Programowanie Grafiki Komputerowej

OpenGL Shading Language cz. 4 - bump mapping i shadow mapping

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. Bump mapping
   - Emboss bump mapping
   - Współrzędne styczne
   - Normal mapping
2. Shadow Mapping
   - Bufor ramki
   - Shadow mapping
   - Shadow mapping w OpenGL
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.

Definicja

Bump mapping (mapowanie wybojów/wypukłości) to jedna z technik oświetlenia *per pixel* pozwalająca na symulację drobnych nierówności powierzchni (wypukłości i zagłębień) z pomocą tekstur. Dzięki wykorzystaniu *wielotekstu rowania*, umożliwia renderowanie obiektów o złożonych kształtach, nie obciążając procesora. Polega na przekształceniu wektora normalnego na poziomie pikseli.

Implementacje

Występuje wiele implementacji *bump mapping* różniących się stopniem złożoności obliczeń oraz jakością uzyskanych efektów:

- ***Emboss bump mapping*** - najstarsza i najprostsza technika, którą można zrealizować bez shaderów.
- ***Normal mapping*** - bardziej złożona obliczeniowo, pozwala na uzyskanie lepszych efektów, wykorzystuje mapy normalnych
- ***EMBM*** (ang. *Environment Mapped Bump Mapping*) - połączenie mapowania wybojów z odbiciami, wymaga specjalnego sprzętu.
- ***Parallax mapping*** - polega na przemieszczeniu współrzędnych tekstur względem kąta widzenia.
- ***Displacement mapping*** - polega na przemieszczeniu współrzędnych wierzchołków względem map wysokości.

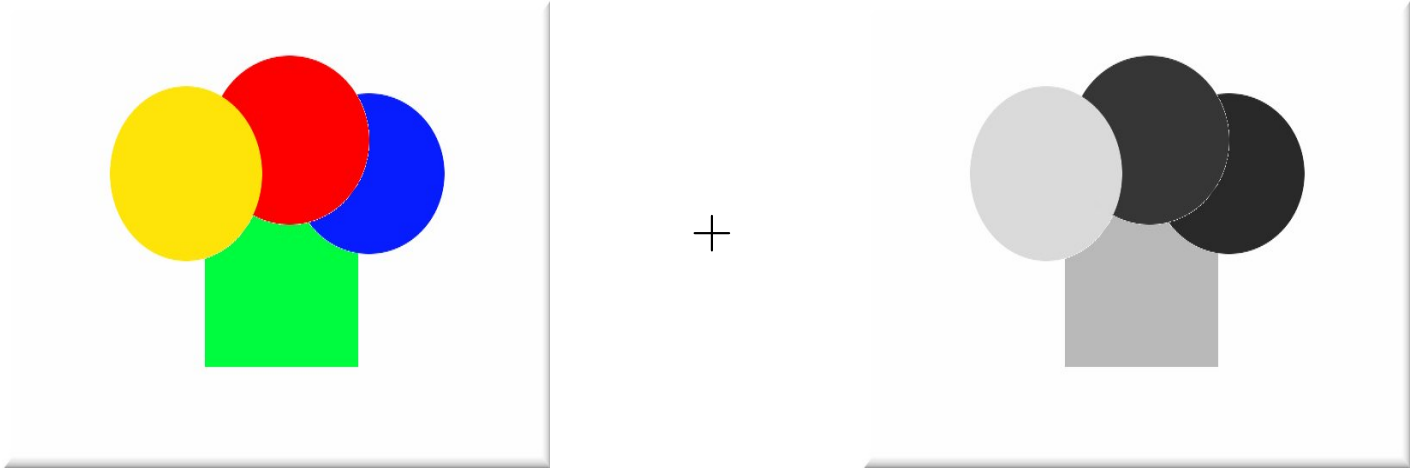
Emboss bump mapping

Do stworzenia efektu nierówności powierzchni, ***Emboss bump mapping*** wykorzystuje mapę wysokości, w której jasny kolor oznacza wypukłość, a ciemny wgniecenie. Nazwa techniki wzięła się od płaskorzeźby (ang. *emboss*), ponieważ tak wygląda mapa wysokości dla figur płaskich. Białe krawędzie są od strony padającego światła, a czarne po stronie zacienionej.

Mapa wysokości

Do określenia kształtu i położenia nierówności wykorzystywana jest bitmapa w odcieniach szarości, tzw. ***mapa wybojów/wypukłości*** (ang. *bump map*), zwana też ***mapą wysokości*** (ang. *height map*). Zazwyczaj jest ona uzupełnieniem właściwej tekstury nakładanej na obiekt. Poziomy szarości reprezentują stopień wybojów/wgnieceń, gdzie stopień szarości:

- [0-50%) - oznacza wgniecenie od największego do najmniejszego.
- 50% - reprezentuje wartość niezmienioną (poziom zerowy).
- (50-100%] - oznacza wypukłość od najmniejszej do największej.



