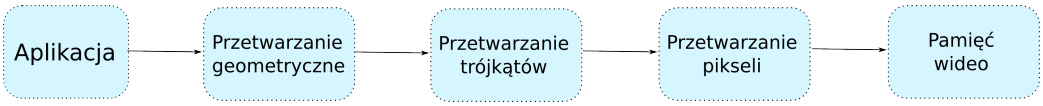
Instrukcja	Architektura procesorów graficznych
1	**Temat**: Vertex Shader Przygotował: mgr inż. Tomasz Michno

1 Wstęp

1.1 Czym jest shader

Shader jest programem (zazwyczaj krótkim), wykonywanym przez kartę graficzną. Nie jest to jednak typowy program (taki jak na przykład programy w języku Pascal, czy C), ponieważ jego zadaniem jest sterowanie częścią potoku graficznego/renderowania (graphics/rendering pipeline).
Potok graficzny - najprościej mówiąc, potok, który przetwarza obraz 3D na obraz 2D. W uproszczeniu można go przedstawić następująco:



Więcej o potoku renderowania i shaderach: `http://developer.nvidia.com/node/76`

Wyróżniamy następujące shadery:

- **Vertex shader** - służy do wykonania przekształceń na wierzchołku (uruchamiany raz dla każdego wierzchołka), możliwy jest dostęp do współrzędnych wierzchołka, jego koloru i współrzędnych tekstur. Nie jest możliwe tworzenie nowych wierzchołków.
- **Geometry shader** - służy do dodawania lub usuwania wierzchołków z siatki wierzchołków; dostępny od DirextX 10 i OpenGL 3.1
- **Pixel/Fragment Shader** - służy do wyliczania koloru pikseli, w DirectX stosowana jest nazwa Pixel Shader, w OpenGL Fragment Shader; pozwala zastosować operacje związane głownie z kolorem obiektów - cieniowanie, oświetlanie, ale również np. mapowanie nierówności

Obecnie najczęściej shadery pisane są w jednym z trzech języków: GLSL, HLSL lub Cg (na którym się skupimy).

1.2 Podstawy języka Cg

Język Cg jest językiem wysokiego poziomu stworzonym przez nVidię przy współpracy Microsoftu (jest kompatybilny z HLSL). Został w dużej mierze oparty o język C - posiada tą samą składnię, instrukcje sterujące i typy danych (zostały dodane też typy przydatne dla programowania shaderów - grafiki).

1.2.1 Podstawowe typy:

- float - 32-bitowa liczba zmiennoprzecinkowa
- half - 16-bitowa liczba zmiennoprzecinkowa
- int - 32-bitowa liczba całkowita
- fixed - 12-bitowa liczba typu fixed-point
- bool - wartość boolowska (true, false)
- sampler* - używane do reprezentacji tekstur
- tablice-wektory na bazie podstawowych typów - w postaci np. float4
- macierze na bazie podstawowych typów - w postaci np. float4x4
- uniform - odpowiednik const - informuje, że zmienna nie będzie zmieniana w kodzie shadera
- struktury - struct - używane podobnie jak w C/C++

1.2.2 Przykład 1 - ustawienie koloru obiektu

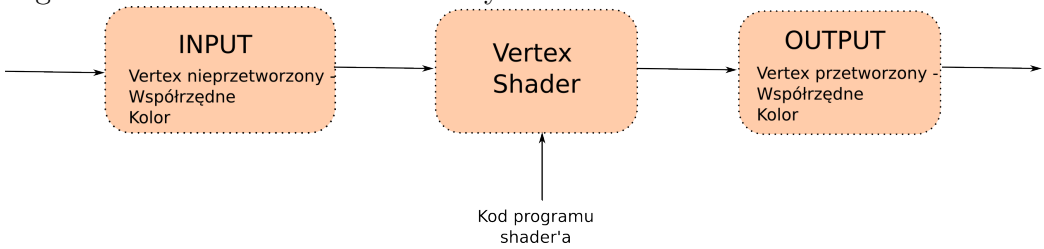
Pierwszy przykład Vertex Shadera będzie polegał na zmianie koloru obiektu w kodzie shadera.

```
1  struct Vertex
2  {
3    float4 position : POSITION;
4    float4 color    : COLOR0;
5  };
6
7  Vertex main(Vertex IN,  uniform float4x4 ModelViewProj)
8  {
9    Vertex OUT;
10   IN.color.x=0.0f;
```

```
11    IN.color.y=1.0f;
12    IN.color.z=1.0f;
13
14    OUT.position = mul(ModelViewProj, IN.position);
15    OUT.color.xyz = IN.color.xyz;
16
17    return OUT;
18  }
```

Listing 1: Kod Shadera

Powyżej znajduje się kod jednego z najprostszych shaderów (Listing 1). Jego działanie można zobrazować rysunkiem:



Jak widać, kod programu otrzymuje na wejściu Vertex (jego pozycję i kolor) i zwraca również Vertex.
Dlatego na początku kodu shadera należy zdefiniować strukturę, która będzie reprezentowała wierzchołek (w naszym kodzie struktura o nazwie Vertex, linie 1-5). Tworzenie struktury odbywa się w identyczny sposób, jak w języku C. Pozycja i kolor powinny być typu float4 (4-elementowy wektor liczb float). Jedyną różnicą, którą można zauważyć są etykiety `POSITION` i `COLOR0`, które informują kompilator, która ze zmiennych opisuje pozycję Vertex'a, a która kolor. Ich użycie jest niezbędne dla kompilatora, ponieważ w inny sposób nie ma możliwości dowiedzenia się, do jakich rejestrów powinny zostać przypisane konkretne zmienne (podobnie jak w C, zmienne mogą mieć dowolne nazwy). Oprócz tych dwóch zmiennych, w strukturze można umieścić jeszcze inne elementy, które będą służyły do przekazywania danych/instrukcji z programu OpenGL.
Następnym krokiem jest utworzenie funkcji main(), która powinna zwracać przetworzony vertex (w naszym przykładzie jest to struktura o nazwie Vertex). Parametrami zazwyczaj są: vertex (Vertex IN) i macierz projekcji (uniform float4x4 ModelViewProj). W kodzie funkcji main powinien znaleźć się kod modyfikujący vertex'a. W naszym przykładzie:
w linii nr 9 tworzymy zmienną typu Vertex o nazwie OUT, która później zostanie zwrócona jako wynik funkcji - vertex wyjściowy
w liniach 10-12 modyfikujemy kolor vertex'a (dostęp do pól struktur jest identyczny jak w C) - pole x odpowiada za czerwony, y za zielony i z za niebieski

w linii 14 ustawiamy współrzędne vertex'a wyjściowego, która musi zostać zawsze pomnożona przez macierz widoku
w linii 15 kopiujemy kolor do vertex'a wyjściowego - jak widać możliwe jest skopiowanie tylko części pól (np. użycie color.xz skopiowałoby tylko pola x i z).

```
1  #include <GL/gl.h>
2  #include <GL/glu.h>
3  #include <GL/glut.h>
4
5  #include <Cg/cg.h>
6  #include <Cg/cgGL.h>
7
8  #include <stdlib.h>
9  #include <stdio.h>
10
11 CGcontext cgContext;
12 CGprogram cgProgram;
13 CGprofile cgVertexProfile;
14 CGparameter modelViewMatrix, position, color;
15
16 void init(){
17
18   glEnable(GL_DEPTH_TEST);
19
20   cgContext = cgCreateContext();
21   if(cgContext==0){
22     printf("Nie mozna utworzyc kontekstu Cg!");
23     exit(1);
24   }
25
26   cgVertexProfile = cgGLGetLatestProfile(CG_GL_VERTEX);
27   if(cgVertexProfile == CG_PROFILE_UNKNOWN){
28     printf("Nieznany profil!");
29     exit(1);
30   }
31
32   cgGLSetOptimalOptions(cgVertexProfile);
33
34   cgProgram = cgCreateProgramFromFile(cgContext, CG_SOURCE, "
       shader.cg", cgVertexProfile, "main", 0);
35
36   if (cgProgram == 0)
37   {
38     CGerror Error = cgGetError();
39
```

```
40      fprintf(stderr,"%s \n",cgGetErrorString(Error));
41      exit(-1);
42    }
43
44    cgGLLoadProgram(cgProgram);
45    position      = cgGetNamedParameter(cgProgram, "IN.position");
46    color         = cgGetNamedParameter(cgProgram, "IN.color");
47    modelViewMatrix    = cgGetNamedParameter(cgProgram, "
          ModelViewProj");
48  }
49
50  void display() {
51    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
52    glLoadIdentity();
53
54    gluLookAt(50.0f, 250.0f, -845.0f, 0.0f, 0.0f, 0.0f, 0, 1, 0);
55    cgGLSetStateMatrixParameter(modelViewMatrix,
          CG_GL_MODELVIEW_PROJECTION_MATRIX, CG_GL_MATRIX_IDENTITY);
56    cgGLEnableProfile(cgVertexProfile);
57    cgGLBindProgram(cgProgram);
58
59    glColor3f(1, 1, 0);
60    glutSolidCube(250);
61    cgGLDisableProfile(cgVertexProfile);
62
63    glutSwapBuffers();
64  }
65
66  /* Pozostala czesc programu standardowa... */
```

Listing 2: Kod programu OpenGL (fragmenty)

Kod programu OpenGL używającego shaderów tak na prawdę wiele się nie różni od zwykłego programu. Musimy dołączyć nagłówki języka Cg (linie 5 i 6). W liniach 11-14 tworzone są zmienne, które są niezbędne do użycia programu shadera:
cgContext - kontekst urządzenia
cgProgram - przechowuje skompilowany program Cg
cgVertexProfile - profil VertexShadera - używany w celu jak najlepszej optymalizacji kodu
modelViewMatrix, position, color - parametry używane w programie shadera

