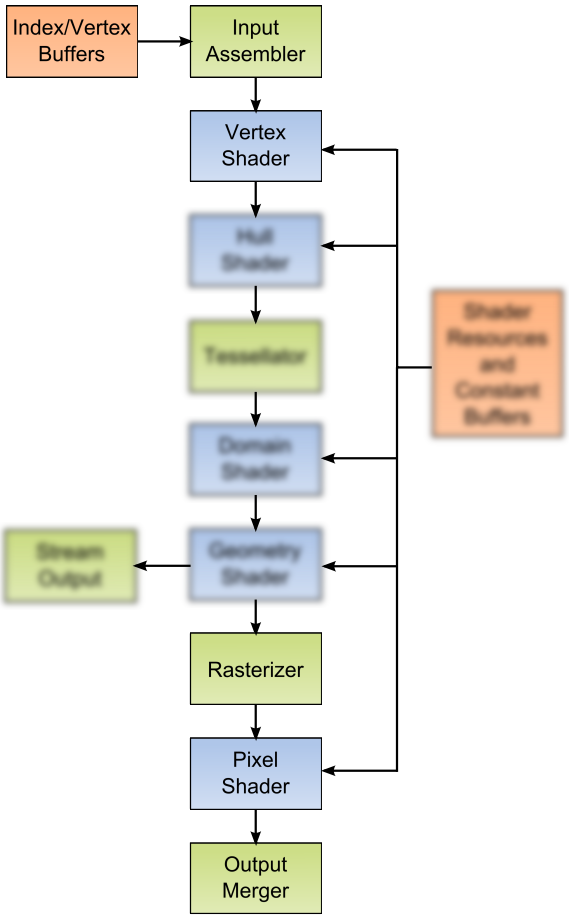
Programowanie w API Graficznych

Laboratorium

Direct3D 11 - ćwiczenie 1

Jakub Stępień, 2012



Rysunek 1: Stadia potoku D3D11 wykorzystane w ramach ćwiczenia 1

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawową strukturą aplikacji wykorzystującej Direct3D 11, prostymi typami zasobów, obiektów stanu i shaderów. Elementy potoku, z którymi zetkniesz się w ramach tego ćwiczenia, zostały przedstawione na Rys. 1 - ograniczają się one do stadiów Direct3D 9. Taki początek przygody z Direct3D 11 wydawać może się skromny, ale są to bardzo ważne podstawy, na których będziemy bazować, wprowadzając kolejne pojęcia na następnych ćwiczeniach.

2 Zadania

2.1 Zadanie 1 - kluczowe procedury

Przyjrzyj się ciałom następujących metod klasy `RenderWidget`:

- `dxInitialize`
- `dxCreateResourcesAndStateObjectsAndShaders`
- `dxRender`

2.1.1 Inicjalizacja

`dxInitialize` tworzy urządzenie, kontekst urządzenia, a także *swap chain* w oparciu o:

- typ sterownika tworzonego urządzenia
- dodatkowe flagi dla urządzenia (np. debugging, patrz niżej)
- strukturę opisującą *swap chain*
- tablicę z akceptowanymi poziomami funkcjonalności (ang. *feature levels*)

Opis pozostałych parametrów nie jest kluczowy do zrozumienia roli, jaką pełni `dxInitialize`.

Jeżeli parametr `debug` ustawiony jest na `true`, urządzenie tworzone jest w taki sposób (flaga), że implementuje również interfejs `ID3D11Debug`, co pozwala na prowadzenie różnego rodzaju debuggingu, a także powoduje wyświetlanie błędów/ostrzeżeń API oraz wykrywanie wycieków pamięci.

2.1.2 Tworzenie zasobów, obiektów stanu i shaderów

`dxCreateResourcesAndStateObjectsAndShaders` tworzy zasoby, obiekty stanu i shaderów. W trakcie tego ćwiczenia interesują nas:

- zasoby dla IA (bufory wierzchołków i indeksów)
- obiekty shaderów dla VS (Vertex Shader Object) i PS (Pixel Shader Object)
- obiekty stanu dla RS (Rasterizer State Object)

2.1.3 Rendering

`dxRender` konfiguruje poszczególne stadia potoku według wymogów obecnej klatki, wywołuje żądania rysowania geometrii (and. *Draw Call*) i prezentuje wynikową zawartość bufora ramki na oknie. Przejdź stąd teraz do ciał metod konfigurujących potok (ich nazwy mówią same za siebie):

- `dxConfigureInputAssemblerStage`
- `dxConfigureVertexShaderStage`
- `dxConfigureRasterizerStage`
- `dxConfigurePixelShaderStage`
- `dxConfigureOutputMergerStage`

Zauważ, że metody konfigurujące w ogromnej większości po prostu *ustawiają* dla danych stadiów przetwarzania obiekty (stanów, shaderów) i zasoby utworzone w `dxCreateResourcesAndStateObjectsAndShaders`; ten podział zadań został zilustrowany na Rys. 2.

2.2 Zadanie 2 - wycieki pamięci

DirectX jest oparty o technologię COM: większość obiektów wykorzystywanych w aplikacji implementuje interfejs `IUnknown`, co nakłada na nas obowiązek zwalniania referencji do nich, kiedy aplikacja przestanie ich używać. Jeżeli tego nie zrobimy, na obiekcie nie zostanie wywołany destruktor.

