Zaawansowane systemy programowania grafiki. Teksturowanie

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

3 marca 2021

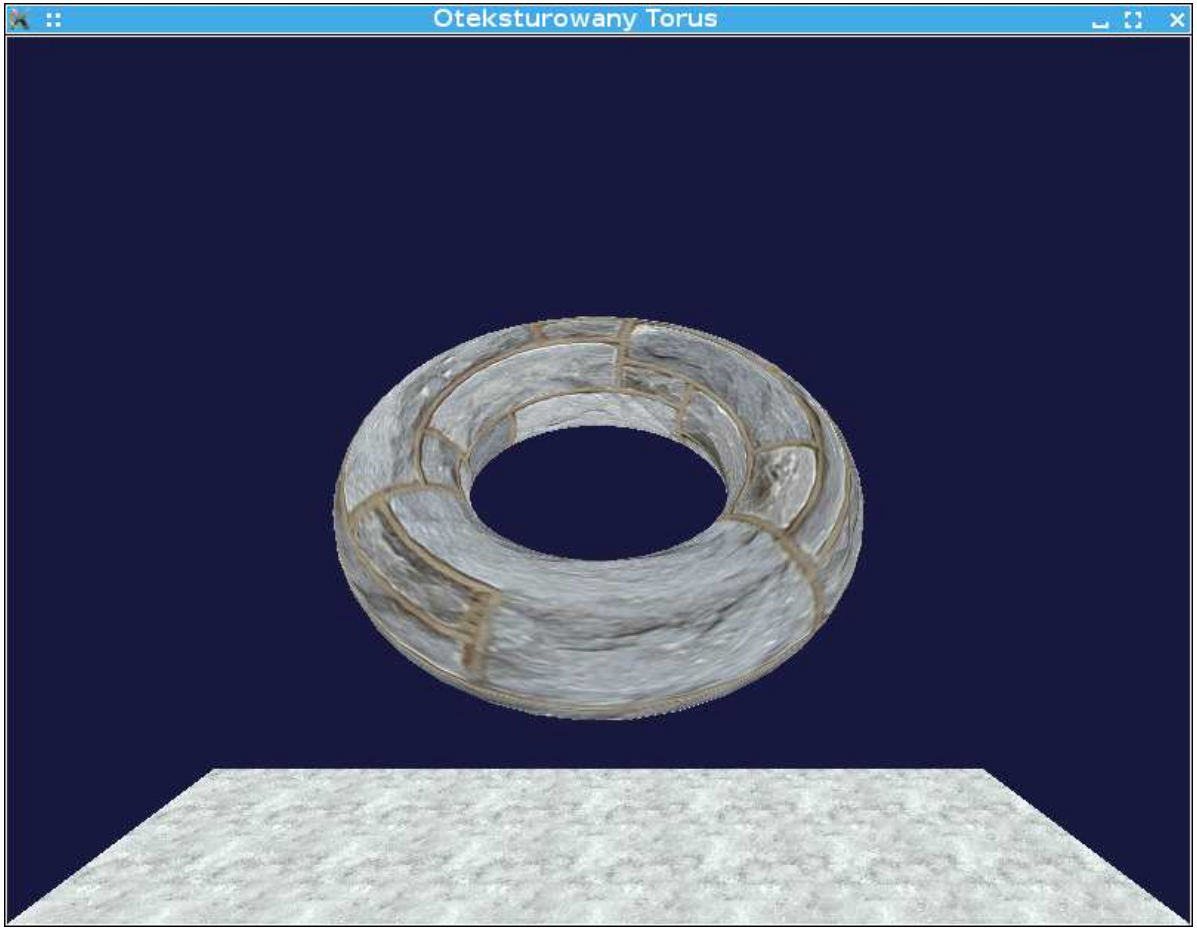
Teksturowanie

Najnowsza wersja tego dokumentu dostępna jest pod adresem

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

Modelowanie Torusa

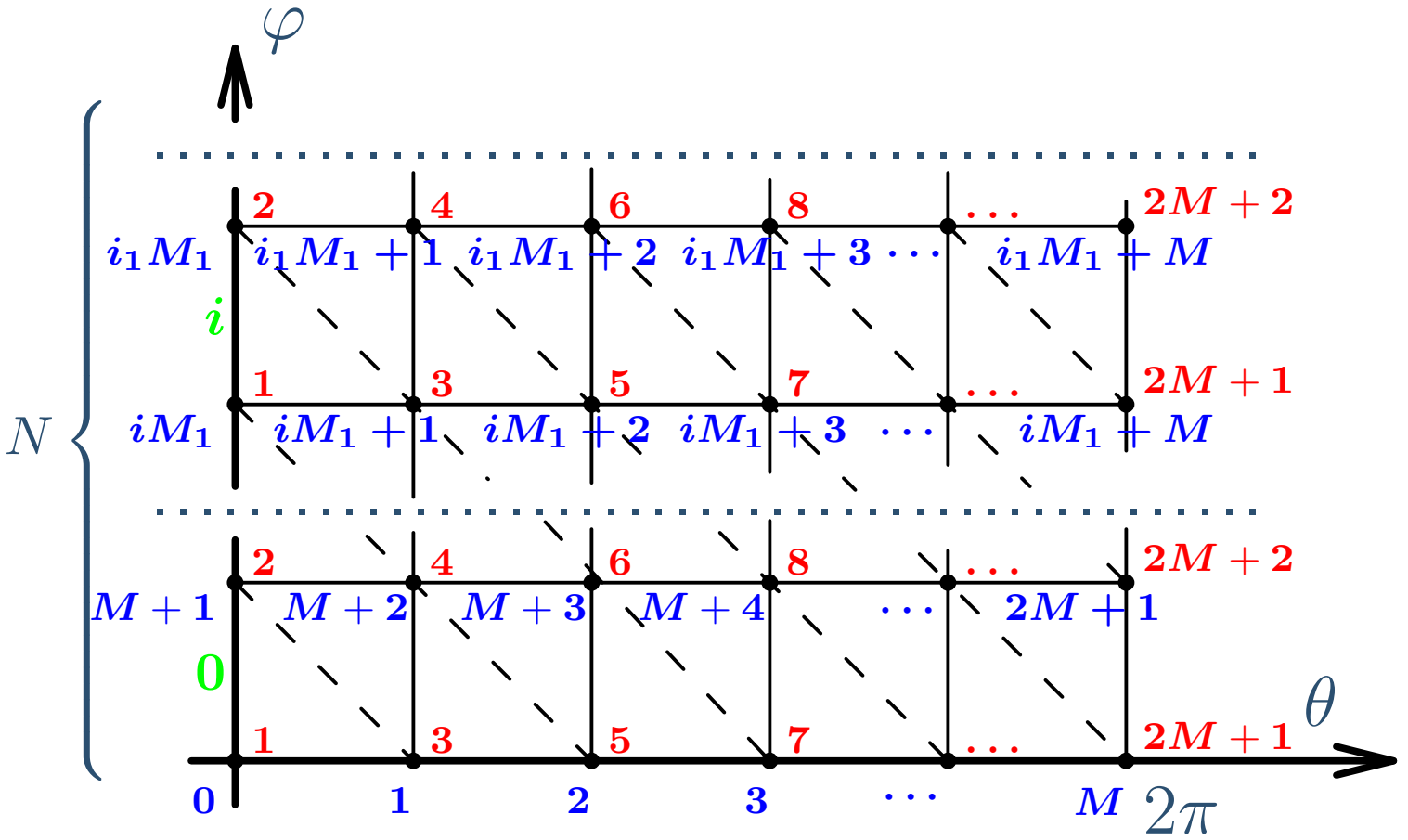
Torus otekstsurowany



Parametryzacja torusa

