

Zaawansowane systemy programowania grafiki. Mapowanie cienia (Shadow Map)

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

31 marca 2021

Mapowanie cienia (Shadow Map)

Algorytm

Obiekt Buforu Ramki

Implementacja

Najnowsza wersja tego dokumentu dostępna jest pod adresem

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

Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja

Algorytm

Mapowanie cienia

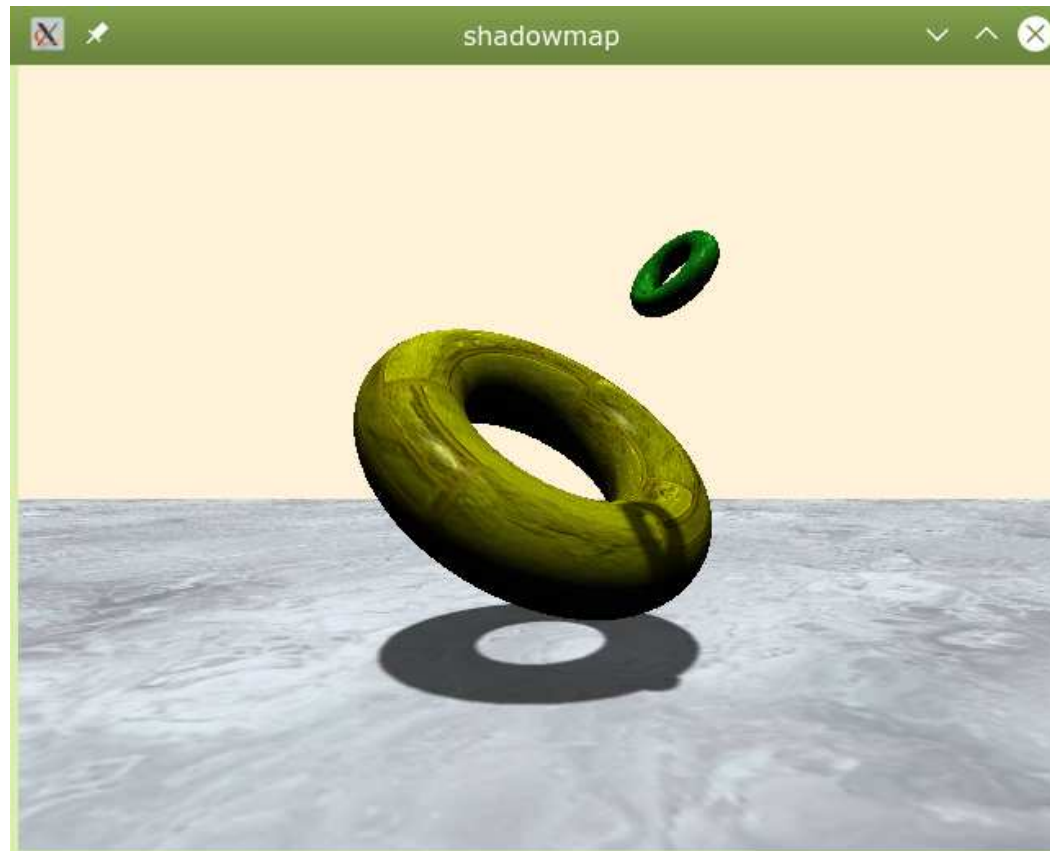
Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja

- Lance Williams (1978). “Casting curved shadows on curved surfaces”



- Zacienione fragmenty to są te, które nie widoczne, jeżeli patrzeć ze źródła światła

Algorytm

Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja

- Wyrenderować scenę, używając źródła światła jako kamery
 - ◆ Dla każdego fragmentu zapisać wartość buforu głębokości (do tekstury)
- Wyrenderować scenę ze zwykłej kamery
 - ◆ Dla każdego fragmentu porównać odległość do światła z zapisaną wartością

Czynności

Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja

- Utworzyć nowy obiekt buforu ramki (*framebuffer object, FBO*)
 - ◆ Utworzyć teksturę `GL_SHADOW_MAP` (mapę cienia) i przekierować wyjście z FBO do tekstury
 - ◆ Opracować tylko głębokość (ani tekstury, ani oświetlenie)
- Wyrenderować scene z wykorzystaniem mapy cienia
 - ◆ w mapie cienia będzie zapisana odległość od światła jako współrzędna z w układzie związanym ze światłem
 - ◆ potrzebna macierz przejścia do tego układu współrzędnych
- Światło kierunkowe:
 - ◆ projekcja postopadła

