Programowanie w API Graficznych

Laboratorium

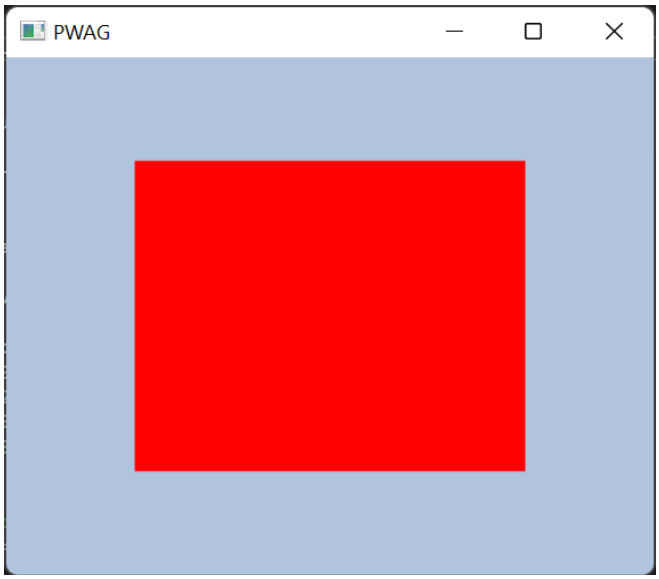
DirectX 12 – ćwiczenie 2

1 Cele ćwiczenia

Celem ćwiczenia jest poznanie procesu tworzenia, wypełniania, wiązania i wykorzystywania buforów stałych. Zaczniesz również powoli zapoznawać się z HLSL-em i jego semantykami. Mimo że bufory stałych zostały wprowadzone w D3D10, są tylko alternatywną wersją funkcjonalności, która była dostępna w D3D9. W trakcie zadania utworzysz dwa bufory stałych, pierwszy z nich zwierać będzie macierz widoku i projekcji. Modyfikowanie tego bufora umożliwi sterowanie kamerą. Drugi bufor stałych zawierać będzie macierz obiektu, modyfikowanie tego bufora umożliwi przemieszczanie obiektu w scenie 3D.

2 Zadania

Dołączony do zadania projekt posiada w pełni skonfigurowany potok graficzny (klasa `RenderWidget`), który wyświetla prosty sześcian oraz okno aplikacji (klasa `System`) z zaimplementowaną pętlą komunikatów. Program cieniujący znajduje się w pliku `shader.fx`, kod tego programu jest automatycznie kompilowany w momencie uruchomienia aplikacji. Zaraz po otwarciu projektu załączonego do zadania sprawdź czy projekt poprawnie kompiluje się i uruchamia w twoim IDE. Po uruchomieniu powinieneś zobaczyć wyrenderowany czerwony sześcian na szarym tle.



2.1 Kolorowe wierzchołki

Zmodyfikuj kod tworzenia wierzchołków, input layout oraz programy cieniujące (ang. shaders) tak aby wierzchołki poza informacją o położeniu zawierały także informację o kolorze.

2.1.1 Wierzchołki

Zacznij od zmodyfikowania definicji wierzchołka poprzez dodanie do niej zmiennej typu `XMFLOAT3`, która będzie przechowywać informację o kolorze w formacie RGB. Definicja wierzchołka (zamieszczona poniżej) znajduje się w pliku `GeometryHelper.h`. W tym samym pliku znajduje się także funkcja `GeometryHelper::CreateBoxGeometry()`, która odpowiada za wygenerowanie wierzchołków wyświetlanego sześcianu, ten fragment kodu także należy zmodyfikować.

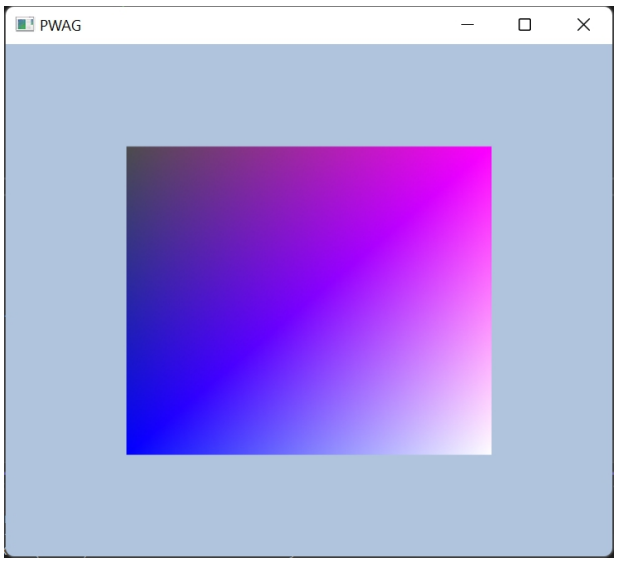
```
struct Vertex
{
	Vertex(DirectX::XMFLOAT3 position) :Position(position){}
	DirectX::XMFLOAT3 Position;
	//Dodaj zmienna przechowującą kolor
};
```