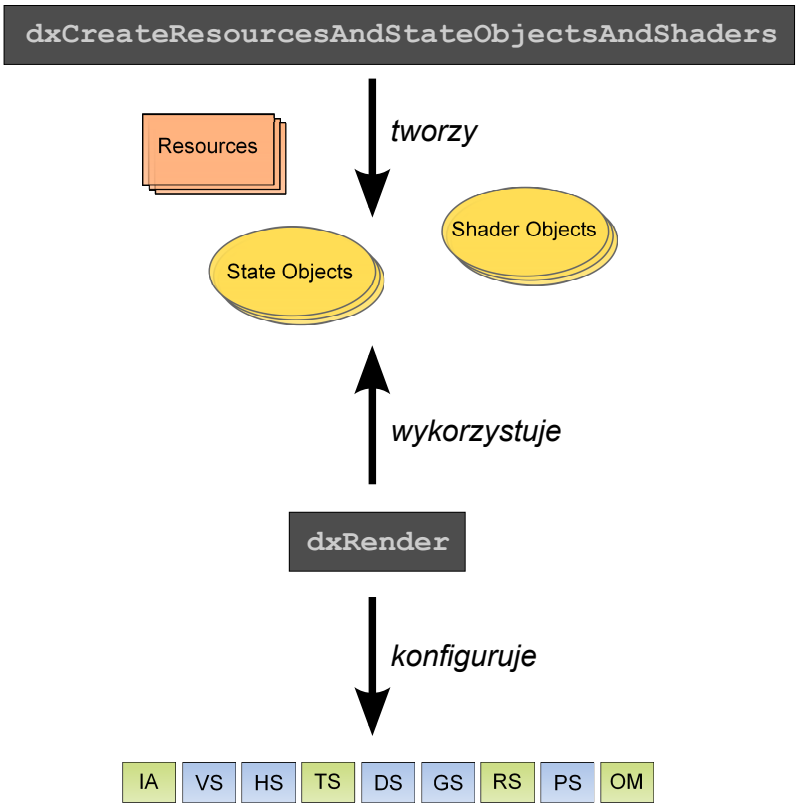
Wróć do `dxCreateResourcesAndStateObjectsAndShaders`. Zakomentuj wybrane linie typu: `m_vToBeReleased.push_back(...)`, uruchom i zamknij aplikację. Teraz spójrz na zakładkę *Output* w *Visual Studio*.

Zajrzyj jeszcze do `dxReleaseCOMRefs`, po czym przywróć zmodyfikowany kod do wyjściowej postaci.

2.3 Zadanie 3 - bufor wierzchołków

Znajdź w `dxCreateResourcesAndStateObjectsAndShaders` wywołanie metody `dxCreateVertexBuffers` i przejdź do niej. Wypełnij `vertices` wierzchołkami, z których powstanie prostokąt na płaszczyźnie XY sięgający od $(-0.5, -0.5)$ do $(0.5, 0.5)$, w odległości (Z) 0.5. [1]

[1] Nie będziemy korzystali z macierzy widoku i rzutowania, więc współrzędne wierzchołków musisz podać od razu w *clip space* (a wręcz w znormalizowanych współrzędnych urządzenia - NDC).



Rysunek 2: Podział zadań konfiguracji potoku przetwarzania w przykładowej aplikacji

Przebuduj i uruchom aplikację - powinieneś zobaczyć szary prostokąt. Zajrzyj jeszcze do pliku `pos.fx` - jak widzisz, rola shaderów sprowadza się w tym przypadku do przesłania wejścia na wyjście. Nie pojawiają się też żadne specyficzne elementy składniowe związane z D3D10/11, więc wszystko powinno być zrozumiałe.

2.4 Zadanie 4 - bufor indeksów

Zmodyfikuj kod, który napisałeś w poprzednim zadaniu tak, by wyeliminować powtarzające się wierzchołki. Uruchom aplikację - powinieneś zobaczyć trójkąt, dlaczego? Teraz przejdź do `dxCreateIndexBuffers` i wypełnij `indices` indeksami wierzchołków w taki sposób, by tworzyły prostokąt z zadania trzeciego.

2.5 Zadanie 5 - rasteryzacja

Przejdź do `dxCreateResourcesAndStateObjectsAndShaders`, a stamtąd do `dxCreateRasterizerStates`. Na podstawie opisu zawartego w strukturze `D3D11_RASTERIZER_DESC` tworzony tam jest obiekt stanu rasteryzatora (*Rasterizer State Object*), który w pełni konfiguruje to stadium potoku. Skupimy się tylko na wybranych opcjach konfiguracji:

- `FillMode` - w jaki sposób wyświetlać geometrię
- `FrontCounterClockwise` - które trójkąty uznawać za skierowane przodem
- `CullMode` - których (skier. przodem/tyłem) trójkątów *nie* rysować

Najpierw zmień `FillMode` na `D3D11_FILL_WIREFRAME` i zobacz efekty. Przywróć poprzednie ustawienie i przejdź do dwóch pozostałych opcji: doprowadź do tego, żeby dla twojej geometrii wyświetlane były tylko trójkąty skierowane przodem. [2]

[2] Jeżeli ukryjesz trójkąty skierowane tyłem, błędna konfiguracja spowoduje zniknięcie geometrii.