Programowanie Grafiki Komputerowej

OpenGL Shading Language cz. 3 - oświetlenie i teksturowanie

Grzegorz Pawiński

Katedra Informatyki, Politechnika Świętokrzyska w Kielcach



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



Politechnika Świętokrzyska

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY



Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

" Program Rozwojowy Potencjału Dydaktycznego Politechniki Świętokrzyskiej w Kielcach: *kształcenie w nowoczesnych obszarach techniki* "

Program Operacyjny Kapitał Ludzki, Priorytet IV Szkolnictwo wyższe i nauka, Działanie 4.1. Wzmocnienie i rozwój potencjału dydaktycznego uczelni oraz zwiększenie liczby absolwentów kierunków o kluczowym znaczeniu dla gospodarki opartej na wiedzy, Poddziałanie 4.1.1. Wzmocnienie potencjału dydaktycznego uczelni.

Umowa UDA–POKL.04.01.01-00395/09-00

Spis treści

1. Oświetlenie
   - Oświetlenie per vertex
   - Oświetlenie obliczane dla pikseli
   - Rodzaje oświetlenia
2. Teksturowanie
   - Obliczanie koloru tekstury
   - Przykłady
3. Bibliografia

Uwagi

- Animacje są uruchamine, gdy plik PDF jest wyświetlany na pełnym ekranie.
- Animacja jest uruchamiana dopiero po naciśnięciu spacji (program Adobe Reader).
- Klatki animacji wyświetlane są co 2 sekundy.
- Symulacje są uruchamiane po kliknięciu w ikonę z odpowiednim systemem operacyjnym. Przeglądarka wyświetli okno z ostrzeżeniem, należy go zaakceptować. W systemie Linux należy dodatkowo kliknąć dwukrotnie w nazwę programu.

Wymagania do uruchomienia symulacji

- Do uruchamiania symulacji z pliku PDF, wymagany jest program Adobe Reader w wersji 5.0 lub nowszej.
- W systemie Windows, należy umieścić bibliotekę *glut32.dll* w katalogu systemowym (zwykle *C:\Windows*), a w jego podkatalogu *System32* bibliotekę *glew32.dll*. Biblioteki można znaleźć TUTAJ.
- W systemie Linux, biblioteki należy umieścić w katalogu */usr/lib*. Biblioteki można znaleźć TUTAJ.
- Symulacje programów zostały napisane dla sprzętu komputerowego obsługującego OpenGL w wersji 2.0 lub nowszej.

Obliczanie oświetlenia

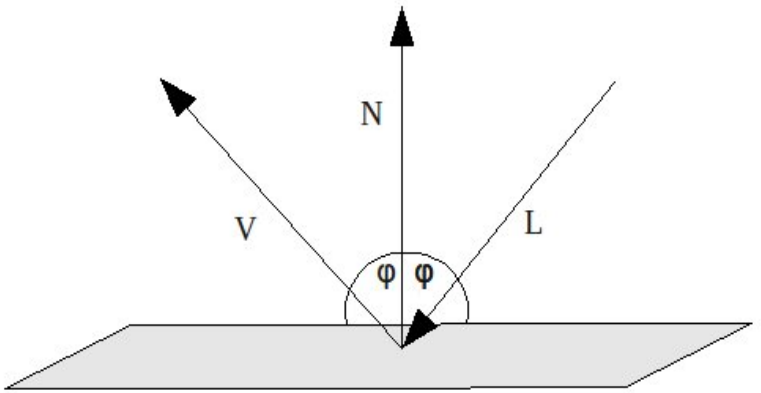
Oświetlenie w GLSL może być obliczane na dwa sposoby:

- ***dla każdego wierzchołka*** (ang. *per vertex*) - realizuje cieniowanie Gourauda (standardowy model w OpenGL). W *vertex shaderze*, dla każdego wierzchołka obliczane jest natężenie oświetlenia. Interpolowane natężenie przekazywane jest do *pixel shadera*, który oblicza prawidłowe cieniowanie koloru.
- ***dla każdego piksela*** (ang. *per pixel*) - realizuje cieniowanie Phonga. *Vertex shader* oblicza i przekazuje do *pixel shadera* tylko wektor normalny. Interpolowana wartość jest następnie wykorzystywana do obliczenia natężenia oświetlenia i ostatecznie koloru piksela.

