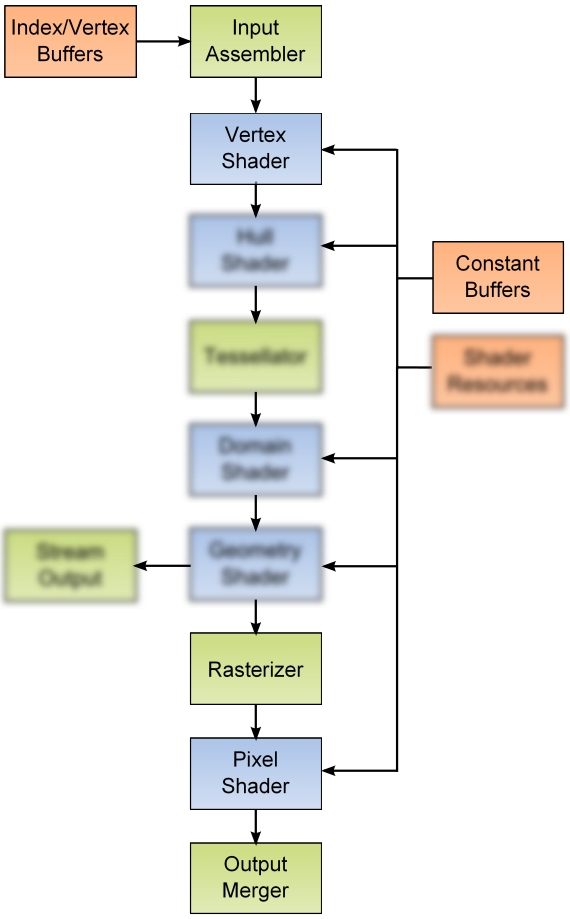
Programowanie w API Graficznych

Laboratorium

Direct3D 11 - ćwiczenie 2

Jakub Stępień, 2012



Rysunek 1: Stadia potoku D3D11 wykorzystane w ramach ćwiczenia 2

1 Cel ćwiczenia

Celem ćwiczenia jest poznanie procesu tworzenia, wypełniania, wiązania i wykorzystywania buforów stałych. Zaczniesz również powoli oswajać 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. Oznacza to, że nadal możemy pozostać na *feature level* `D3D_FEATURE_LEVEL_9_1` i odpowiadających mu modelach shaderów.

2 Zadania

2.1 Zadanie 1 - kolorowe wierzchołki

Zmodyfikuj kod tworzenia wierzchołków, *input layout*-u i shaderów tak, by bazował na wierzchołkach typu `Vertex_PosCol`.

2.1.1 Wierzchołki

Zacznij od zmodyfikowanie `typedef`-u w linii siódmej pliku `renderwidget.cpp`. Teraz zaktualizuj generację wierzchołków, uwzględniając w konstruktorze trzy dodatkowe parametry - komponenty czerwony, zielony i niebieski w zakresie `[0.f, 1.f]`. Propozycja kolorowania: jeden wierzchołek czerwony, jeden zielony, jeden niebieski i jeden biały.

2.1.2 Input layout

Następnie w metodzie `dxCreateInputLayouts()` utwórz i dodaj do `vertexBufferElements` kolejną strukturę `D3D11_INPUT_ELEMENT_DESC`, wzorując się na gotowym już `posElement` i zaglądając w razie potrzeby do dokumentacji (link w komentarzach). Propozycja wyboru semantyki: `COLOR`, indeks `0`.

2.1.3 Shadery

Teraz pozostaje już tylko modyfikacja kodu shadera - zmień strukturę wejściową vertex shadera, pamiętając o tym, by dobrze ustawić semantyki (zgodnie z pierwszymi dwoma parametrami `CreateInputLayout`); struktura wyjściowa vertex shadera (= wejściowa pixel shadera) musi teraz zawierać również kolor wierzchołka/fragmentu, a pixel shader powinien go po prostu zwracać.

2.2 Kamera i pierwszy cbuffer

W tym zadaniu wykorzystasz (gotowe już) macierze widoku i rzutowania, by uzyskać sterowaną myszką kamerę:

- `m_ViewMatrix` - macierz widoku
- `m_ProjectionMatrix` - macierz rzutowania
- `m_ViewProjectionMatrix` - iloczyn powyższych

Zadanie w całości dotyczy bufora/wskaźnika `m_pConstantBuffer_scene`, który będzie zawierać stałe ustawiane *tylko raz* dla całej sceny. W tej chwili jest to tylko macierz *view-projection*, ale mogłyby tu się znaleźć np. zmienne opisujące kierunek, natężenie i kolor głównego światła kierunkowego/słońca.

2.2.1 Tworzenie bufora stałych

W metodzie `dxCreateConstantBuffers()` uzupełnij pola struktury konfigurującej zasób:

- flagę wiązania (zajrzyj do wprowadzenia, jeżeli nie wiesz jaką)
- wielkość bufora stałych (`ByteWidth`)

Pozostałe składowe są już ustawiona poprawnie.

2.2.2 Zmiana zawartości bufora stałych

Zignoruj na razie parametry `sceneConstantsOutdated` i `objectConstantsOutdated`. W metodzie `dxConfigureVertexShaderStage(...)` wykorzystaj `memcpy`, by skopiować do bufora zawartość `m_ViewProjectionMatrix`. Pamiętaj, że obszar pamięci, do którego masz zapisać, jest poprawnie zmapowany tylko pomiędzy odpowiadającą sobie parą wywołań `Map` i `Unmap`.

2.2.3 Wiązanie bufora stałych do VS

W tej samej metodzie wykorzystaj `VSSetConstantBuffers(...)`, by przywiązać utworzony bufor stałych do stadium Vertex Shader. Metoda oczekuje podania *slotu* pierwszego z *tablicy ustawianych buforów* oraz ich ilości. W naszym przypadku jest to oczywiście odpowiednio: `0` i `1`.

2.2.4 Vertex Shader

Teraz musisz jeszcze tylko utworzyć odpowiadający obiekt zasobu HLSL i wykonać w VS odpowiednie mnożenie (`mul(...)`).

2.3 Pozycjonowanie modelu i drugi cbuffer

W tym zadaniu wykorzystasz (gotową już) macierz transformacji *świata* (zmienna lokalna `worldMatrix`), by móc przesuwać prostokąt po świecie/scenie. Wykonaj kroki analogiczne do poprzedniego zadania. Na translację modelu *powinny* wpływać trzy suwaki dostępne w panelu bocznym aplikacji.

Zadanie w całości dotyczy bufora/wskaźnika `m_pConstantBuffer_object`, który będzie zawierać stałe ustawiane dla każdego modelu. Zauważ, że na poziomie aplikacji korzystasz z dwóch buforów stałych - ile trzeba ich zdefiniować w HLSL-u ? To *nie jest* podchwytliwe pytanie.

Wiążąc bufor(y) stałych do potoku, zwróć uwagę na parametry `sceneConstantsOutdated` i `objectConstantsOutdated`. Zastanów się, jaki jest ich sens i jak mógłbyś je wykorzystać do wyeliminowania zbędnych wywołań API i operacji pamięciowych (wydajność!).