Antyaliasing

Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja

- Wygładzanie krawędzi cienia
- Uśrednienie

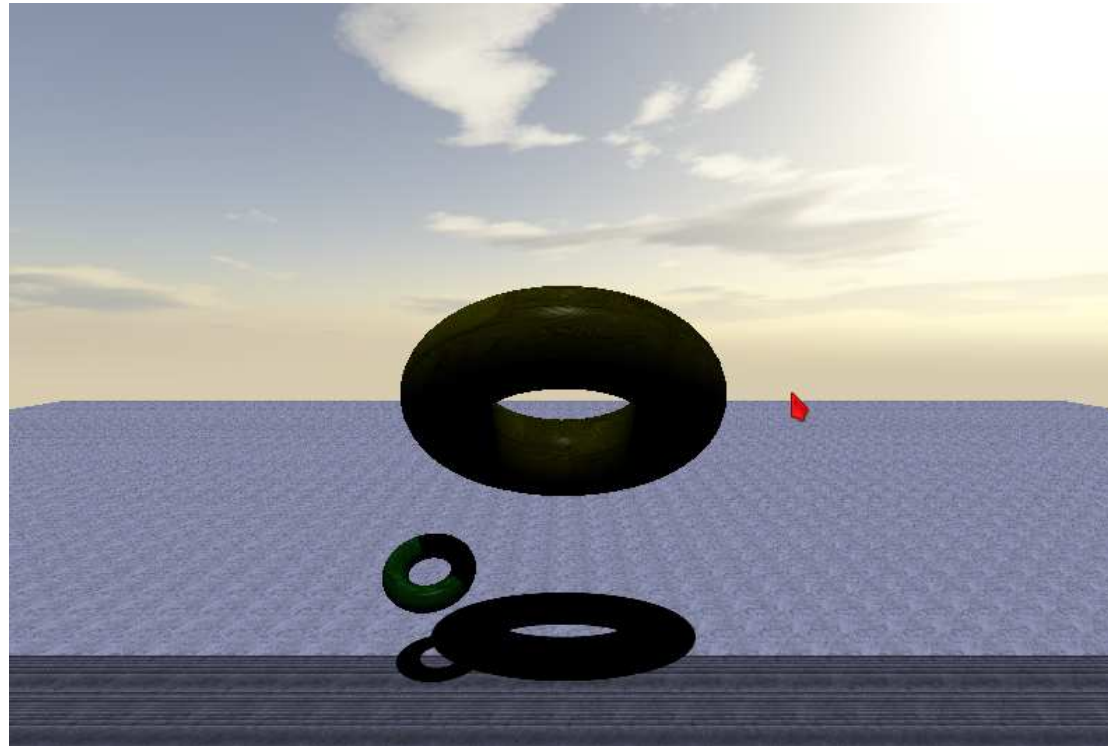
Z-fighting

Algorytm

❖ Mapowanie cienia

Obiekt Buforu Ramki

Implementacja



- wykorzystanie `glPolygonOffset`

```
void glPolygonOffset(GLfloat factor,  
                     GLfloat units);
```

$\text{factor} \cdot DZ + r_{\text{units}}$

Algorytm

Obiekt Buforu Ramki

- ❖ Framebuffer
- ❖ Tekstury
- ❖ Ustawienia
- ❖ Renderowanie

Implementacja

Obiekt Buforu Ramki

Utworzenie obiektu buforu ramki

Algorytm

Obiekt Buforu Ramki

❖ **Framebuffer**

❖ Tekstury

❖ Ustawienia

❖ Renderowanie

Implementacja

```
GLuint framebuffer = 0;  
glGenFramebuffers(1, &framebuffer);  
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
```

- domyślnie obiekt buforu ramki z numerem 0 — ekran

Utworzenie tekstury dla obrazu

Algorytm

Obiekt Buforu Ramki

❖ Framebuffer

❖ Tekstury

❖ Ustawienia

❖ Renderowanie

Implementacja

```
GLuint rendered_texture;  
glGenTextures(1, &rendered_texture);  
glBindTexture(GL_TEXTURE_2D, rendered_texture);
```

```
// Pusty obrazek (ostatnie zero)  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 1024,  
             768, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
```

```
// Parametry interpolacji (koniecznie)  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MAG_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D,  
                GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

- Parametry tekstury mogą być inne

Utworzenie tekstury dla buforu głębokości

Algorytm

Obiekt Buforu Ramki

❖ Framebuffer

❖ Tekstury

❖ Ustawienia

❖ Renderowanie

Implementacja

```
GLuint depthrenderbuffer;  
glGenRenderbuffers(1, &depthrenderbuffer);  
glBindRenderbuffer(GL_RENDERBUFFER,  
                    depthrenderbuffer);  
glRenderbufferStorage(GL_RENDERBUFFER,  
                      GL_DEPTH_COMPONENT, 1024, 768);  
glFramebufferRenderbuffer(GL_FRAMEBUFFER,  
                           GL_DEPTH_ATTACHMENT,  
                           GL_RENDERBUFFER,  
                           depthrenderbuffer);
```

- Można utworzyć tylko jedną z tekstur

Konfiguracja buforu ramki

Algorytm

Obiekt Buforu Ramki

❖ Framebuffer

❖ Tekstury

❖ Ustawienia

❖ Renderowanie

Implementacja

```
// Set "renderedTexture" as our colour attachment  
glFramebufferTexture(GL_FRAMEBUFFER,  
                     GL_COLOR_ATTACHMENT0,  
                     rendered_texture, 0);
```

```
// Set the list of draw buffers.  
GLenum draw_buffers[1] = {GL_COLOR_ATTACHMENT0};  
glDrawBuffers(1, draw_buffers);
```

- 1 — to ilość buforów wejściowych
 - ◆ może ich być więcej
 - ◆ w shaderze fragmentów:
`layout(location = 0) out vec3 color;`
 - ◆ `location = 0 \iff GL_COLOR_ATTACHMENT0`

Sprawdzenie buforu ramki

Algorytm

Obiekt Buforu Ramki

❖ Framebuffer

❖ Tekstury

❖ **Ustawienia**

❖ Renderowanie

Implementacja

```
if (glCheckFramebufferStatus (GL_FRAMEBUFFER)  
    != GL_FRAMEBUFFER_COMPLETE)  
    return false;
```

Renderowanie

Algorytm

Obiekt Buforu Ramki

❖ Framebuffer

❖ Tekstury

❖ Ustawienia

❖ **Renderowanie**

Implementacja

```
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);  
glViewport(0, 0, 1024, 768);  
  
...  
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

Implementacja

ShadowShader.vertex.glsl

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
#version 430 core
```

```
layout(location=0) in vec4 in_position;
```

```
uniform mat4 model_matrix;
```

```
uniform mat4 light_transform_matrix;
```

```
void main(void) {
```

```
    gl_Position = (light_transform_matrix  
        * model_matrix) * in_position;
```

```
}
```

- `light_transform_matrix = projection_matrix * view_matrix` jest obliczana w programie

ShadowShader.fragment.glsl

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
#version 430 core
```

```
void main(void) {  
}
```

Uzupełnienie *SunLightShader.vertex*

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
uniform mat4 light_transform_matrix;
```

```
out Vertex {  
    vec2    texcoord;  
    vec3    normal;  
    vec3    light_dir;  
    vec3    view_dir;  
    vec4    smcoord; //shadow map coordinates  
} frag_vertex;
```

```
frag_vertex.smcoord = light_transform_matrix  
    * model_matrix * in_position;
```

Uzupełnienie *SunLightShader.fragment*

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
in Vertex {  
    vec2 texcoord;  
    vec3 normal;  
    vec3 light_dir;  
    vec3 view_dir;  
    vec4 smcoord; //shadow map coordinates  
} frag_vertex;  
  
uniform sampler2DShadow depth_texture;
```

Antyaliasing

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
float PCF(in vec4 coord) {  
    float res = 0.0;  
    res += textureProjOffset(depth_texture,  
                             coord, ivec2(-1,-1));  
    res += textureProjOffset(depth_texture,  
                             coord, ivec2( 0,-1));  
  
    .....  
  
    res += textureProjOffset(depth_texture,  
                             coord, ivec2( 0, 1));  
    res += textureProjOffset(depth_texture,  
                             coord, ivec2( 1, 1));  
    return (1.0 - 0.6667*(1 - res / 9.0));  
}
```

W funkcji main

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

- współczynnik cienia

```
float shadow = PCF(frag_vertex.smcoord);
```

- ostatecznie

```
color *= texture(material.color_texture,  
frag_vertex.texcoord) * shadow;
```

Uzupełnienie klasy Window

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow SunLightProgram

❖ matma.h

- Program do obliczenia mapy cienia

```
ShadowProgram shadow_program_;
```

- Obiekt bufora ramki

```
Framebuffer depth_frame_buffer_;
```

- Obiekt mapy cienia

```
ShadowTexture shadow_texture_;
```

- Stała rozmiaru mapy cienia

```
const unsigned int  
    kShadowTextureSize = 2048;
```

Uzupełnienie `InitTextures`

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
shadow_texture_.Initialize(kShadowTextureSize,  
                           kShadowTextureSize);
```


Uzupełnienie `InitPrograms`

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
sun_program_.SetLightTransform(kSunLight);  
sun_program_.SetDepthTextureUnit(2);
```

```
shadow_program_.Initialize(kShadowVertexShader,  
                           kShadowFragmentShader);  
glUseProgram(shadow_program_);  
shadow_program_.SetSunLightTransform(kSunLight);
```

Uzupełnienie inicjalizacji

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
depth_frame_buffer_.Initialize(depth_texture_);
```

Renderowanie cienia

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer
❖ ShadowTexture
❖ ShadowProgram
❖ Shadow
SunLightProgram
❖ matma.h

```
glBindFramebuffer(GL_FRAMEBUFFER,  
                  depth_frame_buffer_);
```

```
glViewport(0, 0, kShadowTextureSize,  
           kShadowTextureSize);
```

```
glColorMask(GL_FALSE, GL_FALSE,  
            GL_FALSE, GL_FALSE);
```

```
glDepthMask(GL_TRUE);
```

```
glClear(GL_DEPTH_BUFFER_BIT);
```

```
glEnable      ( GL_POLYGON_OFFSET_FILL );
```

```
glPolygonOffset (2, 1);
```

```
plane_.drawShadow(shadow_program_);
```

```
tori_.drawShadow(shadow_program_);
```

```
glDisable     ( GL_POLYGON_OFFSET_FILL );
```

Renderowanie sceny

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);  
glViewport(0, 0, width(), height());  
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);  
glClear(GL_COLOR_BUFFER_BIT  
        | GL_DEPTH_BUFFER_BIT);
```

```
tori_.Draw(sun_program);  
plane_.Draw(sun_program);
```

Obiekt bufora ramki

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ **Framebuffer**

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
class FrameBuffer{
public:
    ~FrameBuffer();
    void Initialize(GLuint texture);
    operator GLuint(){return fbo_;}
private:
    GLuint fbo_;
};
```

Inicjalizacja

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ **Framebuffer**

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
void FrameBuffer::Initialize(GLuint texture) {  
    GLuint fbo_status;
```

```
    // Obiekt buforu ramki
```

```
    glGenFramebuffers(1, &fbo_);
```

```
    glBindFramebuffer(GL_FRAMEBUFFER, fbo_);
```

```
    glDrawBuffer(GL_NONE);
```

```
    glReadBuffer(GL_NONE);
```

```
    glFramebufferTexture2D(GL_FRAMEBUFFER,  
                           GL_DEPTH_ATTACHMENT,  
                           GL_TEXTURE_2D,  
                           texture, 0);
```

Sprawdzamy wynik inicjalizacji

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
// sprawdzamy FBO
if ((fbo_status
    = glCheckFramebufferStatus(GL_FRAMEBUFFER))
    != GL_FRAMEBUFFER_COMPLETE) {
    std::cerr << "error"
               << fbo_status<<std::endl;
    glfwTerminate();
    exit(EXIT_FAILURE);
}

glBindFramebuffer(GL_FRAMEBUFFER, 0);
}
```

Destruktor

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
Framebuffer::~Framebuffer() {  
    glBindFramebuffer(GL_FRAMEBUFFER, 0);  
    glDeleteFramebuffers(1, &fbo_);  
}
```


Mapa cienia

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
class ShadowTexture : public Texture{  
public:  
    void Initialize(int width, int height);  
};
```

Inicjalizacja

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
void ShadowTexture::Initialize(int width,  
                                int height) {
```

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

```
    // aktywacja  
    glActiveTexture(GL_TEXTURE2);  
    glBindTexture(GL_TEXTURE_2D, texture_);
```

Interpolacja i ekstrapolacja

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow SunLightProgram

❖ matma.h