2.1.2 Input Layout

Następnie w funkcji `CreateGraphicPipeline()` znajdziesz obiekt *`inputLayout`*. Zawiera on opis struktury danych (wierzchołków) umieszczonych w buforze wierzchołków, aktualnie. Do tego obiektu także należy dodać informację o kolorze, tak aby zmodyfikowane przez ciebie wcześniej wierzchołki zostały poprawnie wczytane do programu cieniującego wierzchołki (ang. Vertex shader).

2.1.3 Program cieniujący wierzchołki i piksele

Teraz pozostaje już tylko modyfikacja kodu shadera – zmień strukturę wejściową w programie cieniującym wierzchołki (plik `shader.fx`), pamiętając o tym, aby dobrze ustawić semantyki (tak jak w obiekcie *inputLayout*). Struktura wyjściowa programu cieniującego wierzchołki powinna odpowiadać strukturze wejściowej programu cieniującego piksele – obie powinny zawierać współrzędne wierzchołka oraz jego kolor. Jeżeli poprawnie wykonałeś wszystkie kroki to powinieneś zobaczyć kolorowy sześcian (tak ja na obrazku poniżej).



2.2 Kamera i pierwszy bufor stałych

W tym zadaniu wykorzystasz już gotowe macierze widoku i projekcji. Kod odpowiedzialny za wyznaczanie obu macierzy znajduje się w funkcji `UpdateViewProjectionCBuffer()`, natomiast wspomniane macierze są zapisane w obiekcie `cameraConstants`. Wspomniana funkcja `UpdateViewProjectionCBuffer()` jest wywoływany za każdym razem gdy użytkownik zmieni rozmiar okna programu lub wciśnie lewy przycisk myszki (obrót sześcianu) bądź prawy przycisk myszki (zmiana rozmiaru sześcianu) i poruszy myszką.

2.2.1 Tworzenie bufora stałych

Bufor stałych, podobnie jak inne bufory danych (np. tekstury) jest zasobem karty graficznej. Kod odpowiedzialny za utworzenie bufora należy umieścić w funkcji `RenderWidget::CreateContantBuffers()`, bufor można utworzyć za pomocą funkcji `CreateCommittedResource()`. Macierze widoku i projekcji przechowywane są w obiekcie typu `CameraConstants`, tworzony bufor musi być na tyle duży aby pomieścić ten obiekt. Należy takżże pamiętać, że rozmiar bufora musi być wielokrotnością 256 bajtów, aby ułatwić ci zadanie napisana została funkcja `DirectXHelper::CalcConstantBufferByteSize`, która wyznaczy poprawny rozmiar bufor stałych.

Następnie, jeżeli dysponujesz już obiektem bufora stałych, będziesz musiał zaalokować współdzieloną pamięć, która umożliwi ci skopiowanie macierzy widoku i projekcji z pamięci RAM do pamięci GPU. Służy do tego funkcja Map, poprawnie wywołana zwróci wskaźnik do którego w kolejnym zadaniu skopiujesz wspomniane macierze. Niepotrzebny bufor stałych można zwolnić za pomocą funkcji `Unmap()`.

2.2.2 Root signature

W DirectX12 nie przypisujemy zasobów bezpośrednio do poszczególnych etapów potoku renderującego, tak jak to miało miejsce w poprzednich wersjach, lecz do obiektu ***Root Signature***. Nim to jednak zrobisz to musisz ten obiekt wcześniej stworzyć, kod za to odpowiedzialny znajduje się w funkcji `RenderWidget::BuildRootSignature`. Już na tym etapie należy zasygnalizować jakie zasoby będą użyte aby DirectX zarezerwował dla nich miejsce, służą do tego parametry – CD3DX12_ROOT_PARAMETER. Twoim zadaniem jest poprawne skonfigurowanie obiektu *`m_rootSignature`* tak aby możliwe było podpięcie bufora stałych do potoku renderującego.

