają na transformacje na nich.

12. Klasy obiektów rysowanych bezpośrednio

W silniku zostały domyślnie zaimplementowane klasy do rysowania bezpośredniego: Linii, punktów, wielolinii, czworokątów, czworokątów teksturowanych, trójkątów, wachlarzy trójkątnych i pasków trójkątnych. Aby były rysowane należy je dodać do ObjectManager.

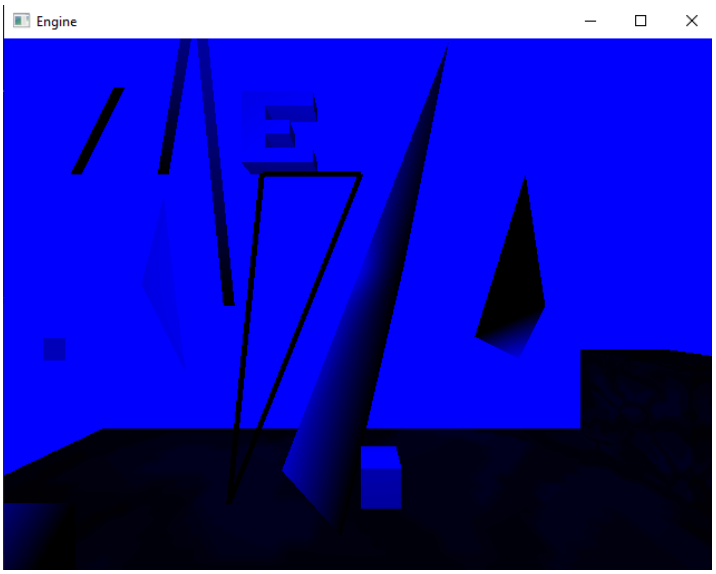
13. Klasy obiektów rysowanych za pomocą indeksów

W silniku zostały domyślnie zaimplementowane klasy do rysowania indeksowanego: sześcianu, sześcianu teksturowanego i litery E w 3d. Aby były rysowane należy je dodać do ObjectManager.

14. Demo technologiczne

Domyślny widok programu po włączeniu demo:

Zmienienie koloru wyczyszczenia na czysto-niebieski:



Zmiana widoku po zmienieniu paru opcji oświetlenia

Płaskie cieniowanie: