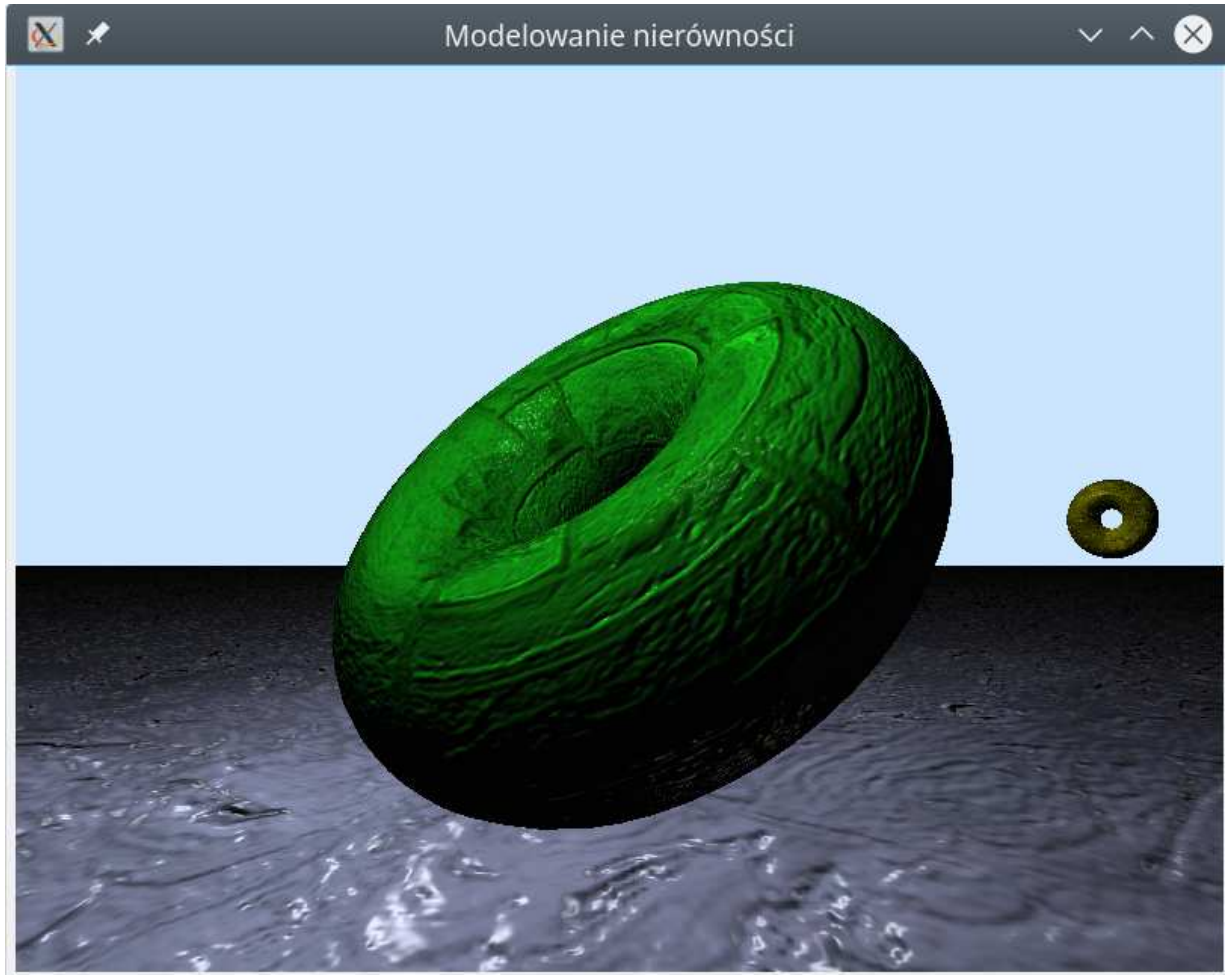
Programowanie 3W grafiki w OpenGL. Mapowanie wypukłości

Aleksander Denisiuk
Polsko-Japońska Akademia Technik Komputerowych
Wydział Informatyki w Gdańsku
ul. Brzegi 55
80-045 Gdańsk

denisjuk@pja.edu.pl

Najnowsza wersja tego dokumentu dostępna jest pod adresem
http://users.pja.edu.pl/~denisjuk/



Czynności

- Przygotowanie specjalnej tekstury, mapy wektorów normalnych
  - GIMP, wtyczka normalmapping
  - Blender: `Bake` (`Normals`)
- Przy obliczeniu oświetlenia pobieramy wektor normalny z tekstury
  - potrzebna jest baza wektorów stycznych do powierzchni
    - wektor styczny t i drugi wektor styczny b
    - $2(r, g, b) - 1 \iff (x, y, z)$
  - współrzędne pobranego z tekstury wektora dane są w układzie (t, b, n)
  - kierunki do kamery, do światła — w układzie globalnym
    - przeliczyć wszystkie wektory do jednego układu
      - po co i do którego?

Torus

- parametryzacja: $P(\theta, \varphi) = \begin{pmatrix} (R + r\cos\varphi)\sin\theta \\ r\sin\varphi \\ (R + r\cos\varphi)\cos\theta \end{pmatrix}$,
- wektory styczny: $t(\theta, \varphi) = \begin{pmatrix} \cos\theta \\ 0 \\ -\sin\theta \end{pmatrix}$,
- drugi wektory styczny: $b(\theta, \varphi) = \begin{pmatrix} -\sin\varphi\sin\theta \\ \cos\varphi \\ -\sin\varphi\cos\theta \end{pmatrix}$
- ortogonalizować

Sfera

- parametryzacja: $P(\theta, \varphi) = \begin{pmatrix} R\cos\varphi\sin\theta \\ R\sin\varphi \\ R\cos\varphi\cos\theta \end{pmatrix}$,
- wektory styczny: $t(\theta, \varphi) = \begin{pmatrix} \cos\theta \\ 0 \\ -\sin\theta \end{pmatrix}$,
- drugi wektory styczny: $b(\theta, \varphi) = \begin{pmatrix} -\sin\varphi\sin\theta \\ \cos\varphi \\ -\sin\varphi\cos\theta \end{pmatrix}$

Płaszczyzna

- parametryzacja: $P(u,v) = \begin{pmatrix} u \\ 0 \\ v \end{pmatrix}$,
- wektory styczny: $t(u,v) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$,
- drugi wektory styczny: $b(u,v) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$,

Uwaga o drugim wektorze stycznym

- Drugi wektory styczny można obliczyć jako iloczyn wektorowy $b = n \times t$
- Można nawet w shaderze:

```
b=cross(n,t)
```

Shader wierzchołków

- dane wejściowe

```
layout(location=0) in vec4 in_position;
layout(location=2) in vec2 in_texture;
layout(location=3) in vec3 in_normal;
layout(location=4) in vec3 in_tangent;
layout(location=5) in vec3 in_bitangent;
```

Dane wyjściowe

- przeliczamy `light_dir` oraz `view_dir` do bazy (t, b, n)

```
out struct Vertex {
    vec2  texcoord;
    vec3  light_dir;
    vec3  view_dir;
    float dist;
} frag_vertex;
```

Początek `main`

- jak wcześniej

```
void main(void){
  vec4 vertex = model_matrix * in_position;
  frag_vertex.light_dir
    = (light.position.xyz - vertex.xyz);
  vec4 camera
    = -view_matrix*vec4(0.0, 0.0, 0.0, 1);
```

Zmiany wektorów

- obliczamy współrzędne wektorów (t, n, b) w układzie globalnym

```
vec3 e_normal
    = normalize(normal_matrix * in_normal);
vec3 e_tangent
    = normalize(normal_matrix * in_tangent);
vec3 e_bitangent
    = normalize(normal_matrix * in_bitangent);
```

Macierz zmiany bazy

- dla maciery ortogonalnej macierz odwrotna zgadza się z macierzą transponowaną: $A^{-1} = A^t$

```
mat3 e_tbn = transpose(
    mat3( e_tangent, e_bitangent, e_normal )
);
```

Zmiana bazy na (t, b, n)

- przeliczamy współrzędne kierunków do kamery i do światła

```
frag_vertex.view_dir
  = normalize(e_tbn*(camera.xyz-vertex.xyz));
frag_vertex.light_dir
  = normalize(e_tbn*frag_vertex.light_dir);
```

Drugi unit teksturowy

- W shaderze którym?

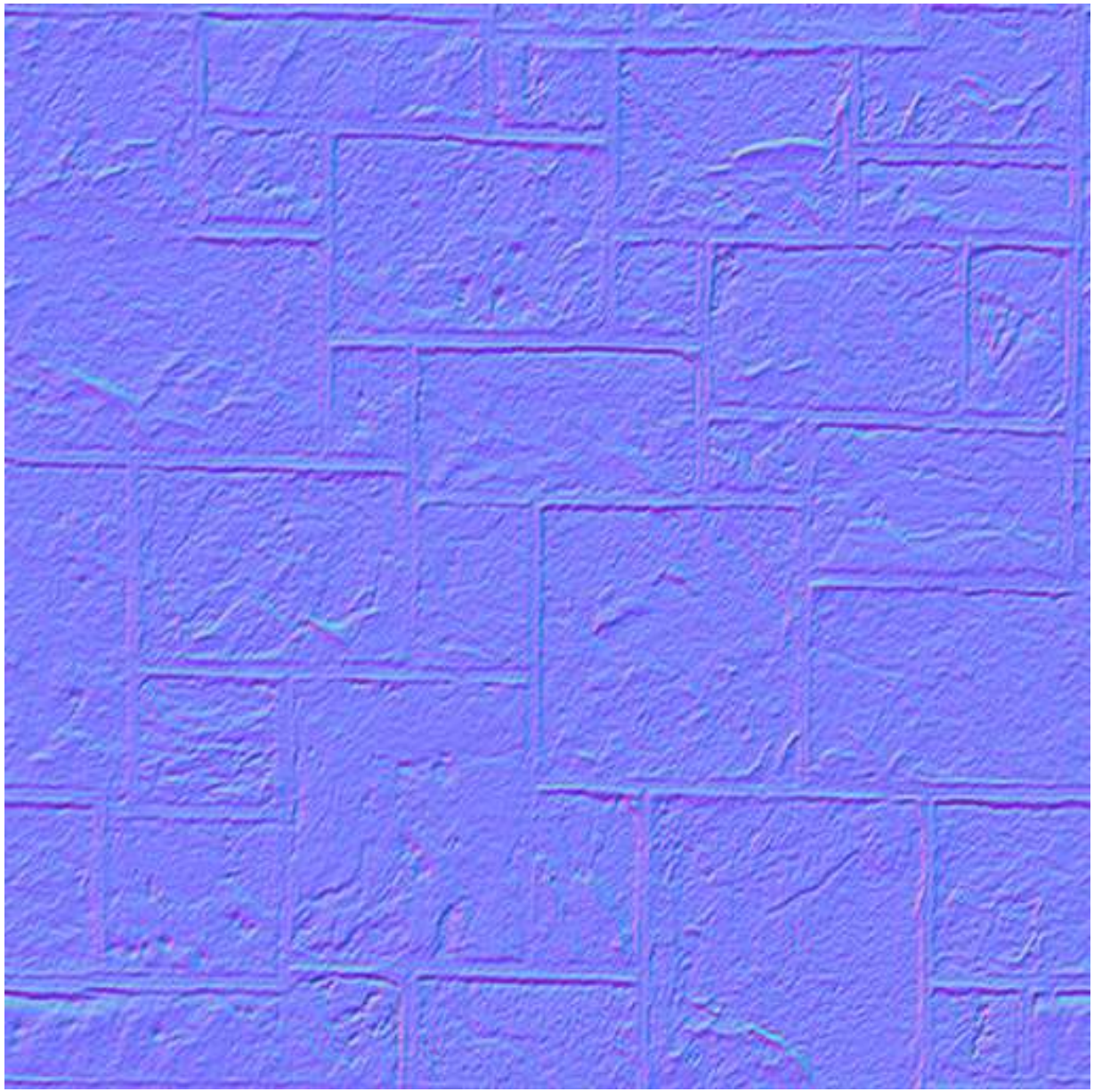
```
uniform sampler2D texture_unit;
uniform sampler2D bump_texture_unit;
```

Obliczenia w shaderze fragmentów


```
vec3 light_dir
   = normalize(frag_vertex.light_dir);
vec3 view_dir
   = normalize(frag_vertex.view_dir);
vec3 normal
   = normalize(texture(bump_texture_unit,
     frag_vertex.texcoord ).rgb*2.0-1.0);
```

- dalej wszystko jak wcześniej

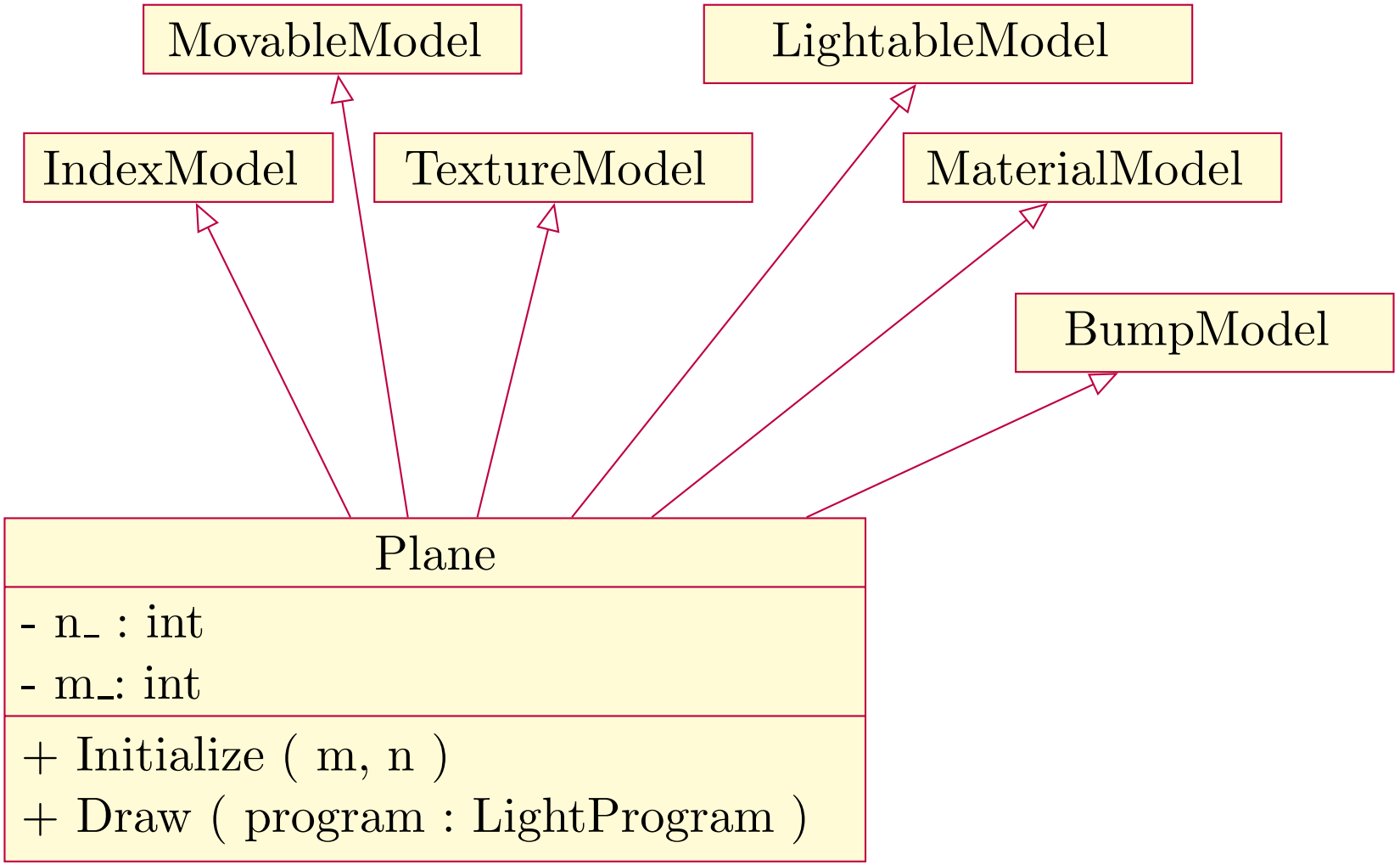
Mapa wektorów normalnych



```
class BumpModel{
public:
    void SetBumpTextureUnit(GLuint t)
      {bump_texture_unit_=t;}
    void SetBumpTexture(GLuint t){bump_texture_ = t;}
protected:
    GLuint bump_texture_unit_;
    GLuint bump_texture_;
};
```

Nowa struktura wierzchołków


```
typedef struct BumpTextureVertex{
    float position[4];
    float texture[2];
    float normal[3];
    float tangent[3];
    float bitangent[3];
} BumpTextureVertex;
```



