31 maja 2012

GLSL - Jednostki cieniujące

dr inż. Agnieszka Szczęsna

1. Cel laboratorium

Celem laboratorium jest zapoznanie studentów z językiem GLSL umożliwiającym zaimplementowanie wysokopoziomowej jednostki cieniującej wierzchołków i fragmentów wykonującej się na karcie graficznej.

2. Wprowadzenie

OpenGL umożliwia zamianę standardowego potoku renderingu na dowolny stworzony przez użytkownika oraz dostosowany do jego potrzeb. Zamiana potoku renderingu polega na stworzeniu specjalnych programów zwanych jednostkami cieniującymi (*shaders*), które pozwalają zaprogramować potok renderingu zgodnie z życzeniem programisty. Jednostki te dzielą się na dwa rodzaje – jednostki cieniujące wierzchołków odpowiadają za obróbkę wierzchołków oraz jednostki fragmentów (pikseli) odpowiadają za obróbkę fragmentów. Tradycyjnie minimalny zestaw przekształceń w jednostce cieniowania wierzchołków to transformacja wejściowej pozycji macierzą model-widok oraz macierzą rzutowania. Wierzchołki są przekształcane na fragmenty na etapie rasteryzacji. Jednostki cieniujące fragmentów otrzymują na wejściu te same współrzędne tekstur, mgły i kolorów, jakie w klasycznym potoku otrzymuje faza obróbki fragmentów. Analogicznie, na wyjściu jednostki fragmentów powinien pojawić się pojedynczy kolor. Droga między wejściem, a wyjściem jest uzależniona wyłącznie od programisty. Jednostki cieniujące fragmentów mają możliwość pobierania kolorów z tekstury.

Stosując jednostki cieniujące wysokopoziomowe za pomocą języka GLSL musimy użyć dwóch typów obiektów – obiektów shaderów i obiektów programów. Obiekty shaderów tworzy się z użyciem funkcji **`glCreateShaderObject`**, przekazując jako argument typ shadera (**`GL_VERTEX_SHADER, GL_FRAGMENT_SHADER`**). Usuwanie obiektu shadera odbywa się za pomocą funkcji **`glDeleteObject`**. Celem obiektu shadera jest przyjęcie kodu shadera i jego kompilacja. Kod shadera może być zaszyty w program w postaci łańcucha znaków lub może być odczytany z pliku na dysku. Kod shadera należy załadować do obiektu shadera za pomocą funkcji **`glShaderSource`**. Kompilacja programu shadera odbywa się za pomocą funkcji **`glCompileShader`**.

Programy są kontenerami dla shaderów. Do programu można przypisać shader wierzchołków i shader fragmentów. Można dołączać do programu kilka shaderów jednego typu, co pozwala na wykorzystanie współdzielonych bibliotek kodu. Przydatne funkcje: **`glCreateProgramObject`** – tworzenie programu, **`glDeleteObject`** – usuwanie programu, **`glAttachObject`** – dołączanie shadera

do programu, **glDetachObject** – odłączanie shadera od programu, **glLinkProgram** – konsolidacja programu, **glValidateProgram** – walidacja programu uwzględniająca bieżący stan renderingu.

3. Zadanie

W ramach zadania należy w środowisku VisualC na podstawie pliku **cw5_shader.cpp** oraz plików jednostek cieniujących **perVertexLightingSimple.vs**, **passThrough.fs** wykonać następujące elementy:

a. Zapoznać się ze sposobem dołączania i kompilowania jednostek cieniujących do programu:
   i. funkcja **extensionSetup** – ustawianie wskaźników potrzebnych funkcji;
   ii. funkcja **setShaders** – wczytywanie, kompilacja i uruchamianie jednostek cieniujących.

b. Zaimplementować oświetlenie sceny za pomocą oświetlanei globalnego i zdefiniowanego źródła światła **GL_LIGHT0** w jednostce cieniującej wierzchołków, należy wykorzystać model Phonga uwzględniający:

$$V_X = L_{globAmb,X} * V_{amb,X} + L_{0,amb,X} * V_{amb,X} + \max\{l_0 \cdot n, 0\} * L_{0,diff,X} * V_{diff,X} + (\max\{s_0 \cdot n, 0\})^f * L_{0,spec,X} * V_{spec,X})$$

Gdzie:

V_x=>R,G,B – wynikowy kolor wierzchołka

L – właściwości światła (w shaderze wierzchołków na przykład dla światła globalnego mamy **gl_LightModel.ambient,** dla światła 0: **gl_LightSource[0].ambient, gl_LightSource[0].diffuse, gl_LightSource[0].specular**)

V – właściwości materiału (w shaderze wierzchołków parametry: **gl_FrontMaterial.ambient, gl_FrontMaterial.diffuse, gl_FrontMaterial.specular**)

l_0 – wektor kierunku źródła światła 0

n – wektor normalny

s_0 – wektor połówkowy między wektorem patrzenia (można przyjąć [0, 0, 1, 0]), a wektorem światła (w shaderze wierzchołków **gl_LightSource[0].halfVector**)

f – eksponenta jasności (w shaderze wierzchołków parametr ***gl_FrontMaterial.shininess***)

Należy skorzystać z następujących elementów języka GLSL: **`gl_Vertex`**, **`gl_NormalMatrix`**, **`gl_LightSource[0]`**, **`gl_LightModel`**, **`gl_FrontMaterial`**.

Wynikiem jednostki cieniującej wierzchołków powinno być: kolor (**`gl_FrontColor`**) i pozycja wierzchołka (**`gl_Position`**).

Potrzebne operacje w GLSL: **`normalize`**, **`dot`**, **`max`**, **`pow`**, **`ftransform`**.

c. Zrealizować efekt uproszczonego *normal mappingu*, gdzie wektory normalne powinny zostać zmienione w jednostce cieniującej wierzchołków w celu uzyskania efektu przedstawionego na rysunku poniżej bez zmiany geometrii.