Natężenie oświetlenia

Natężenie oświetlenia jest obliczane na podstawie kosinusa kąta pomiędzy wektorem kierunku oświetlenia i wektora normalnego do powierzchni. Do obliczenia natężenia oświetlenia potrzebny jest więc:

- ***wektor padania światła*** (L) - dla światła kierunkowego, przekazywany z aplikacji OpenGL, natomiast dla światła punktowego należy go wyliczyć.
- ***wektor padania światła*** (N) - wektor normalny dla danego wierzchołka/piksela



Uwagi

Jeśli wektory są znormalizowane i znajdują się w tym samym układzie współrzędnych (lokalnych - obiektu lub globalnych - świata). Natężenie jest liczone, jako iloczyn kartezjański tych wektorów.

Dostęp do parametrów oświetlenia

Predefiniowane zmienne oświetlenia

```
STRUCT gl_LightSourceParameters {
  VEC4 ambient; // światło otoczenia
  VEC4 diffuse; // światło rozproszenia
  VEC4 specular; // światło odbite
  VEC4 position; // pozycja światła
  VEC4 halfVector; // wektor połówkowy
  VEC3 spotDirection; // kierunek reflektora
  FLOAT spotExponent; // rozkład natężenia reflektora
  FLOAT spotCutoff; // kąt reflektora
  FLOAT spotCosCutoff;
  FLOAT constantAttenuation; // stałe zanikanie
  FLOAT linearAttenuation; // liniowe zanikanie
  FLOAT quadraticAttenuation; // kwadratowe zanikanie
};
UNIFORM gl_LightSourceParameters gl_LightSource[gl_MaxLights];
```

Dostęp do wektora normalnego z vertex shadera

Vertex shader ma dostęp do wektora normalnego poprzez zmienną predefiniowaną *gl_Normal*, która jest ustawiana przez funkcję OpenGL'a *glNormal()*.

Dostęp do wektora normalnego z vertex shadera

ATTRIBUTE VEC4 *gl_Normal*

Specyfikacja OpenGL podaje, że kierunek światła jest automatycznie konwertowany na współrzędne świata. Przed obliczeniem natężenia oświetlenia, należy tak samo przekształcić wektor normalny. W przeciwnym wypadku liczenia kąta pomiędzy wektorami nie ma sensu.

Transformacja wektora normalnego

Transformacja wektora normalnego do współrzędnych świata polega na pomnożeniu jego współrzędnych przez predefiniowaną macierz:

podmacierz z widoku modelu

UNIFORM MAT3 *gl_NormalMatrix*

Transformacja wektora normalnego do współrzędnych świata

```
vec3 normal;
void main() {
  normal = gl_NormalMatrix * gl_Normal;
}
```

Oświetlenie per vertex - vertex shader

Kod źródłowy vertex shadera

```
varying float intensity; // zmienna przekazywana do pixel shadera

void main()
{
  // skonwertowanie wektora normalnego do wsplrzednych swiata
  vec3 normal = gl_NormalMatrix * gl_Normal;

  // pobranie wektora kierunkowego swiatla
  vec3 lightDir = vec3(gl_LightSource[0].position);

  //obliczenie natezenia swiatla - iloczyn kartezjanski wektorow
  intensity = dot(lightDir,normal);

  gl_FrontColor = gl_Color;
  gl_Position = ftransform();
}
```

Uwagi

Przykład pokazuje użycie światła kierunkowego i zakłada, że wektory zostały znormalizowane w aplikacji OpenGL.

Oświetlenie per vertex - pixel shader

Kod źródłowy pixel shadera

```
// natezenie swiatla wyliczone w vertex shaderze
varying float intensity;

void main()
{
  // skorzystanie ze zmiennej lokalnej, poniewaz gl_Color
  // nie mozna modyfikowac
  vec4 color;

  // pomnozenie skladowych koloru przez natezenie swiatla
  color.rgb = gl_Color.rgb * intensity;

  //ustawienie koloru piksela
  gl_FragColor = color;
}
```

Pixel shader pobiera od *vertex shadera* natężenie oświetlenie i przemnaża je przez składowe koloru (RGB). W rezultacie wynikowy kolor jest jaśniejszy, gdy natężenie jest większe.

Przykład

Przykładowy program pokazuje oświetlenie obiektu 3D per vertex z wykorzystaniem shaderów.

