Zaawansowane systemy programowania grafiki. Modelowanie nierówności

Aleksander Denisiuk
Uniwersytet Warmińsko-Mazurski
Olsztyn, ul. Słoneczna 54
denisjuk@matman.uwm.edu.pl

25 maja 2021

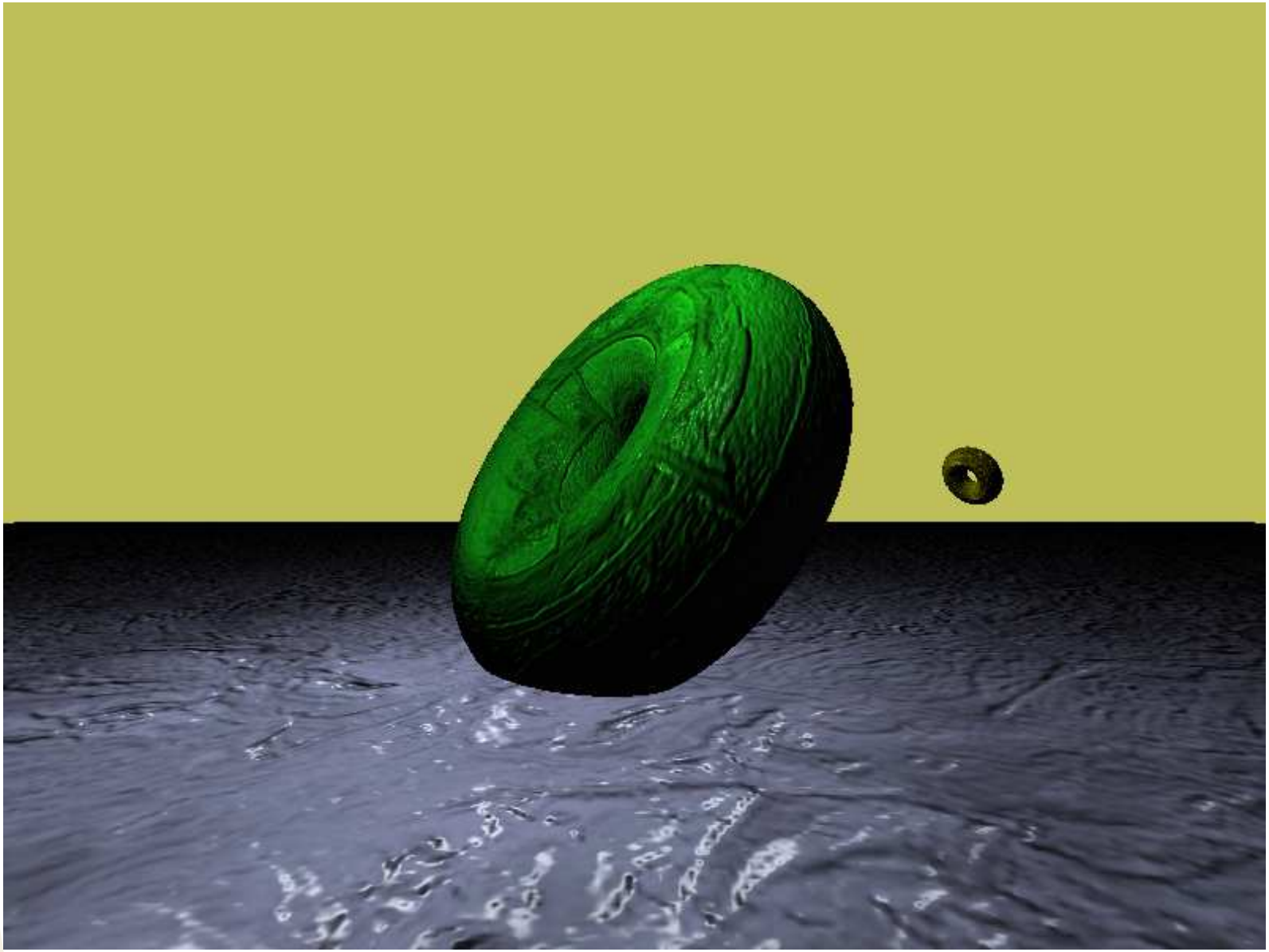
Modelowanie nierówności

Najnowsza wersja tego dokumentu dostępna jest pod adresem

http://wmii.uwm.edu.pl/~denisjuk/uwm

Zagadnienie

Scena



Czynności

- Przygotowanie specjalnej tekstury, mapy wektorów normalnych
  - GIMP, wtyczka normalmapping
- Przy obliczeniu oświetlenia pobieramy wektor normalny z tekstury
  - potrzebna jest baza wektorów stycznych do powierzchni
    - wektor styczny t i *drugi wektor styczny* b
    - $(r, g, b) \iff (x, y, z)$
  - współrzędne pobranego z tekstury wektora dane są w układzie (t, n, b)
  - 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: $b(u,v) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$,