Generowanie mapy wysokości

Mapę wysokości można utworzyć na kilka sposobów:

- zastępując złożony obiekt, składający się z dużej liczby prymitywów, jednym prymitywem z nałożoną mapą wypukłości.
- użycie skanera 3D do odczytania rzeczywistej struktury powierzchni.
- otrzymanie i zmodyfikowanie płaskiej tekstury (najbardziej powszechna metoda).
- wygenerowanie mapy wybojów za pomocą tekstur proceduralnych.
- ręczne utworzenie mapy, itp.

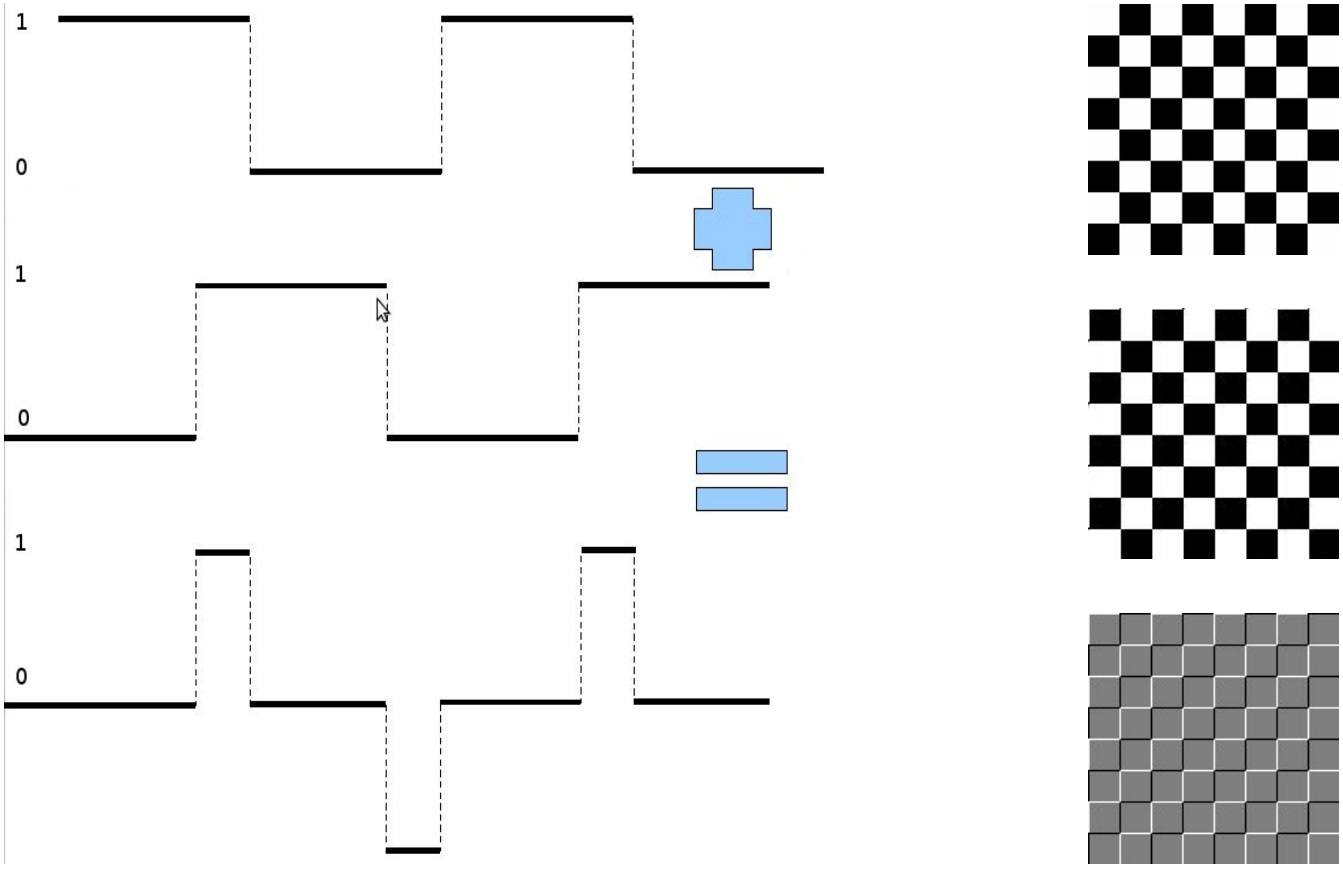
Interpretacja wybojów/wypukleń

Aby otrzymać wrażenie wypukłości należy wykonać na mapie:

- utworzenie negatywu.
- pobranie wartości piksela z negatywu.
- przesunięcie źródłowej mapy w kierunku padającego światła.
- pobranie wartości piksela.
- dodanie wartości pikseli.

Po zsumowaniu przesuniętej mapy wybojów i jej negatywu otrzymujemy rzut wektora padania światła do współrzędnych stycznych tekstury nałożonej na powierzchnię.

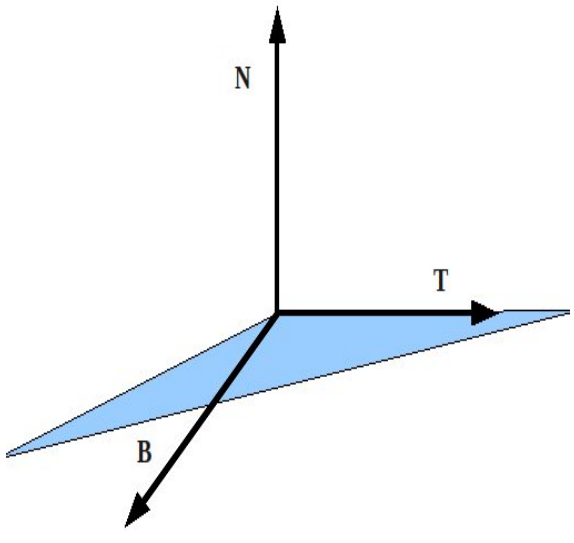
Sumowanie map wybojów



Układ współrzędnych stycznych

Układ współrzędnych styczny do powierzchni (ang. *tangent space*) to układ współrzędnych związany z powierzchnią obiektu. Opisany jest z pomocą trzech prostopadłych do siebie wektorów:

- ***Wektor normalny (N)*** (ang. *Normal*) - normalna powierzchni.
- ***Wektor styczny (T)*** (ang. *Tangent*) - styczna do powierzchni, współrzędna S tekstury.
- ***Wektor binormalny (B)*** - binormalna powierzchni, współrzędna T tekstrury.



Każdy wierzchołek wielokąta posiada zwykle swój własny układ współrzędnych stycznych. W każdym wierzchołku może być inna normalna.

Konwersja do współrzędnych stycznych

Wektor normalny jest wyrażany w *układzie współrzędnych stycznych*, ale pozostałe wektory nie są, tj. wektor światła. Aby realizować operacje dotyczące tekstury, wszystkie wektory należy przekształcić do współrzędnych stycznych. Wektory konwertuje się za pomocą tzw. ***macierzy TBN*** złożonej z wetkorów T, B i N.

Konwersja wektora do współrzędnych stycznych

$$\begin{bmatrix} x_{wy} \\ y_{wy} \\ z_{wy} \end{bmatrix} = \begin{bmatrix} x_T & x_B & x_N \\ y_T & y_B & y_N \\ z_T & z_B & z_N \end{bmatrix} \cdot \begin{bmatrix} x_{we} \\ y_{we} \\ z_{we} \end{bmatrix}$$

Przekazanie współrzędnych stycznych do shadera

Shader ma dostęp do współrzędnych stycznych w następujący sposób:

- *Wektor normalny* (N) - jest przekazywany w predefiniowanej zmiennej *gl_Normal*.
- *Wektor styczny* (T) - jest niezależny od transformacji i niezmienny w czasie, musi być obliczony przez aplikację OpenGL i dostarczony do shadera, jako zmienna typu *attribute* lub we współrzędnej tekstury (S, T, R lub Q) jednej z jednostek teksturujących.
- *Wektor binormalny* (B) - może zostać wyliczony w shaderze, jako iloczyn wektorowy wektorów N i T.

Uwagi

Wszystkie wektory powinny zostać poddane takim samym transformacjom jak obiekt.

Przekazanie wektora stycznego z aplikacji

Przekazanie wektora stycznego z aplikacji

```
void rysujSzescian(){
    float tang[4];
    glBegin(GL_QUADS);
    for(int i=0;i < 6;i++) {
      glNormal3fv(box_normal[i]);
      // Wyznaczenie wektora stycznego (jeden dla kadej ciany):
      CalcTangentVector(
        box_vertex[i*4+0],    box_vertex[i*4+1],    box_vertex[i*4+3],
        box_texcoord[i*4+0],  box_texcoord[i*4+1],  box_texcoord[i*4+3],
        box_normal[i], tang
      );
      // Wspolrzedne tekstur, wektor styczny i wierzcholek sciany:
      for(int j=0;j < 4;j++) {
        glMultiTexCoord2fv(GL_TEXTURE0, box_texcoord[i*4 + j]);
        glMultiTexCoord4fv(GL_TEXTURE1, tang);
        glVertex3fv(box_vertex[i*4 + j]);
      }
        }
    glEnd();
}
```

Emboss mapping - Vertex shader

Emboss mapping - Vertex shader

```
const float BUMP_SHIFT = 0.28;  // Moc efektu
varying vec3 normal, lightDir;

void main() {
    // Skonwertowanie wektora normalnego do wsplrzednych swiata
  normal = normalize(gl_NormalMatrix * gl_Normal);
  // Normalizacja polozenia swiatla
  lightDir = normalize(vec3(gl_LightSource[0].position));

  // Wektory TANGENT i BINORMAL:
  vec3 t = normalize(gl_NormalMatrix * gl_MultiTexCoord1.xyz);
  vec3 b = cross(normal, t) * gl_MultiTexCoord1.w;

  // Przesuniecie wspolrzednych tekstury nr 1:
  vec4 texc = gl_MultiTexCoord0;
  texc.s += dot(lightDir, t) * BUMP_SHIFT; // Rzut do powierzchni stycznej
  texc.t += dot(lightDir, b) * BUMP_SHIFT; // J.w.

  // Przypisanie wspolrzednych tekstur:
  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_TexCoord[1] = texc;

  gl_Position = ftransform();
}
```

Emboss mapping - Fragment shader

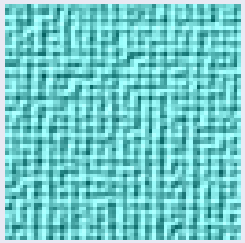
Emboss mapping - Fragment shader

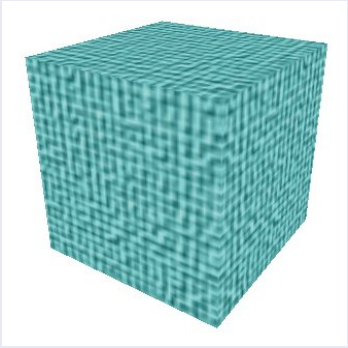
```
varying vec3 normal, lightDir;
uniform sampler2D tex0, bumpMap;
vec4 kolor;
void main() {
  float intensity = max(dot(normal, lightDir), 0.0);
  vec4 diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
  vec4 ambient = gl_FrontMaterial.ambient* gl_LightSource[0].ambient;
  vec4 globalAmbient = gl_LightModel.ambient * gl_FrontMaterial.ambient;
    vec4 specular = specular = vec4(0.0);
  if (intensity > 0.0) {
    float intensityHV = max(dot(normal, gl_LightSource[0].halfVector.xyz),0.0);
    specular = gl_FrontMaterial.specular * gl_LightSource[0].specular *
        pow(intensityHV,gl_FrontMaterial.shininess);
  }
  // Wyliczenie koloru piksela na podstawie pierwszej tekstury:
  kolor = texture2D(tex0, gl_TexCoord[0].st);

  // Wyznaczenie jasnosci wybrzuszen za pomoca odejmowania map
  vec4 bumpz = texture2D(bumpMap, gl_TexCoord[1].st) +
                                  1.0 - texture2D(bumpMap, gl_TexCoord[0].st);
  // Kolor piksela z nalozonymi wybrzuszeniami na teksture i oswietlenie
  gl_FragColor = kolor * bumpz * diffuse + ambient + globalAmbient + specular;
}
```

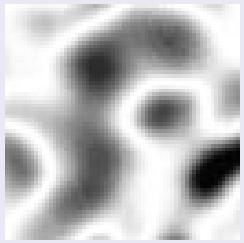
Przykład

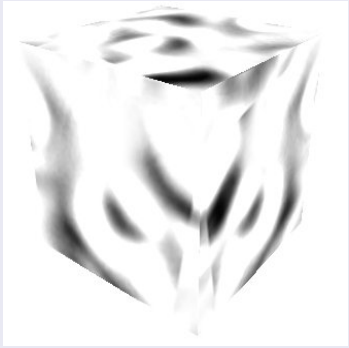
Obiekt z teksturą





Mapa wybojów z tekstury

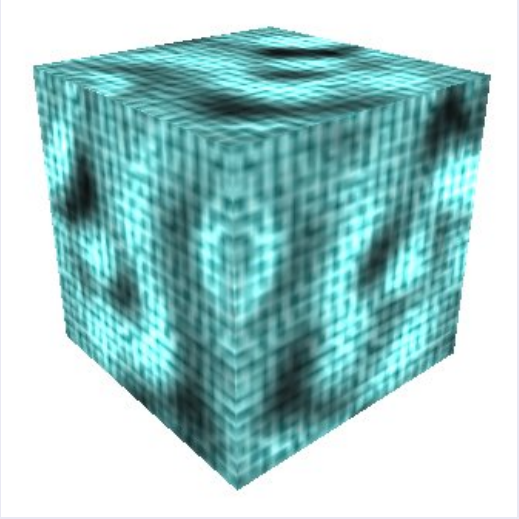




Przykładowy program

Windows

Linux

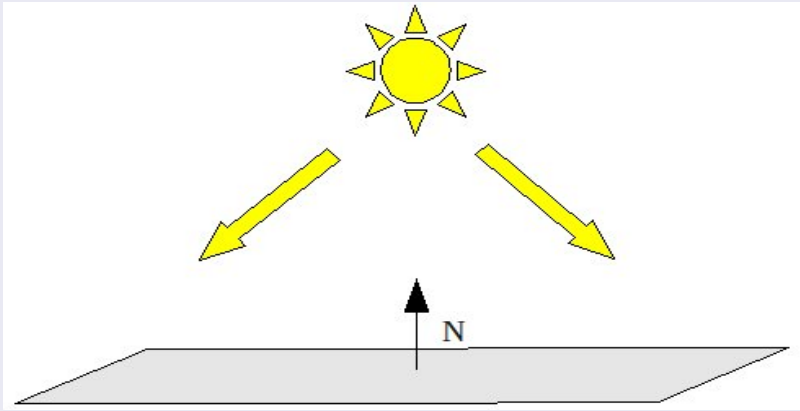


Normal mapping

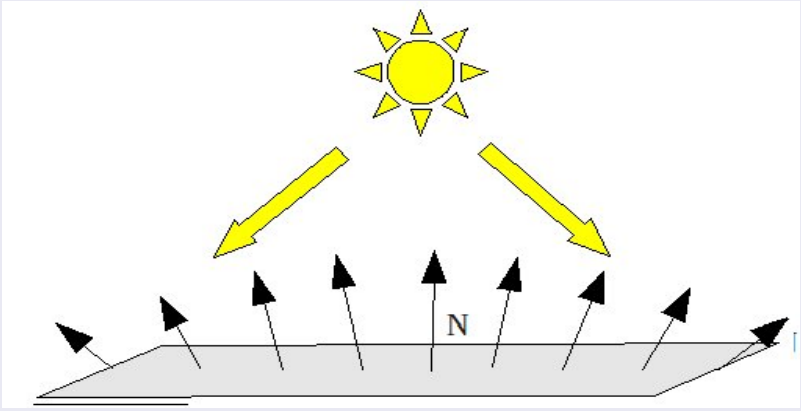
Normal mapping (lub *Dot3 bump mapping*) jest techniką symulującą oświetlenie wybrzuszeń i wgnieceń dla każdego piksela. Umożliwia renderowanie szczegułów obiektów, bez użycia dużej ilości wielokątów. Wykorzystuje ***mapy normalnych***, w których zapisane są współrzędne X, Y i Z powierzchoniowych wektorów normalnych.

Porównanie modeli cieniowania

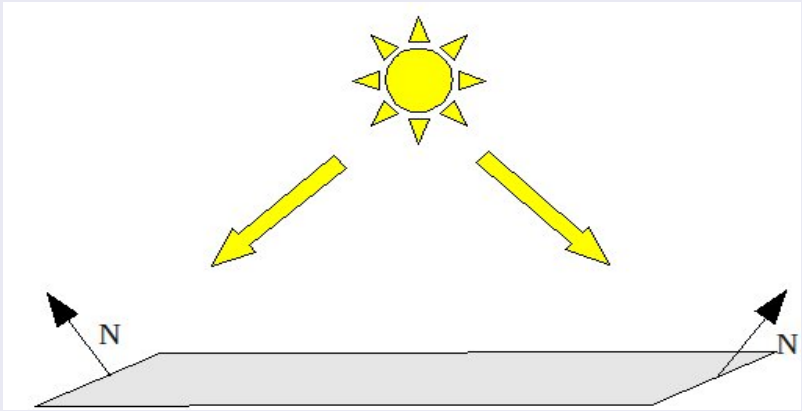
Cieniowanie płaskie



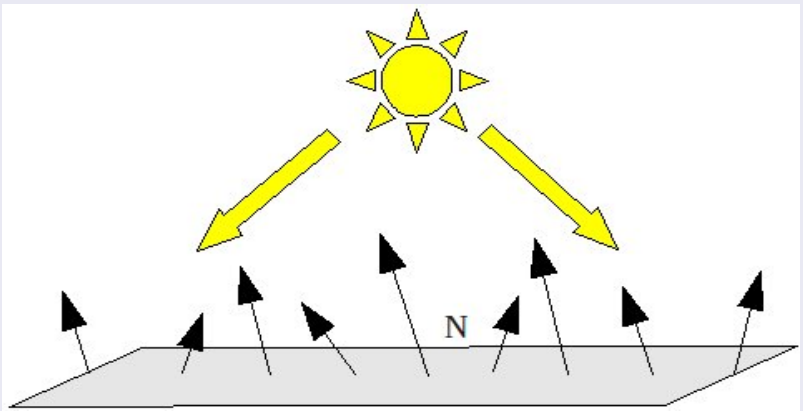
Cieniowanie Phonga



Cieniowanie Gourauda



Normal mapping



Mapa normalnych

Mapy normalnych są wykorzystywane do przechowywania wektorów normalnych. Wektor normalny wskazuje kierunek prostopadły do powierzchni i jest jednostkowy. Do zapisania kierunku w przestrzeni, potrzeba trzech zmiennych - x, y i z. Na mapie normalnych, wykorzystuje się 3 kanały kolorów do zakodowania informacji o kierunku wektora:

- kanał czerwony (R) - składowa x.
- kanał zielony (G) - składowa y.
- kanał niebieski (B) - składowa z.

Uwagi

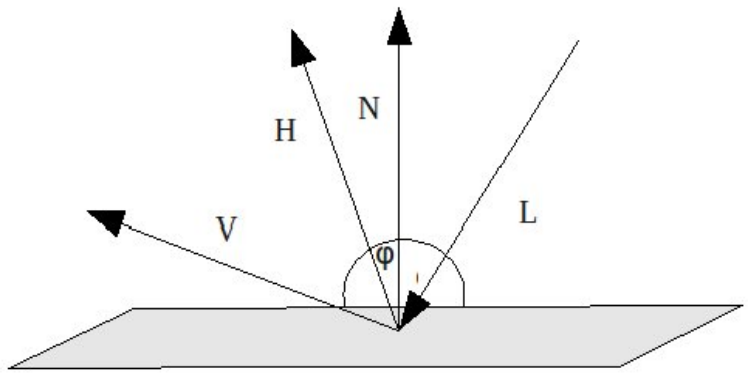
Wektory normalne mogą posiadać przeciwne zwroty, dlatego wartości [-1,1] odwozrowuje się jako przedział [0,1].