Przykładowy program





Oświetlenie per pixel - vertex shader

Kod źródłowy vertex shadera

```
varying vec3 normal; // zmienna przekazywana do pixel shadera

void main()
{
  // skonwertowanie wektora normalnego do wsplrzednych swiata
  normal = gl_NormalMatrix * gl_Normal;

  gl_FrontColor = gl_Color;
  gl_Position = ftransform();
}
```

Vertex shader konwertuje tylko wektor normalny do współrzędnych świata. Wyliczenie natężenia oświetlenia wykonywane jest w *pixel shaderze*. Interpolowany wektor normalny posiada prawidłowy kierunek, ale długość inną niż 1, dlatego musi zostać znormalizowany.

Oświetlenie per pixel - pixel shader

Kod źródłowy pixel shadera

```
// wektor normalny wyliczony w vertex shaderze
varying vec3 normal;

void main()
{
  // skorzystanie ze zmiennej lokalnej, poniewaz gl_Color
  // nie mozna modyfikowac
  vec4 color;

  // pobranie wektora kierunkowego swiatla
  vec3 lightDir = vec3(gl_LightSource[0].position);

  //obliczenie natezenia swiatla - iloczyn kartezjanski wektorow
  float intensity = dot(lightDir,normalize(normal));

  // pomnozenie skladowych koloru przez natezenie swiatla
  color.rgb = gl_Color.rgb * intensity;
  gl_FragColor = color;

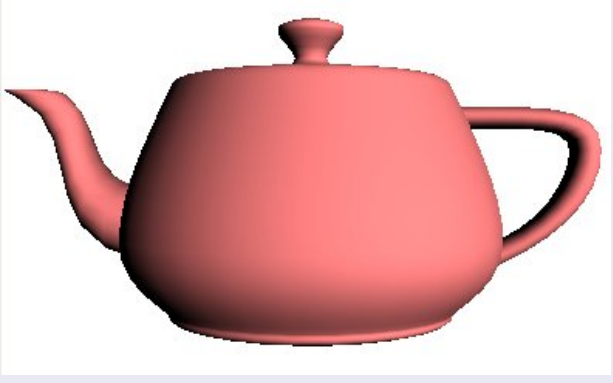
}
```

Przykład

Przykładowy program pokazuje oświetlenie obiektu 3D per pixel z wykorzystaniem shaderów.

Przykładowy program





Rodzaje oświetlenia

W OpenGl występują różne typy oświetlenia:

- ***światło kierunkowe*** (ang. *directional light*) - światło otoczenia (ang. *ambient light*), światło rozproszenia (ang. *diffuse light*) i światło odbicia (ang. *specular light*).
- ***światło punktowe*** (ang. *point light*).
- ***reflektor*** (ang. *spot light*).

Bez względu na typ oświetlenia, kolor piksela jest uzależniony od właściwości materiału na jaki pada światło. Każdy materiał określa jaką ilość światła pochłania, a jaką odbija.

Dostęp do parametrów materiału

Predefiniowane zmienne modelu światła i materiału

```
STRUCT gl_LightModelParameters {
  VEC4 ambient; // globalne oświetlenie
};
UNIFORM gl_LightModelParameters gl_LightModel;

STRUCT gl_MaterialParameters {
  VEC4 emission; // stopień emisji światła przez obiekt
  VEC4 ambient; // stopień odbicia światła otoczenia
  VEC4 diffuse; // stopień rozproszenia światła
  VEC4 specular; // stopień odbicia światła odbitego
  FLOAT shininess; // stopień rozbłysku światła
};
UNIFORM gl_MaterialParameters gl_FrontMaterial;
UNIFORM gl_MaterialParameters gl_BackMaterial;
```

Światło rozproszone

Do obliczenia światła otoczenia zostaną użyte właściwości światła tj. położenie i natężenie, a także stopień rozproszenia materiału. Natężenie światła rozproszenia jest proporcjonalne do cosinusa kąta pomiędzy kierunkiem padania światła, a wektorem normalnym do powierzchni.

Wzór na natężenie światła rozproszenia

$I_d = L_d * M_d * cos(\theta)$

Oznaczenia

- L_d - światło rozproszenia.
- M_d - stopień rozproszenia światła.
- $cos(\theta)$ - kąt pomiędzy kierunkiem światła, a wektorem normalnym do powierzchni.