Uzupełnienie procedury generacji płaszczyzny


```
vertices[pos].texture[0]=(float)j/(float)m_;
vertices[pos].texture[1]=(float)i/(float)n_;
vertices[pos].normal[0]=0.0f;
vertices[pos].normal[1]=1.0f;
vertices[pos].normal[2]=0.0f;
vertices[pos].tangent[0]=0.0f;
vertices[pos].tangent[1]=0.0f;
vertices[pos].tangent[2]=1.0f;
vertices[pos].bitangent[0]=1.0f;
vertices[pos].bitangent[1]=0.0f;
vertices[pos].bitangent[2]=0.0f;
```

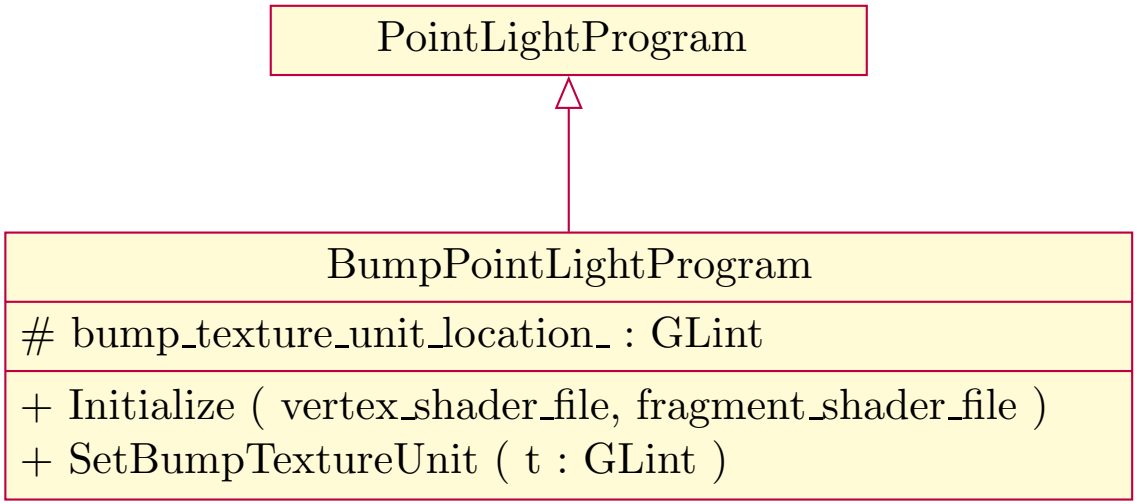
Uzupełnienie kodu aktywacji argumentów shadera


```
glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE,
    sizeof(vertices[0]),
    (GLvoid*)(sizeof(vertices[0].position)
        + sizeof(vertices[0].texture)
        + sizeof(vertices[0].normal)));
glEnableVertexAttribArray(4);
glVertexAttribPointer(5, 3, GL_FLOAT, GL_FALSE,
    sizeof(vertices[0]),
    (GLvoid*)(sizeof(vertices[0].position)
        + sizeof(vertices[0].texture)
        + sizeof(vertices[0].normal)
        + sizeof(vertices[0].tangent)) );
glEnableVertexAttribArray(5);
```

Uzupełnienie procedury wyświetlania


```
.....................................
glActiveTexture(texture_unit_);
glBindTexture(GL_TEXTURE_2D, texture_);
glActiveTexture(bump_texture_unit_);
glBindTexture(GL_TEXTURE_2D, bump_texture_);
```

BumpPointLight



```
.............................
color_texture_.Initialize(kColorTextureFile);
ice_texture_.Initialize(kIceTextureFile);
bump_color_texture_.Initialize(kBumpColorTextureFile);
bump_ice_texture_.Initialize(kBumpIceTextureFile);
```

```
.............................
plane_.Initialize(kPlaneM, kPlaneN);
plane_.SetTexture(ice_texture_);
plane_.SetTextureUnit(GL_TEXTURE0);
plane_.SetBumpTexture(bump_ice_texture_);
plane_.SetBumpTextureUnit(GL_TEXTURE2);
plane_.SetMaterial(kIceMaterial);
```

```
  ..............................
bump_point_program_.Initialize(
    kBumpPointLightVertexShader,
    kBumpPointLightFragmentShader);
glUseProgram(bump_point_program_);
bump_point_program_.SetLight(kPointLight);
bump_point_program_.SetTextureUnit(0);
bump_point_program_.SetBumpTextureUnit(2);
bump_point_program_.SetProjectionMatrix(
    projection_matrix_);
bump_point_program_.SetViewMatrix(view_matrix_);
```

SetViewMatrix, SetProjectionMatrix

- Dodać `bump_point_program_`