Obliczenie oświetlenia

Oświetlenie w każdym punkcie jest wyliczane na podstawie 3 wektorów. Tylko wektor normalny (odczytywany z tekstury mapy normalnych) jest zapisany w układzie stycznym powierzchni, pozostałe wektory należy przekonwertować mnożąc je przez macierz TBN.

Do obliczenia oświetlenia używany jest:

- ***wektor normalny powierzchni*** (N).
- ***wektor kierunku padania światła*** (L).
- ***wektor kierunku patrzenia obserwatora*** (V).



Normal mapping - Vertex shader

Normal mapping - Vertex shader

```
varying vec3 normal,lightDir, viewDir;
void main() {
  gl_Position = ftransform();
  // Wektor normalny, TANGENT i BINORMAL:
  vec3 n = normalize(gl_NormalMatrix * gl_Normal);
  vec3 t = normalize(gl_NormalMatrix * gl_MultiTexCoord2.xyz);
  vec3 b = cross(n, t) * gl_MultiTexCoord1.w;

  // Macierz TBN:
  mat3 tbnMatrix = mat3(  t.x, b.x, n.x,
                          t.y, b.y, n.y,
                          t.z, b.z, n.z);
  // Kierunek swiatla i jego konwersja do wspolrzednych stycznych:
  lightDir = gl_LightSource[0].position.xyz - gl_Position.xyz;
  lightDir = tbnMatrix * lightDir;

  // Kierunek patrzenia obserwatora
  viewDir = -gl_Position.xyz;
  viewDir = tbnMatrix * viewDir;
  normal = tbnMatrix * n;

  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_TexCoord[1] = gl_MultiTexCoord0;
}
```

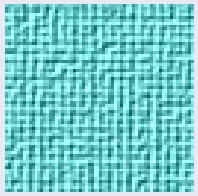
Normal mapping - Fragment shader

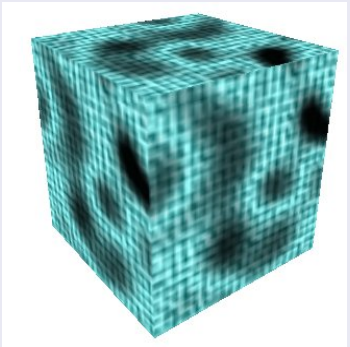
Normal mapping - Fragment shader

```
uniform sampler2D tex0, tex1, normalMap;
varying vec3 normal, lightDir, viewDir; vec4 kolor0, kolor1;
void main() {
  // Wektor normalny, kierunek padania swiatla i patrzenia obserwatora
  vec3 n = normalize(texture2D(normalMap, gl_TexCoord[0].st).xyz * 2 - 1.0);
  vec3 l = normalize(lightDir);
  vec3 v = normalize(viewDir);
  // Wyliczenie wektora polowkowego:
  vec3 h = normalize(l + v);
  float intensity = max(dot(n, l), 0.0);
  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;
  vec4 specular = vec4(0.0);
  if (intensity > 0.0) {
    float intensityHV = max(dot(n, h),0.0);
    specular = gl_FrontMaterial.specular * gl_LightSource[0].specular *
               pow(intensityHV,32);
  }
  kolor0 = texture2D(tex0, gl_TexCoord[0].st);
  kolor1 = texture2D(tex1, gl_TexCoord[1].st);
  gl_FragColor = kolor0 * kolor1 * intensity * diffuse + ambient +
            globalAmbient + specular;
}
```

Przykład

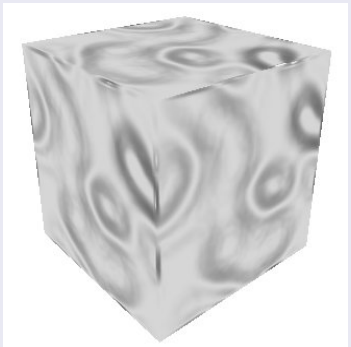
Obiekt z teksturami





Mapa normalnych

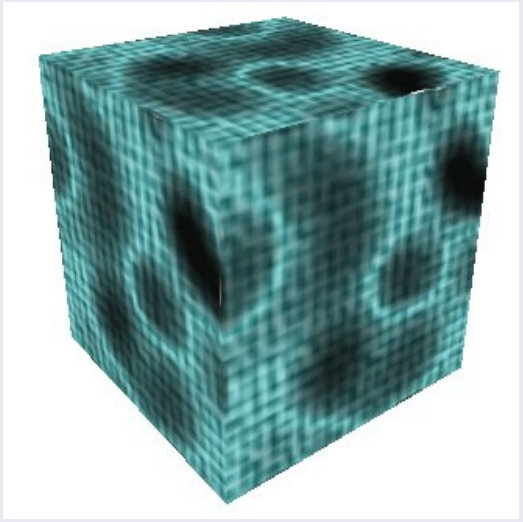




Przykładowy program

Windows

Linux



Generowanie map normalnych

Mapy normalnych generowane są za pomocą specjalnych narzędzi:

- ***NVIDIA Texture Tools*** - Adobe Photoshop Normal Map and DDS Authoring Plugins.
  http : //developer.nvidia.com/object/nv_texture_tools.html.
- ***ATI Developer Tools*** - NormalMapGenerator.
  http : //developer.amd.com/gpu/radeon/archives/bumpmap/Pages/default.aspx.