Światło rozproszone - vertex shader

Kod źródłowy vertex shadera

```
void main()
{
  // skonwertowanie wektora normalnego do wsplrzednych swiata
  vec3 normal = normalize(gl_NormalMatrix * gl_Normal);
  // normalizacja polozenia swiatla
  vec3 lightDir = normalize(vec3(gl_LightSource[0].position));
  //obliczenie natezenia swiatla - iloczyn kartezjanski wektorow
  float intensity = max(dot(normal, lightDir), 0.0);

  // obliczenie rozproszenia swiatla
  vec4 diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
  // kolor jest proporcjonalny do natezenia i rozproszenia swiatla
  gl_FrontColor = gl_Color.rgba * intensity * diffuse;
  gl_Position = ftransform();
}
```

Przed obliczeniem cosinusa kąta, należy znormalizować kierunek padania światła oraz wektor normalny do powierzchni..

Światło rozproszone - pixel shader

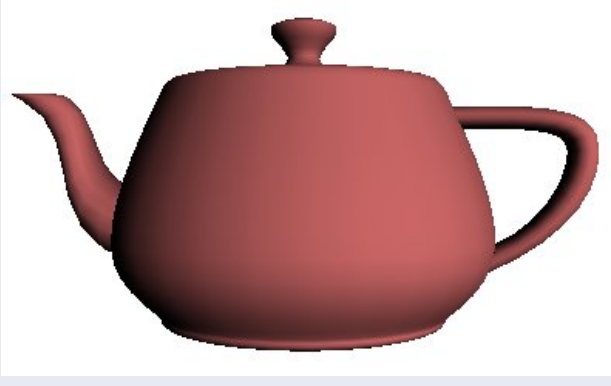
Kod źródłowy pixel shadera

```
void main()
{
  gl_FragColor = gl_Color;
}
```

Pixel shader otrzymuje interpolowaną wartość koloru i jedyne co musi zrobić, to ustawić ją dla piksela.

Przykładowy program





Światło otoczenia

Aby obliczyć natężenie światła otoczenia, należy wziąć pod uwagę globalne warunki otoczenia oraz warunki światła otoczenia dostępne w aplikacji OpenGL. Warunki należy dodać do natężenia światła rozproszenia (I_d).

Wzór na natężenie światła otoczenia

$$I_a = I_d + G_a * M_a + L_a * M_a$$

Oznaczenia

- I_d - natężenie światła rozproszonego.
- G_a - globalne światło otoczenia.
- M_a - stopień odbicia światła rozproszenia.
- L_a - światło otoczenia.

Światło otoczenia - vertex shader

Kod źródłowy vertex shadera

```
void main()
{
  // skonwertowanie wektora normalnego do wsplrzednych swiata
  vec3 normal = normalize(gl_NormalMatrix * gl_Normal);
  // normalizacja polozenia swiatla
  vec3 lightDir = normalize(vec3(gl_LightSource[0].position));
  //obliczenie natezenia swiatla - iloczyn kartezjanski wektorow
  float intensity = max(dot(normal, lightDir), 0.0);

  // obliczenie rozproszenia swiatla
  vec4 diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
  vec4 ambient = gl_FrontMaterial.ambient * gl_LightSource[0].ambient;
  vec4 globalAmbient = gl_LightModel.ambient * gl_FrontMaterial.ambient;

  // kolor jest proporcjonalny do natezenia i rozproszenia swiatla
  // oraz uzalezniony do swiatla otoczenia i globalnych warunkow otoczenia
gl_FrontColor = gl_Color.rgba * intensity * diffuse + globalAmbient + ambient;

  gl_Position = ftransform();
}
```

Światło otoczenia - pixel shader

Kod źródłowy pixel shadera

```
void main()
{
  gl_FragColor = gl_Color;
}
```

Pixel shader otrzymuje interpolowaną wartość koloru i jedyne co musi zrobić, to ustawić ją dla piksela.

Przykładowy program



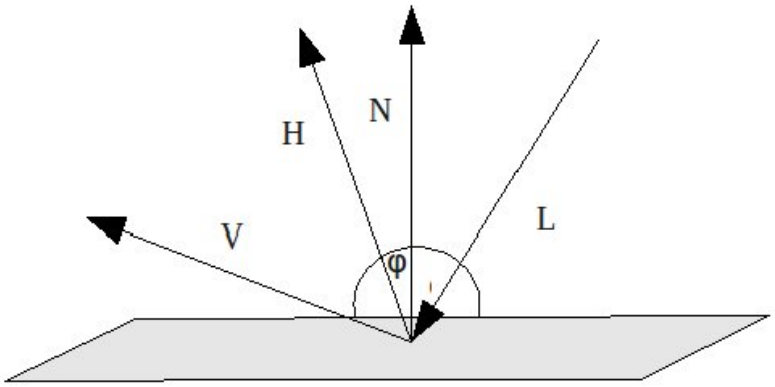


Model oświetlenia Blinna-Phonga

Model oświetlenia ***Blinna-Phonga*** jest urposzczoną metodą sumylacji odbicia zwierciadlanego, wykorzystującą ***wektor połówkowy*** (H), zamiast ***wektora odbicia światła*** (V).Metoda daje efekty prawie identyczne jak metoda Phonga, ale przy mniejszym narzucie obliczeniowym.

Wzór na wektor połówkowy

$$\vec{H} = \frac{\vec{L}+\vec{V}}{2}$$



Światło odbite

Światło odbite jest proporcjonalne do kąta pomiędzy wektorem połówkowym a wektorem normalnym, światła odbicia oraz właściwości materiału. Dodawane jest do natężenia światła otoczenia (I_a). Efekt odbicia zwierciadlanego zależy również od współczynnika rozbłysku materiału.

Wzór na natężenie światła odbite

$$I_s = I_a + (N \cdot H)^n * L_s * M_s$$

Oznaczenia

- I_a - natężenie światła otoczenia.
- N - wektor normalny do powierzchni.
- H - wektor połówkowy.
- n - współczynnik rozbłysku materiału.
- L_s - światło odbite.
- M_s - stopień odbicia światła odbitego.

Światło odbite - vertex shader

Kod źródłowy vertex shadera

```
void main()
{
  /**      kod jak, dla swiatla otoczenia      **/
  // zadeklarowanie wektora dla swiatla odbitego
  vec4 specular = specular = vec4(0.0);
  // obliczenie swiatla odbitego, gdy natezenie oswietlenia > 0
  if (intensity > 0.0) {

    // normalizacja wektora polowkowego i obliczenie kosiuna kata
    // pomiedzy wektorem polowkowym a normalnym
    float intensityHV = max(dot(normal, gl_LightSource[0].halfVector.xyz),0.0);
    // potega okresla stopien skupienia plamy rozblysku
    specular = gl_FrontMaterial.specular * gl_LightSource[0].specular *
        pow(intensityHV,gl_FrontMaterial.shininess);
  }
  // kolor jest proporcjonalny do natezenia i rozproszenia swiatla
  // oraz uzalezniony do swiatla otoczenia, globalnych warunkow otoczenia
  // i swiatla odbitego
  gl_FrontColor = gl_Color.rgba * intensity * diffuse +
      globalAmbient + ambient + specular;
  gl_Position = ftransform();
}
```

Światło odbite - pixel shader

Kod źródłowy pixel shadera

```
void main()
{
  gl_FragColor = gl_Color;
}
```

Pixel shader otrzymuje interpolowaną wartość koloru i jedyne co musi zrobić, to ustawić ją dla piksela.

Przykładowy program





Współrzędne tekstur

Przed przeprowadzeniem operacji na teksturach, należy uzyskać dostęp do ich współrzędnych. *Vertex shader* wykorzystuje predefiniowane zmienne, dla każdej jednostki teksturującej, do pobrania *współrzędnych tekstur* z aplikacji OpenGL.

Współrzędne tekstur, dla każdej jednostki teksturującej (0-7)

ATTRIBUTE VEC4 *gl_MultiTexCoord*{0-7}

Predefiniowana tablica tekstur o wielkości *gl_MaxTextureCoords*

UNIFORM MAT4 *gl_TextureMatrix*[*gl_MaxTextureCoords*]

Przekazanie współrzędnych tekstur

Vertex shader oblicza, dla każdego wierzchołka, wynikowe wartości współrzędnych tekstur. Nawet jeśli ich nie zodyfikuje, musi je zapisać w predifiniowanej tablicy.

Tablica z wynikowymi współrzędnymi tekstur (x - jednostka teksturująca)

VARYING VEC4 *gl_TexCoord*[x]

Przekazanie współrzędnych tekstur z *vertex shadera*

```
void main() {
  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = ftransform();
}
```

Interpolowane współrzędne tekstur

Za pomocą zmiennej *gl_TextCoord*, *pixel shader* otrzymuje interpolowane *współrzędne tekstur*. Aby odczytać dostęp do wartości tekstur, musi zostać w nim zadeklarowana specjalna zmienna wykorzystywana do przekazywania numeru jednostki teksturującej z aplikacji OpenGL.