- $P(\theta, \varphi) = \begin{pmatrix} (R + r\cos\varphi)\sin\theta \\ r\sin\varphi \\ (R + r\cos\varphi)\cos\theta \end{pmatrix}$,
- $0 \leqslant \theta, \varphi \leqslant 2\pi$
- Mapowanie toroidalne: $\begin{cases} s = \frac{\theta}{2\pi}, \\ t = \frac{\varphi}{2\pi}. \end{cases}$
- Węzły: $\begin{cases} \theta_j = \frac{2\pi}{M} j, & j = 0, \ldots, M, \\ \varphi_i = \frac{2\pi}{N} i, & i = 0, \ldots, N. \end{cases}$

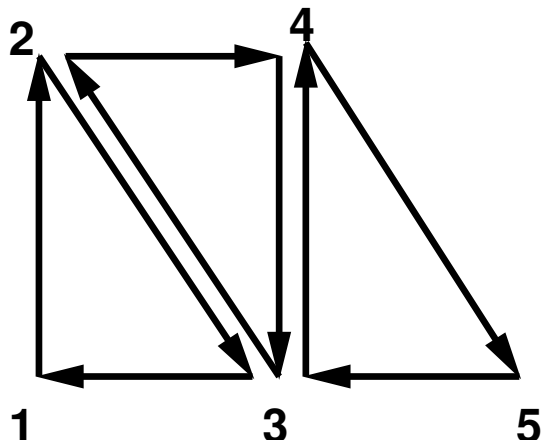
Triangulacja torusa (GL_TRIANGLE_STRIP)



- wierzchołki, $M_1 = M + 1$, $i_1 = i + 1$
- indeksy
- paski (GL_TRIANGLE_STRIP)

Niewidoczne ściany

- Ściany trójkątów, którę są wewnątrz torusa nigdy nie bedą widoczne
- Opcja `GL_CULL_FACE` pozwala na wyłączenie części ścian z renderowania
- Wyłączone ściany określa się przez orientację
  - `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`
  - `GL_CW`, `GL_CCW`



Przykładowy kod


```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
glFrontFace(GL_CW);

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

glDisable(GL_CULL_FACE);
```

Teksturowanie

Tesktura

- Obiekt tekstury na GPU
- Unit teksturowy na GPU
- Przykładowy obrazek:

Aktywizacja tekstury

- `texture` jest zmienną typu `GLuint`

```
// nowy obiekt teksturowy
glGenTextures(1, &texture);

// aktywacja
glBindTexture(GL_TEXTURE_2D, texture);
```

funkcja `glBindTexture`


```
void glBindTexture( enum target, uint texture );
```

- `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D`, `GL_TEXTURE_1D_ARRAY`, `GL_TEXTURE_2D_ARRAY`, `GL_TEXTURE_RECTANGLE`, `GL_TEXTURE_BUFFER`, `GL_TEXTURE_CUBE_MAP`, `GL_TEXTURE_2D_MULTISAMPLE`, oraz `GL_TEXTURE_2D_MULTISAMPLE_ARRAY`

Wysyłanie tekstury na GPU


```
void glTexImage2D( enum target, int level,
                   int internalformat,
                   sizei width, sizei height,
                   int border, enum format,
                   enum type, const void *data );
```

- `border`$= 0$
- `width`, `height` — w każdej implementacji ograniczenie do co najmniej 1024
- 2^k
- `format` — `GL_RED`, `GL_RG`, `GL_RGB`, `GL_BGR`, `GL_RGBA`, oraz `GL_BGRA`

Usuwanie tekstury z GPU


```
glDeleteTextures(1, &texture);
```

Intrapolacja i ekstrapolacja aktywnej tekstury


```
// parametry interpolacji tekstury
glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// parametry ekstrapolacji tekstury
glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Unit teksturowy

- w shaderze — zmienna uniform (typu `sampler2D`)

```
uniform sampler2D Texture;
```

- przypisać numer unitu (0, 1, etc), jak zwykłej zmiennej uniform
- aktywizacja unitu teksturowego i dowiązanie do niego obiektu tektury w programie

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
```

  - ten sam unit może działać z różnymi teksturami
  - zazwyczaj tego samego typu

- pobieranie teksela (w shaderze)

```
out_Color = texture(Texture, Texcoord);
```

  - gdzie `Texcoord` jest zmienną typu `vec2`

Mipmapping

- inaczej rezerwuje się miejsce w GPU
- przy wysyłaniu tekstury karta graficzna może utworzyć mipmapę

```
glBindTexture(GL_TEXTURE_2D, texture);

glTexStorage2D(GL_TEXTURE_2D, levels,
   internal_format, width, height);
glTexSubImage2D(GL_TEXTURE_2D, levels,
   x_offset, y_offset,
   width, height, format, type, image);
glGenerateMipmap(GL_TEXTURE_2D);
```

Przykład


```
glBindTexture(GL_TEXTURE_2D, texture);

glTexStorage2D(GL_TEXTURE_2D, 7, GL_RGBA,
               512, 512);
glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 512,
                512, GL_RGBA, GL_UNSIGNED_BYTE,
               (const GLvoid*)image);
glGenerateMipmap(GL_TEXTURE_2D);
```

Wykorzystanie mipmapy


```
glTexParameteri(GL_TEXTURE_2D,
      GL_TEXTURE_MIN_FILTER,
      GL_LINEAR_MIPMAP_LINEAR);
```

Nadpróbkowanie


```
glfwWindowHint(GLFW_SAMPLES, 8);
glEnable(GL_MULTISAMPLE);
```

Format TGA

- opracowany przez Truevision w 1986
- jest bardzo prostym formatem, bez patentów
- składa się z:
  - nagłówka
  - danych

Nagłówek TGA

1. (1 byte) długość ID
2. (1 byte) typ mapy kolorów
3. (1 byte) typ obrazka (2 — bez kompresji, bez mapy kolorów)
4. (5 bytes) specyfikacja mapy kolorów
5. (10 bytes) specyfikacja obrazka
   (a) (2 bytes) współrzędna x lewego dolnego rogu
   (b) (2 bytes) współrzędna y lewego dolnego rogu
   (c) (2 bytes) szerokość obrazka w pikselach
   (d) (2 bytes) wysokość obrazka w pikselach
   (e) (1 byte) bitów na pixel (24 — rgb, 32 — rgbα)
   (f) (1 byte)
      i. bity 3–0 określają kanał α (bitów na pixel),
      ii. bity 5–4 określają kierunek, (w prawo, w dół)
      iii. bity 8-7 są zerowe

Implementacja

Shader Wierzchołków


```
#version 430 core
layout(location=0) in vec4 in_position;
layout(location=2) in vec2 in_texture;

uniform mat4 model_matrix;
uniform mat4 view_matrix;
uniform mat4 projection_matrix;

out vec2 tex_coord;

void main(void){
  gl_Position = (projection_matrix * view_matrix
     * model_matrix) * in_position;
  tex_coord = in_texture;
}
```

Shader Fragmentów


```
#version 430 core

layout (location = 0) out vec4 color;

in vec2 tex_coord;

uniform sampler2D texture_unit;

void main(void){
    color = texture(texture_unit, tex_coord);
}
```

Nowe klasy C++

- `TextureModel` — model z teksturą
- `Torus` — torus
- `Plane` — płaszczyzna
- `TextureCameraProgram` — nowy program, z teksturą
- `Texture` — tekstura

Nowa struktura dla wierzchołków

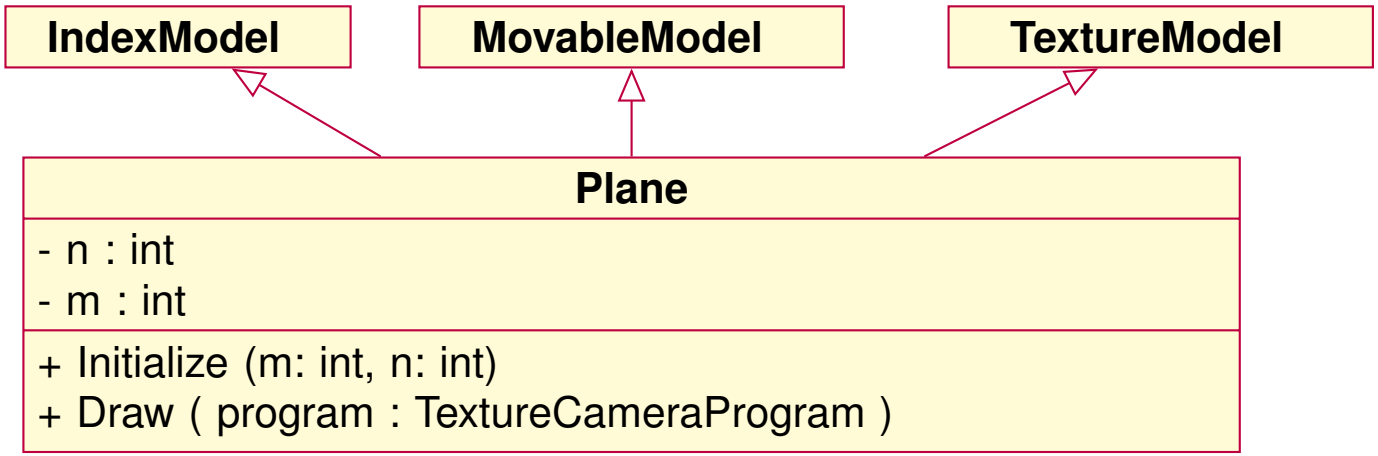

```
typedef struct TextureVertex{
    float position[4];
    float texture[2];
} TextureVertex;
```

Model z teksturą

TextureModel
texture_unit_ : GLuint # texture_ : GLuint
+ SetTexture (t: GLuint) + SetTextureUnit (t: GLuint)

```
class TextureModel{
public:
 void SetTextureUnit(GLuint t){texture_unit_=t;}
 void SetTexture(GLuint t){texture_ = t;}
protected:
 GLuint texture_unit_;
 GLuint texture_;
};
```

Klasa `Plane`




```
class Plane : public IndexModel,
    public MovableModel, public TextureModel{
public:
    void Initialize(int m, int n);
    void Draw(const Program & program) const;
private:
    int n_; // mesh parameter
    int m_; // mesh parameter
};
```

Inicjalizacja. Wierzchołki i indeksy


```
void Plane::Initialize(int m, int n){
  m_=m;
  n_=n;
  model_matrix_.Translate(0, -4, 0);
  TextureVertex vertices[4]={
    {{-(float)(m),0.0f,-(float)(n),1.0f},
                    {-(float)(m),-(float)(n)}},
    {{(float)(m),0.0f,-(float)(n),1.0f},
                    {(float)(m),-(float)(n)}},
    {{(float)(m),0.0f,(float)(n),1.0f},
                    {(float)(m),(float)(n)}},
    {{-(float)(m),0.0f,(float)(n),1.0f},
                    {-(float)(m),(float)(n)}}
  };
  GLuint indices[4]={ 0, 1, 3, 2};
```

Inicjalizacja. VAO i VBO


```
glGenVertexArrays(1, &vao_);
glBindVertexArray(vao_);
glGenBuffers(1, &vertex_buffer_);
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
    vertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE,
    sizeof(vertices[0]), (GLvoid*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
    sizeof(vertices[0]),
    (GLvoid*)sizeof(vertices[0].position));
glEnableVertexAttribArray(2);
glGenBuffers(1, &index_buffer_);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, index_buffer_);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,
  sizeof(indices), indices, GL_STATIC_DRAW);
```

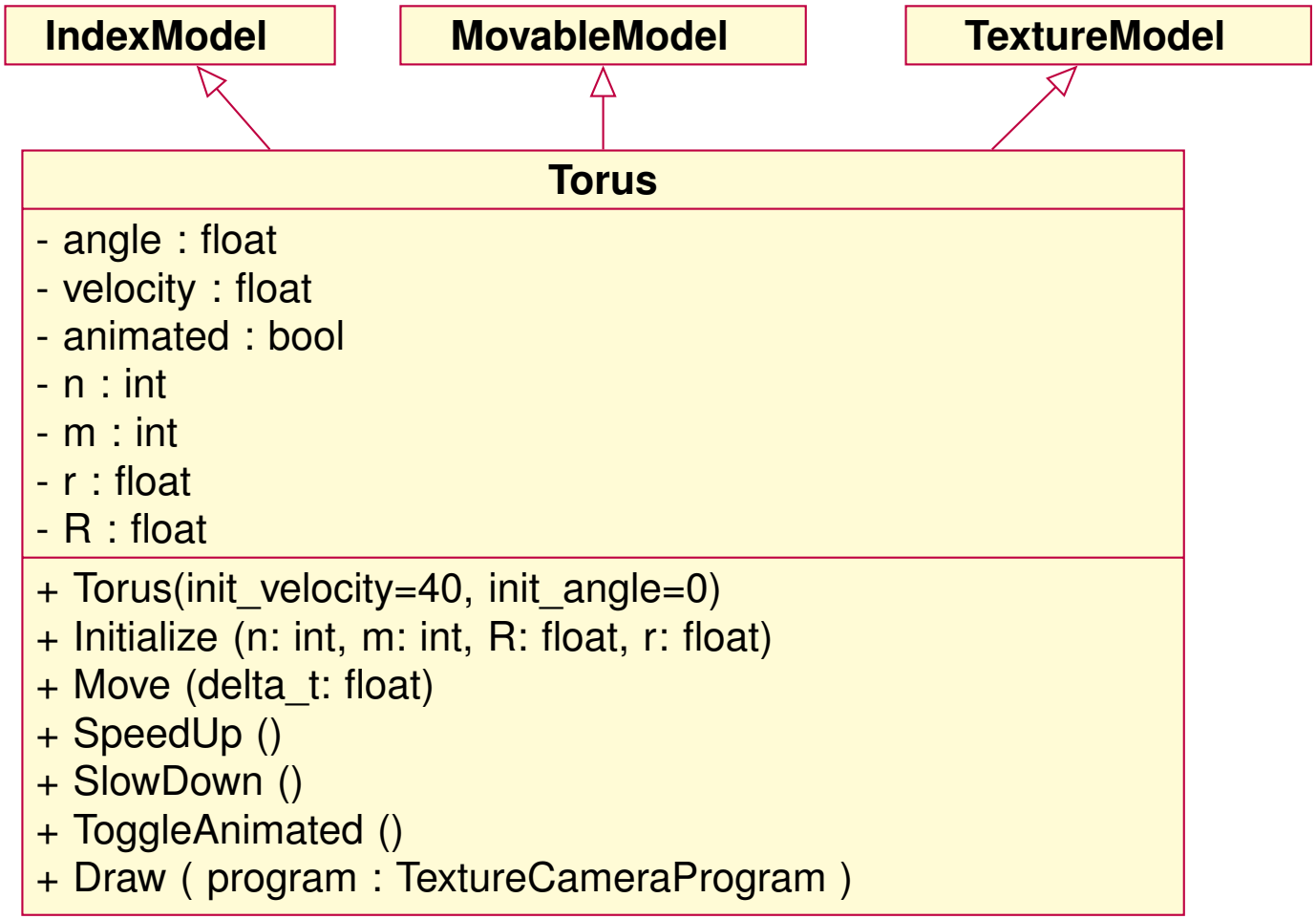
Wyświetlenie


```
void Plane::Draw(const Program & program) const{
  glBindVertexArray(vao_);
  glUseProgram(program);
  program.SetModelMatrix(model_matrix_);
  glActiveTexture(texture_unit_);
  glBindTexture(GL_TEXTURE_2D, texture_);

  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK);
  glFrontFace(GL_CW);
  glDrawElements(GL_TRIANGLE_STRIP, 4,
                 GL_UNSIGNED_INT, (GLvoid*)0);
  glDisable(GL_CULL_FACE);

  glBindVertexArray(0);
  glUseProgram(0);
}
```

Torus



Inicjalizacja. Wierzchołki


```
TextureVertex* vertices
      =new TextureVertex[(m_+1)*(n_+1)];
int i,j;
for (i=0;i<=n_;i++) {
 float phi=2*M_PI/(float)n_*i;
 for (j=0;j<=m_;j++){
  float theta=2*M_PI/(float)m_*j;
  vertices[i*(m_+1)+j].position[0]
        =(R  + r*cos(phi)) * sin(theta);
  vertices[i*(m_+1)+j].position[1]=r*sin(phi);
  vertices[i*(m_+1)+j].position[2]
        =(R + r*cos(phi))*cos(theta);
  vertices[i*(m_+1)+j].position[3]=1.0f;
  vertices[i*(m_+1)+j].texture[0]=(float)j/(float)m_;
  vertices[i*(m_+1)+j].texture[1]=(float)i/(float)n_;
 }
}
```

Inicjalizacja. Indeksy


```
GLuint * indices=new GLuint[2*n_*(m_ + 1)];

unsigned int k=0;

for(i=0; i<=n_ - 1; i++){
  for(j=0; j<=m_; j++){
    indices[2*(i*(m_ + 1)+j)]=k;
    indices[2*(i*(m_ + 1)+j)+1]=k+m_+1;
    k++;
  }
}
```

Inicjalizacja. VAO i VBO dla wierzchołków


```
glGenVertexArrays(1, &vao_);
glBindVertexArray(vao_);


glGenBuffers(1, &vertex_buffer_);
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_);
glBufferData(GL_ARRAY_BUFFER,
        (m_+1)*(n_+1)*sizeof(vertices[0]),
        vertices,
        GL_STATIC_DRAW);
```

Inicjalizacja. Argumenty shadera wierzchołków


```
glVertexAttribPointer(0, 4, GL_FLOAT,
  GL_FALSE, sizeof(vertices[0]), (GLvoid*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(2, 2, GL_FLOAT,
  GL_FALSE, sizeof(vertices[0]),
  (GLvoid*)sizeof(vertices[0].position));
glEnableVertexAttribArray(2);
```

- dla współrzędnych tekstury wykorzystywany argument 2
- argument 1 zostawmy dla kolorów

Inicjalizacja. VBO dla indeksów


```
glGenBuffers(1, &index_buffer_);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,
                                index_buffer_);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,
             (m_+1)*n_*2*sizeof(GLuint),
             indices,
             GL_STATIC_DRAW
             );
```

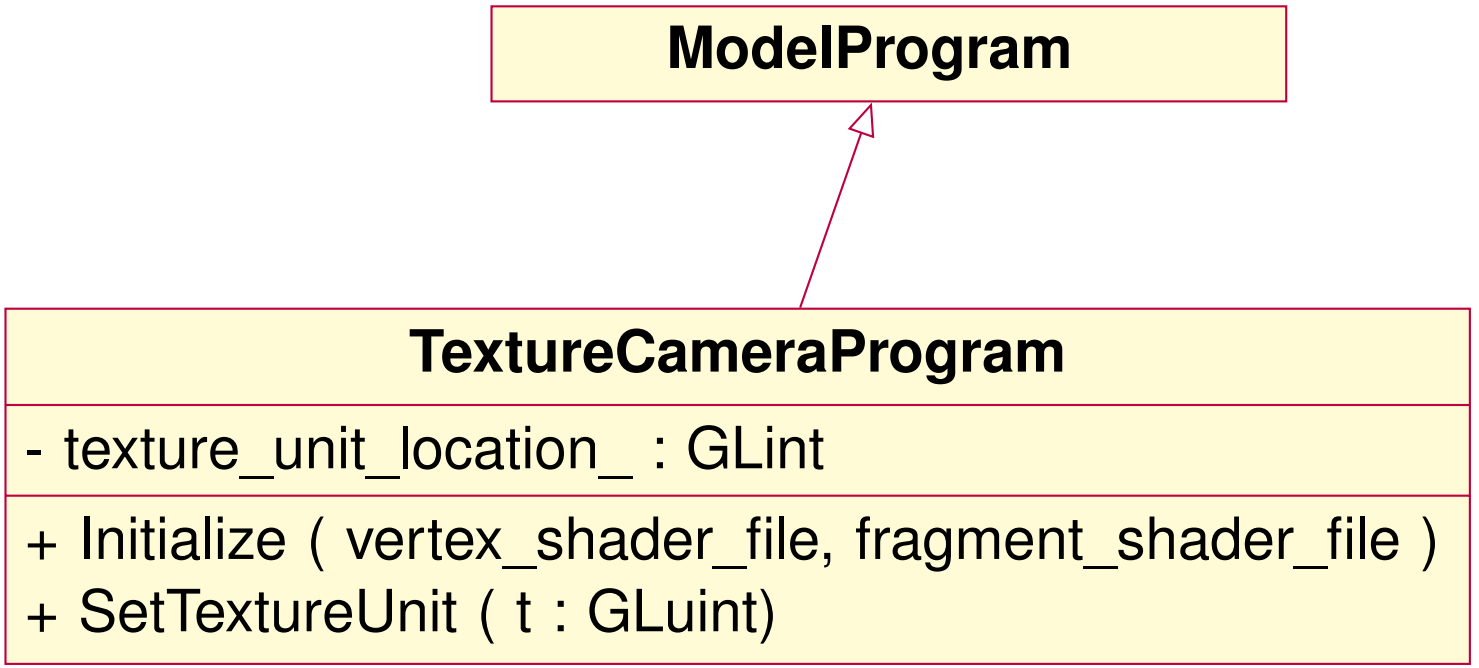
Inicjalizacja. Zakończenie


```
    delete [] vertices;
    delete [] indices;
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Wyświetlenie


```
void Torus::Draw(const Program &program) const{
  glBindVertexArray(vao_);
  glUseProgram(program);
  program.SetModelMatrix(model_matrix_);
  glActiveTexture(texture_unit_);
  glBindTexture(GL_TEXTURE_2D, texture_);
  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK);
  glFrontFace(GL_CW);
  for (int i=0;i<n_;i++){
    glDrawElements(GL_TRIANGLE_STRIP,
     2*(m_ + 1), GL_UNSIGNED_INT,
     (GLvoid*)(sizeof(GLuint)*2*i*(m_+1)));
  }
  glDisable(GL_CULL_FACE);
  glBindVertexArray(0);
  glUseProgram(0);
}
```

TextureCameraProgram



Inicjalizacja


```
void TextureCameraProgram::Initialize(
    const char *vertex_shader_file,
    const char *fragment_shader_file){
      ModelProgram::Initialize(
          vertex_shader_file,
          fragment_shader_file);
      texture_unit_location_
          = GetUniformLocationOrDie("texture_unit");
}
```

Zmienne uniform


```
void TextureCameraProgram::SetTextureUnit(
                                    GLuint t) const{
    glUniform1i(texture_unit_location_, t);
}
```

Klasa Texture


```
class Texture{
public:
    ~Texture();
    void Initialize(const char *filename);
    operator GLuint() const {return texture_;}
private:
    GLuint texture_;
    void LoadTGAFileOrDie(GLenum target,
                          const char  * filename);
};
```

Inicjalizacja


```
void Texture::Initialize(const char * filename){
  glGenTextures(1, &texture_);
  // aktywacja
  glBindTexture(GL_TEXTURE_2D, texture_);
  LoadTGAFileOrDie(GL_TEXTURE_2D, filename);
  // parametry interpolacji tekstury
  glTexParameteri(GL_TEXTURE_2D,
          GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D,
          GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  // parametry ekstrapolacji tekstury
  glTexParameteri(GL_TEXTURE_2D,
          GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D,
          GL_TEXTURE_WRAP_T, GL_REPEAT);
  glBindTexture(GL_TEXTURE_2D, 0);
}
```

Destruktor