Bufor ramki

Bufor ramki (FBO - ang. *framebuffer object*) pozwala na renderowanie do buforów znajdujących się w pamięci RAM oraz do tekstur (ang. *render to texture*).

Operacje na buforze ramki

- ***glGenFramebuffers(Gluint count, Gluint *buffer)*** - utworzenie obiektu bufora.
- ***glBindFramebuffer(GLenum mode, Gluint buffer)*** - aktywacja bufora (*mode: GL_FRAMEBUFFER*).
- ***glDrawBuffer(GLenum mode)*** - zablokowanie rysowania w buforze koloru (*mode: GL_NONE*).
- ***glReadBuffer(GLenum mode)*** - zablokowanie odczytu z bufora koloru (*mode: GL_NONE*).

Renderowanie do tekstury

Bufor ramki wykorzystuje się do utworzenia ***tekstury głębokości***, zawierającej informacje tylko o głębokości obiektów na scenie.

Renderowanie do tekstury

```
// Utworzenie tekstury:
GLuint depthTex;
glGenTextures(1, &depthTex);
glBindTexture(GL_TEXTURE_2D, depthTex);

// Utworzenie miejsca na teksture 2D:
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, WIDTH, HEIGHT, 0,
             GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);

// Powiazanie tekstury z aktywnym buforem ramki:
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                       GL_TEXTURE_2D, depthTex, 0);
```

Sprawdzenie stanu bufora ramki

Przed wykorzystaniem tekstury z mapą cieni, należy sprawdzić czy operacja renderowania się powiodła. Ostatecznie należy ustawić renderowanie na standardowe bufory OpenGL.

Sprawdzenie stanu bufora ramki

```
GLenum stat = glCheckFramebufferStatus(GL_FRAMEBUFFER);
if(stat != GL_FRAMEBUFFER_COMPLETE)
    printf("FBO ERROR!!");
```

Wskazanie bufora do renderowania

```
glBindFramebuffer(GL_FRAMEBUFFER, fbo);   // bufor ramki
glBindFramebuffer(GL_FRAMEBUFFER, 0);   // bufory OpenGL
```

Mapa cieni

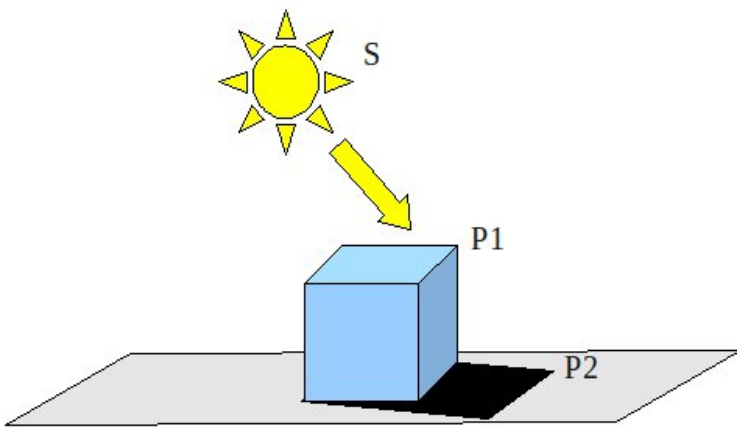
Mapowanie cieni (ang. *shadow mapping*) jest techniką generowania cieni sprawdzającą, czy każdy piksel jest widoczny z pozycji źródła światła. Widoczność piksela określana jest na podstawie mapy cieni, w której zapisana jest zawartość *Z-bufora*. Zawiera on informacje o odległości między źródłem światła, a najbliższym (oświetlonym przez to światło) obiektem.

Mapa cieni powstaje z użyciem standardowego algorytmu Z-Bufora. Punkt obserwacji zostaje przesunięty do miejsca położenia światła i zrzutowany na powierzchnię. Dla światła punktowego wykonuje się rzut perspektywiczny, a rzut równoległy dla światła kierunkowego.

Sprawdzenie widoczności pikseli

W celu sprawdzenia widoczności pikseli, obliczana jest odległość od źródła światła S, dla punktu P1 (obiektu rzeczywistego) i punktu P2 (odczytanego z mapy cieni). Wykonywane jest porównanie, w którego rezultacie wynikowy piksel:

- pozostaje bez zmian, gdy $| S, \quad P1 | = | S, \quad P2 |$.
- jest przyciemniany, gdy $| S, \quad P1 | < | S, \quad P2 |$



Schemat renderowania sceny z cieniem

1. przesunięcie kamery do punktu położenia źródła światła.
2. wyłączenie rysowania w buforze koloru.
3. renderowanie sceny z obiektami.
4. zapis zawartości *Z-bufora* do mapy cieni.
5. włączenie rysowania w buforze koloru.
6. ponowne renderowanie sceny z przyciemnieniem pikseli, które są niewidoczne z punktu położenia źródła światła.