Numer jednostki teksturującej przekazany z aplikacji OpenGL

UNIFORM SAMPLER{1, 2, 3}D *tex*

Ostateczny kolor piksela

Kolor ***teksla*** (piksela w obrazie tekstury) wyliczany jest za pomocą predefiniowanej funkcji. na podstawie ustawień tekstury zdefiniowanych w aoplikacji OpenGL, tj. filtrowanie, mipmapy, itp.

Funkcja zwracająca kolor teksla

VEC4 texture2D(*sampler2D*, *vec2*);

Argumenty funkcji

- *sampler2D* - numer jednostki teksturującej.
- *vec2* - współrzędne tekstury.

Ustawienie koloru piksela w *pixel shaderze*

```
uniform sampler2D tex;
void main() {
  vec4 color = texture2D(tex,gl_TexCoord[0].st);
  gl_FragColor = color;
}
```

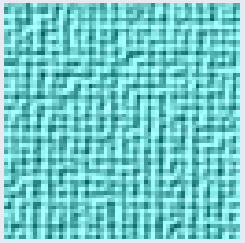
Przykład

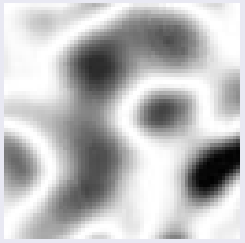
Program demonstruje teksturowanie obiektu 3D za pomocą dwóch tekstur. Kolory tekstur są ze sobą mnożone, tworząc wynikowy kolor piksela. Aplikacja OpenGL powinna wczytać, przygotować i aktywować tekstury dla jednostek teksturujących oraz przypisać numery jednostek teksturujących do zmiennych typu sampler2D.

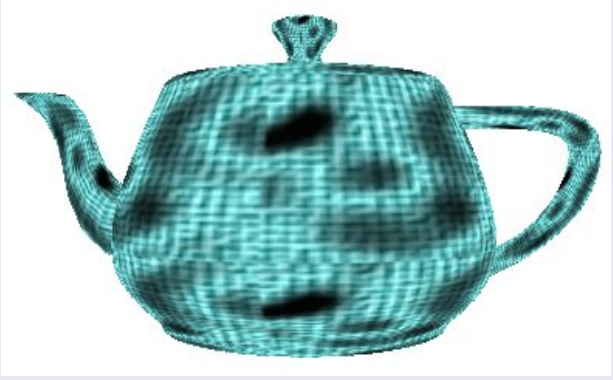
Przykładowy program

Windows

Linux







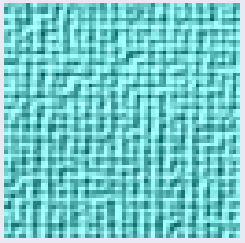
Przykład

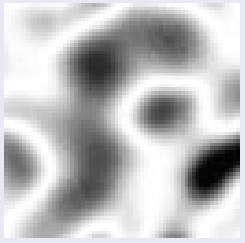
Program przedstawia obiekt 3D z nałożónymi dwoma teksturami, który dodatkowo jest oświetlony. Oświetlenie jest per pixel, a wynikowy kolor tekstury zależy od intensywności i kierunku światła. Jest to zmodyfikowana wersja prorgamu ze światłem odbitym.

Przykładowy program

Windows

Linux







Bibliografia


R. Fernando, Mark J. Kilgard:
Język Cg. Programowanie grafiki w czasie rzeczywistym..
Wydawnictwo Helion, Gliwice, 2003.


OpenGL Shading Language.
http://www.opengl.org/documentation/glsl


Lighthouse 3D - GLSL Tutorial
http://www.lighthouse3d.com/opengl/glsl



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI



Politechnika Świętokrzyska

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY



Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

" Program Rozwojowy Potencjału Dydaktycznego Politechniki Świętokrzyskiej w Kielcach: *kształcenie w nowoczesnych obszarach techniki* "

Program Operacyjny Kapitał Ludzki, Priorytet IV Szkolnictwo wyższe i nauka, Działanie 4.1. Wzmocnienie i rozwój potencjału dydaktycznego uczelni oraz zwiększenie liczby absolwentów kierunków o kluczowym znaczeniu dla gospodarki opartej na wiedzy, Poddziałanie 4.1.1. Wzmocnienie potencjału dydaktycznego uczelni.

Umowa UDA–POKL.04.01.01-00395/09-00