```
Texture::~Texture(){
    glDeleteTextures(1, &texture_);
}
```

Wczytywanie z pliku. Struktura nagłówka TGA


```
typedef struct TGAHeader{
  uint8_t  idlength;
  uint8_t  colormap;
  uint8_t  datatype;
  uint8_t  colormapinfo[5];
  uint16_t xorigin;
  uint16_t yorigin;
  uint16_t width;
  uint16_t height;
  uint8_t  bitperpel;
  uint8_t  description;
} TGAHeader;
```

LoadTGAFileOrDie. Zmienne


```
void Texture::LoadTGAFileOrDie(GLenum target,
                               const char  * filename){
  TGAHeader*  header;
  GLchar*     buffer;
  size_t      file_size;
  GLuint format;
  GLuint internal_format;
```

LoadTGAFileOrDie. Wczytywanie pliku


```
ifstream file (filename,
        ios::in|ios::ate|ios::binary);
if (file.is_open()) {
    file_size = file.tellg();
    buffer = new GLchar [file_size];
    file.seekg (0, ios::beg);
    file.read (buffer, file_size);
    file.close();
}
else{ //file was not opened
    cerr<<"Could not open the texture file "
        << filename << endl;
    exit(EXIT_FAILURE);
}
```

LoadTGAFileOrDie. Testowanie danych


```
if (file_size <= sizeof(TGAHeader)) {
  cerr<<"Too small texture file "
                        <<filename<<endl;
  delete [] buffer;
  glfwTerminate();
  exit(EXIT_FAILURE);
}
header = (TGAHeader*)buffer;
// tylko nieskompresowany RGB lub RGBA obraz
if (header->datatype != 2
  || (header->bitperpel != 24
      && header->bitperpel != 32)){
    cerr<<"Wrong TGA format "<<filename<<endl;
    delete [] buffer;
  glfwTerminate();
    exit(EXIT_FAILURE);
}
```

`LoadTGAFileOrDie`. Wysyłanie danych i zakończenie


```
format =
  (header->bitperpel == 24 ? GL_BGR : GL_BGRA);
internal_format =
  (format == GL_BGR ? GL_RGB8 : GL_RGBA8);

glTexImage2D(target, 0, internal_format,
   header->width, header->height, 0, format,
   GL_UNSIGNED_BYTE,
   (const GLvoid*)(buffer
       + sizeof(TGAHeader) + header->idlength));

delete [] buffer;
```

Uzupełnienie klasy `Window`


```
private:
    Torus torus_;
    Plane plane_;
    Texture color_texture_;
    void InitTextures();

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

Stałe w pliku `window.h`


```
const char* kVertexShader="TextureShader.vertex.g
const char* kFragmentShader="TextureShader.fragme
const char* kColorTextureFile="texture.tga";
const char* kIceTextureFile="ice.tga";
const int kPlaneM=10;
const int kPlaneN=10;
const int kTorusM=32;
const int kTorusN=32;
const float kTorusR=2.5f;
const float kTorusr=0.5f;
```

Uzupełnienie Inicjalizacji


```
InitTextures();

InitModels();

InitPrograms();
```

InitTextures


```
void Window::InitTextures(){
    color_texture_.Initialize(kColorTextureFile);
    ice_texture_.Initialize(kIceTextureFile);
}
```

InitModels


```
void Window::InitModels(){
   torus_.Initialize(kTorusN, kTorusM,
                     kTorusR, kTorusr);
   torus_.SetTexture(color_texture_);
   torus_.SetTextureUnit(GL_TEXTURE0);

   plane_.Initialize(kPlaneM, kPlaneN);
plane_.SetTexture(ice_texture_);
plane_.SetTextureUnit(GL_TEXTURE0);
}
```

InitPrograms


```
void Window::InitPrograms(){
    program_.Initialize(kVertexShader,
                        kFragmentShader);
    program_.SetTextureUnit(0);
}
```

Render


```
void Window::Run(void){
 while (!glfwWindowShouldClose(window_)){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_B

  clock_t now = clock();
  if (last_time_ == 0) last_time_ = now;
  torus_.Move((float)(now - last_time_)
                              / CLOCKS_PER_SEC );
  last_time_ = now;

  torus_.Draw(program_);
  plane_.Draw(program_);

  glfwSwapBuffers(window_);
  glfwPollEvents();
 }
}
```