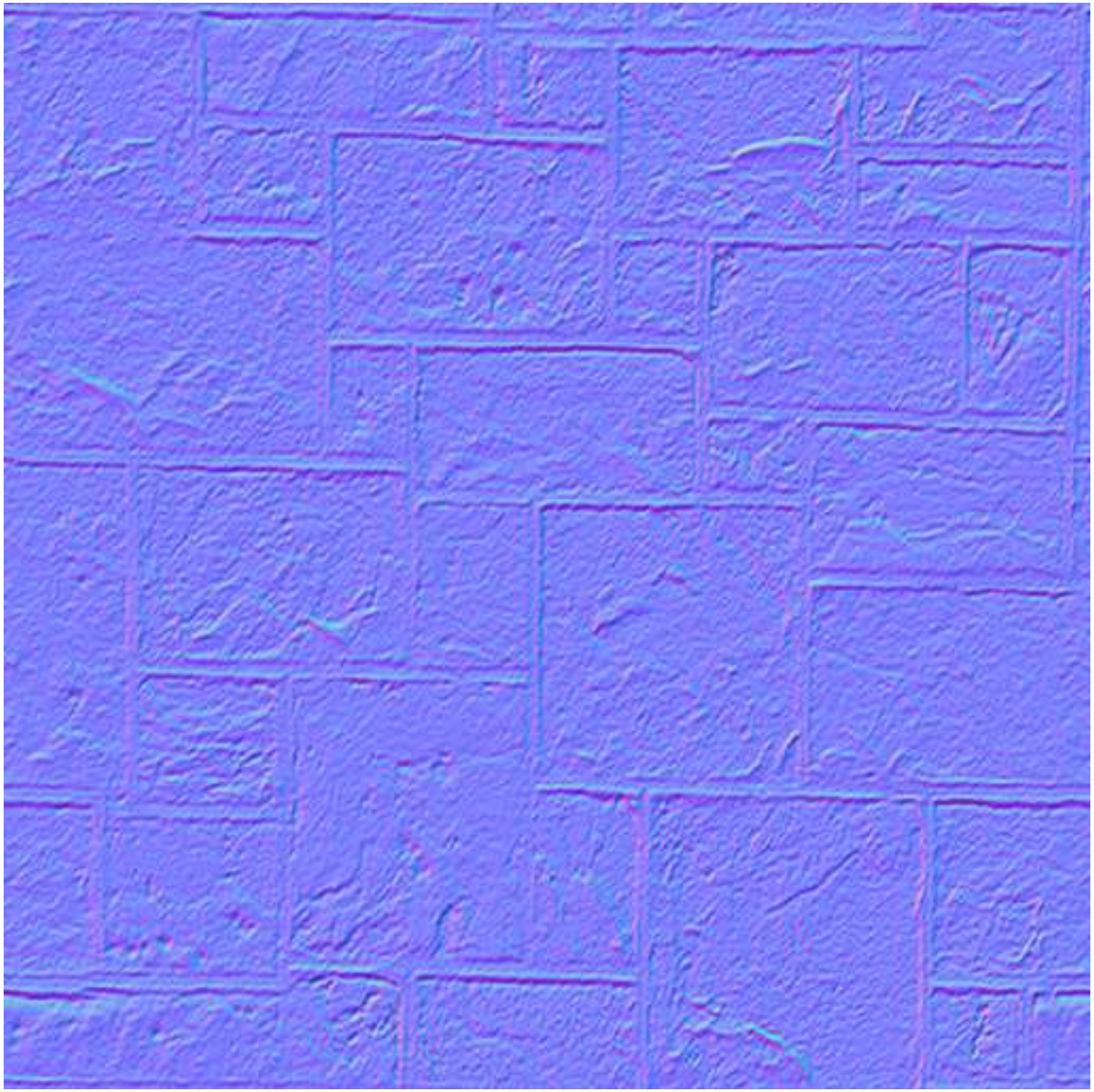
Uwaga o drugim wektorze stycznym

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

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

Tekstura

Mapa wektorów normalnych



Implementacja

Dane wejściowe


```
#version 430 core

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


```
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);
```

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, n, b)

- przeliczamy współrzędne kierunków do kamery i do światła

```
vec4 camera =
  -view_matrix*vec4(0.0, 0.0, 0.0, 1);
frag_vertex.view_dir
  = normalize(e_tbn*(camera.xyz-vertex.xyz));
frag_vertex.light_dir
  = normalize(e_tbn*frag_vertex.light_dir);
```

Obliczenia w shaderze fragmentów


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

- dalej wszystko jak wcześniej

Dodatkowy unit teksturowy


```
sampler2D color_texture;
sampler2D bump_texture;
```

Nowa klasa `BumpModel`


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

- `Tori` i `Plane` dziedziczą z tej klasy

Nowa struktura dla wierzchołków


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

Uzupełnienie inicjalizacji torusa


```
BumpTextureVertex * vertices
  =new BumpTextureVertex[(m_ + 1)*(n_ + 1)];

.............................

vertices[i*(m_ + 1)+j].tangent[0]=cos(theta);
vertices[i*(m_ + 1)+j].tangent[1]=0;
vertices[i*(m_ + 1)+j].tangent[2]=-sin(theta);
vertices[i*(m_ + 1)+j].bitangent[0]=
                          -sin(phi)*sin(theta);
vertices[i*(m_ + 1)+j].bitangent[1]=cos(phi);
vertices[i*(m_ + 1)+j].bitangent[2]=
                          cos(phi)*cos(theta);

.............................
```

Uzupełnienie inicjalizacji płaszczyzny


```
BumpTextureVertex* vertices =
  new BumpTextureVertex[(m_ + m_ + 1)
                        *(n_ + n_ +1)];

 ..............................

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 wyświetlenia


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

 ..............................
```

Nowa klasa `PointLightBumpProgram`


```
class PointLightBumpProgram :
        public PointLightProgram{
public:
  void Initialize(
    const char *vertex_shader_file,
    const char *fragment_shader_file);
  void SetBumpTextureUnit(GLuint);
private:
  GLuint bump_texture_unit_location_;
};
```

Zmiany w klasie `Window`

- Zmiany w klasie `Window` są oczywiste