Schemat renderowania sceny z cieniem

1. przesunięcie kamery do punktu położenia źródła światła.
2. wyłączenie rysowania w buforze koloru.
3. renderowanie sceny z obiektami.
4. zapis zawartości *Z-bufora* do mapy cieni.
5. włączenie rysowania w buforze koloru.
6. ponowne renderowanie sceny z przyciemnieniem pikseli, które są niewidoczne z punktu położenia źródła światła.

Uwagi

Dokładność mapy cieni zależy od jej rozmiaru i precyzji działania *Z-bufora*.

Transformacje mapy cieni

Przed nałożenim mapy cieni na oświetloną scenę, współrzędne punktów sceny muszą zostać zamienione na współrzędne widoczne z pozycji światła. Przygotowywana jest macierz transformacji współrzędnych, która jest iloczynem *macierzy widoku modelu* i *macierzy rzutowania*. Odpowiada ona transformacji realizowanej podczas renderowania mapy cieni.

W rezultacie otrzymywane są ***współrzędne homogeniczne***, które muszą zostać znormalizowane i podzielone, aby uwzględnić perspektywę (dla punktowych źródeł światła).

Zakres współrzędnych tekstury

Dla punktów widocznych z pozycji światła, współrzędne znormalizowane są z zakresu [-1, 1]. Zakres tych współrzędnych musi zostać dopasowany do zakresu ***współrzędnych tekstury*** [0, 1]. W tym celu, macierz przekształceń należy pomnożyć przez dodatkową macierz.

Macierz konwersji do współrzędnych tekstury

$$\begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Przekształcenie współrzędnych tekstury

Przekształcenie współrzędnych tekstury

```
static double modelView[16], projection[16];
// Macierz transformujaca wspolrzedne
const GLdouble bias[16] = { 0.5, 0.0, 0.0, 0.0,
                            0.0, 0.5, 0.0, 0.0,
                            0.0, 0.0, 0.5, 0.0,
                            0.5, 0.5, 0.5, 1.0};
// Odczyt macierzy widoku modelu i rzutowania
glGetDoublev(GL_MODELVIEW_MATRIX, modelView);
glGetDoublev(GL_PROJECTION_MATRIX, projection);

// Macierz transformujaca -> macierz tekstury nr 7:
glMatrixMode(GL_TEXTURE);
glActiveTexture(GL_TEXTURE7);

// Macierz konwertujaca:
glLoadIdentity();
glLoadMatrixd(bias);

// Dolaczenie macierzy rzutowania i widoku modelu:
glMultMatrixd(projection);
glMultMatrixd(modelView);

// Powrot do standardowej macierzy widoku modelu:
glMatrixMode(GL_MODELVIEW);
```

Implementacje shadow mappingu w OpenGL

- standardowe funkcje OpenGL tj. automatyczne generowanie współrzędnych tekstur, test kanału alfa oraz rozszerzenia, tj. *GL_ARB_texture_env_combine*.
- shadery i mapa cieni wczytana do *fragment shadera* jako *sampler2D* - programowa realiacja perspektywy światła i porównania odległości.
- shadery i mapa cieni wczytana jako *sampler2DShadow* - sprzętowa realiacja perspektywy światła i porównania odległości. *Fragment shader* tylko przyciemnia piksele leżące w cieniu.

Wczytanie mapy cieni, jako *sampler2DShadow*

Aby móc skorzystać we *fragment shaderze* z typu *sampler2DShadow* i funkcji *shadow2DProj*, należy w aplikacji OpenGL włączyć sprzętowe porównanie współrzędnej R tekstury z wartością tekseli z aktualnie wybranej tekstury głębokości (mapy cieni).

Włączenie sprzętowego porównanie współrzędnej R tekstury

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
                               GL_COMPARE_R_TO_TEXTURE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
```

Zapisanie wyniku jako intensywności

```
glTexParameteri(GL_TEXTURE_2D, GL_DEPTH_TEXTURE_MODE, GL_INTENSITY);
```

Shadow mapping - Vertex shader

Shadow mapping - Vertex shader

```
varying vec4 ShadowCoord; // Do fragment shadera:

void main() {
  // Wspolrzedna tekstury w mapie cieni,
  // uzyskana ze wspolrzednej wierzcholka przeksztalcona
  // przez macierz konwertujaca tex7:
  ShadowCoord = gl_TextureMatrix[7] * gl_Vertex;

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

Shadow mapping - Fragment shader

Shadow mapping - Fragment shader

```
uniform sampler2DShadow ShadowMap;// Mapa cieni
varying vec4 ShadowCoord; // Z vertex shadera

void main() {
  // Uzywajac sampler2DShadow funkcja shadow2DProj zwraca
  // wynik porownania (0 lub 1) zamiast odleglosci
  // Do wspolrzednej R dodajemy przesuniecie, aby zmniejszyc more
  // (przy filtrowaniu GL_NEAREST):
  vec4 distanceFromLight = shadow2DProj(ShadowMap, ShadowCoord +
                                        vec4(0, 0, 0.02, 0));

  float shadow = 1.0;
  if(distanceFromLight.w < 1.0)
    shadow = 0.4;

  // Wierzcholki w cieniu zostaja przyciemnione
  gl_FragColor = gl_Color * shadow;
}
```

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