Następnie w liniach 20-32 tworzony jest kontekst i pobierany jest profil (więcej informacji w tutorialu do języka Cg od nVidii).
Linia 34 jest bardzo ważna, ponieważ odpowiada za kompilację programu shadera. Jako parametry funkcji cgCreateProgramFromFile podajemy kontekst, flagę CG_SOURCE, ścieżkę do kodu źródłowego programu Cg, profil

VertexShadera i nazwę głównej funkcji programu. Jeśli coś się nie powiodło (funkcja zwróciła 0), odczytujemy informacje o błędzie i je wyświetlamy.
W linii 44 ładujemy program do pamięci oraz w liniach 45-47 łączymy parametry funkcji main programu Cg ze zmiennymi z programie OpenGL.
W celu użycia shadera, w funkcji wyświetlającej ustawiamy macierz widoku (linia 55), włączamy profil VertexShadera (56), bindujemy (wybieramy) program shadera (57) i rysujemy, to co mamy narysować. Później należy wyłączyć profil VertexShadera (linia 61).

1.2.3 Przykład 2 - przekazywanie parametrów z kodu głównego do kodu shader'a

Przykład pokazuje, w jaki sposób przekazywać parametry do programu Cg. Program będzie spłaszczał sferę, poprzez ustawienie tej samej wartości współrzędnej y dla vertexów znajdujących się wyżej niż ustalona granica.

```
1  struct InputVertex
2  {
3    float4 position : POSITION;
4    float4 color    : COLOR0;
5    float  yThreshold;
6  };
7
8  struct OutputVertex
9  {
10   float4 position : POSITION;
11   float4 color    : COLOR0;
12 };
13
14 OutputVertex main(InputVertex IN, uniform float4x4 ModelViewProj)
15 {
16   OutputVertex OUT;
17
18   if(IN.position.y>IN.yThreshold){
19       IN.position.y=IN.yThreshold;
20   }
21
22   OUT.position = mul(ModelViewProj, IN.position);
23
24   OUT.color.xyz = IN.color.xyz;
25
26   return OUT;
27 }
```

Listing 3: Kod programu shader'a

Sam kod programu Cg uległ jedynie niewielkim zmianom. Został wprowadzony podział na strukturę dla vertexów wejściowych (InputVertex) i wyjściowych (OutputVertex). W strukturze InputVertex pojawił się nowy element yThreshold (linia nr 5), przez który będziemy przekazywali graniczną wartość y, powyżej której sfera ma być spłaszczona.
W funkcji main, poza zmianą struktur vertexów, została dodana instrukcja if (linie 18-20). Sprawdza ona, czy współrzędna y vertexa jest większa od zmiennej yThreshold, jeśli tak, to ustawia ją na wartość yThreshold (dzięki czemu sfera zostaje spłaszczona).

W celu ustawiania parametru yThreshold, należy w programie głównym (dla CPU):

- stworzyć dodatkową zmienną typu CGparameter (w przykładzie jest to yThresholdVertexShader)
- połączyć ją z parametrem głównej funkcji programu shader'a; Po modyfikacji może to wyglądać tak:

```
position=cgGetNamedParameter(cgProgram, "IN.position");
color=cgGetNamedParameter(cgProgram, "IN.color");
yThresholdVertexShader=cgGetNamedParameter(cgProgram, "IN.
    yThreshold");
modelViewMatrix=cgGetNamedParameter(cgProgram, "
    ModelViewProj");
```

- w funkcji display, po zbindowaniu programu Cg należy ustawić ten parametr na jakąś wartość, np:
  `cgGLSetParameter1f(yThresholdVertexShader, 10);`
  (w przykładzie zamiast stałej liczby stosujemy zmienną o nazwie yThreshold, która zmienia swoją wartość w czasie działania programu)

Do ustawiania parametrów służy rodzina funkcji cgGLSetParameter, najpopularniejsze z nich to:

```
void cgGLSetParameter1{fd}( CGparameter param,
                            TYPE x );

void cgGLSetParameter2{fd}( CGparameter param,
                            TYPE x,
                            TYPE y );

void cgGLSetParameter3{fd}( CGparameter param,
```

```
                                TYPE x,
                                TYPE y,
                                TYPE z );

void cgGLSetParameter4{fd}( CGparameter param,
                                TYPE x,
                                TYPE y,
                                TYPE z,
                                TYPE w );
```

gdzie:

- liczba po cgGLSetParameter oznacza, ile pól posiada ten parametr (np. dla float3 dajemy 3, dla zwykłego float 1)
- litera f lub d w nawiasie klamrowym służy do wyboru odpowiedniego typu (wybieramy tylko jedną z nich) - f oznacza float, d double.

1.3 Przydatne linki

The Cg Tutorial - `http://developer.nvidia.com/node/76` - po prawej stronie znajdują się linki do kolejnych rozdziałów
Artykuł w NeHe Productions - `http://nehe.gamedev.net/tutorial/cg_vertex_shader/25002/`
(polskie tłumaczenie: `http://aklimx.sppieniezno.pl/nehepl/display.php?id=47`)