2.2.3 Podpięcie bufora stałych do potoku renderującego

W tym zadaniu utworzony wcześniej bufor stałych podepniesz do potoku renderującego, zrób to zaraz po podpięciu obiektu Root Signature (m_rootSignature). Służy do funkcja `SetGraphicsRootConstantBufferView`.

2.2.4 Zmiana zawartości bufora stałych

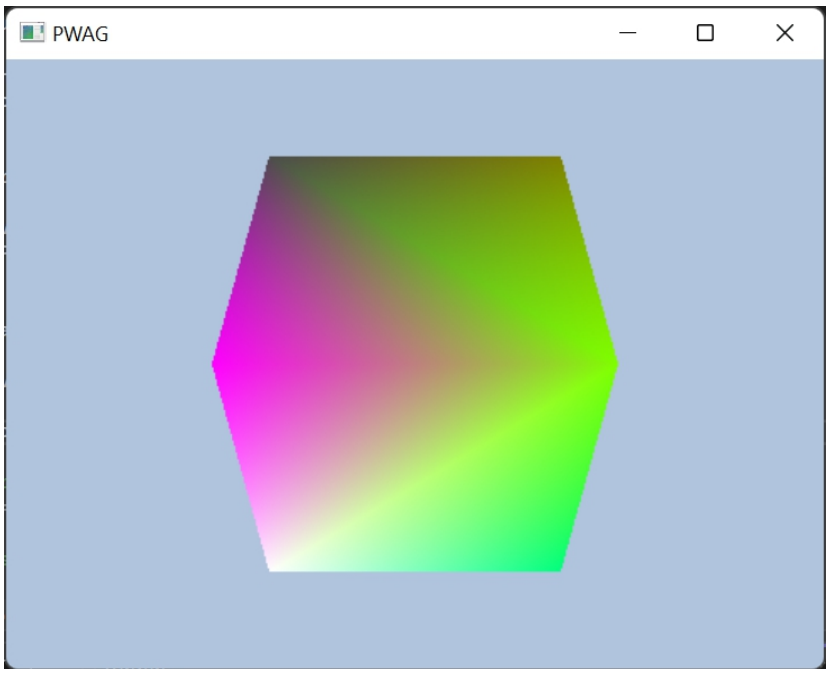
Zawartość bufora możesz zaktualizować poprzez skopiowanie danych pod otrzymany za pomocą funkcji `Map()` adres w pamięci RAM. Kod za to odpowiedzialny należy umieścić w wspomnianej wcześniej funkcji `UpdateViewProjectionCBuffer()`. Czy wiesz dlaczego obie macierze (widoku i projekcji) są transponowane za pomocą funkcji `XMMatrixTranspose` przed zapisaniem ich w buforze stałych ?

2.2.5 Program cieniujący wierzchołki

Pozostało już tylko zaktualizować program cieniujący wierzchołki. W tym celu powinieneś utworzyć obiekt który będzie reprezentować macierz widoku i projekcji (przykładowy kod umieszczono poniżej, pamiętaj aby ustawić odpowiedni rejestr!) i wykorzystać programie cieniującym wierzchołki.

```
cbuffer cbCamera : register(?)
{
	float4x4 gViewMatrix;
	float4x4 gProjMatrix;
};
```

Jeżeli wykonałeś wszystkie kroki poprawnie to powinieneś być teraz w stanie manewrować za pomocą myszki sześcianem. Lewy przycisk myszki odpowiada za obracanie kamery, a prawy przycisk za oddalenie.



2.3 Utworzenie drugiego bufora stałych

Ostatnie zadanie jest bardzo podobne do poprzedniego, twoim zadaniem jest utworzenie drugiego bufora. Tym razem bufor ten będzie przechowywać macierz świata/obiektu sześcianu., która znajduje się w funkcji `RenderWidget::UpdateWorldCBuffer()`. Funkcja ta jest wywoływana za każdym razem gdy użytkownik kliknie którykolwiek przycisk strzałki na klawiaturze. Podobnie jak wcześniej, będziesz musiał utworzyć nowy zasób, zaktualizować obiekt Root Signature, zaktualizować programy cieniujące i podpiąć nowo-utworzony bufor stałych do potoku renderującego. Jeżeli poprawnie wykonasz wszystkie kroki to będziesz w stanie przemieszczać sześcian za pomocą strzałek na klawiaturze.