```
// 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);
```

Wykorzystanie tekstury jako mapy cienia

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ **ShadowTexture**

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

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

Utworzenie nowej pustej tekstury

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow SunLightProgram

❖ matma.h

```
glTexImage2D(GL_TEXTURE_2D,  
             0, GL_DEPTH_COMPONENT,  
             width, height,  
             0, GL_DEPTH_COMPONENT,  
             GL_FLOAT, NULL);  
}
```

Klasa ShadowProgram

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

BaseProgram



ShadowProgram

light_transform_matrix_location_ : GLint

model_matrix_location_ : GLint

+ Initialize (vertex_shader_file, fragment_shader_file)

+ SetSunLightTransform (light : SunLight)

+ SetModelMatrix (matrix : Mat4)

```
class ShadowProgram : public BaseProgram{
public:
    void Initialize(const char* vertex_shader_file,
                   const char* fragment_shader_file);
    void SetSunLightTransform(const SunLight &light);
    void SetModelMatrix(Mat4 & matrix);
private:
    GLint light_tranform_matrix_location_;
    GLint model_matrix_location_;
};
```

Macierz widzenia „od światła”

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow SunLightProgram

❖ matma.h

```
void ShadowProgram::SetSunLightTransform
    (const SunLight &light){
    Mat4 light_transform_matrix;
    light_transform_matrix =
        Mat4::CreateSunLightTransformMatrix(light);

    glUniformMatrix4fv(
        light_transform_matrix_location_,
        1, GL_FALSE, light_transform_matrix);
}
```

Klasa ShadowSunLightProgram

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

SunLightProgram



ShadowSunLightProgram

light_transform_matrix_location_ : GLint

depth_texture_unit_location_ : GLint

+ Initialize (vertex_shader_file, fragment_shader_file)

+ SetSunLightTransform (light : SunLight)

+ SetModelMatrix (matrix : Mat4)

```
class ShadowSunLightProgram : public SunLightProgram{
public:
    void SetSunLightTransform(const SunLight & light)
    void Initialize(const char *vertex_shader_file,
                   const char *fragment_shader_file);
    void SetDepthTextureUnit(GLint t);
private:
    GLint light_transform_matrix_location_;
    GLint depth_texture_unit_location_;
};
```


Macierz widzenia „od światła”

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
void ShadowSunLightProgram::SetSunLightTransform  
    (const SunLight & light){  
    Mat4 light_transform_matrix;  
    light_transform_matrix =  
        Mat4::CreateBiasSunLightTransformMatrix(light);  
  
    glUniformMatrix4fv(  
        light_transform_matrix_location_,  
        1, GL_FALSE, light_transform_matrix);  
}
```

CreateSunLightTransformMatrix

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow

SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow

SunLightProgram

❖ matma.h

```
Mat4 Mat4::CreateSunLightTransformMatrix  
    (const SunLight & light, GLfloat s){
```

```
    GLfloat x, y, z, r;
```

```
    GLfloat theta_x, theta_y;
```

```
    Mat4 view_matrix;
```

```
    Mat4 projection_matrix;
```

```
    r = sqrtf(light.position[0]*light.position[0]  
            +light.position[1]*light.position[1]  
            +light.position[2]*light.position[2]);
```

```
    x = light.position[0]/r;
```

```
    y = light.position[1]/r;
```

```
    z = light.position[2]/r;
```

Obrót i rzutowanie

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow SunLightProgram

❖ matma.h

```
theta_x =  
    -asinf(y) * 180.0f / static_cast<GLfloat>(M_PI);  
theta_y =  
    atan2f(x, z) * 180.0f / static_cast<GLfloat>(M_PI);  
  
view_matrix.RotateAboutY(-theta_y);  
view_matrix.RotateAboutX(-theta_x);  
  
projection_matrix =  
    Mat4::CreateOrthoProjectionMatrix(-s, s,  
    -s, s, -s, s);  
  
view_matrix.MultiplyBy(projection_matrix);  
  
return view_matrix;
```

bias_matrix

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
Mat4 Mat4::CreateBiasMatrix() {  
    Mat4 out;  
    out.matrix_[0]=0.5f;  
    out.matrix_[5]=0.5f;  
    out.matrix_[10]=0.5f;  
    out.matrix_[12]=0.5f;  
    out.matrix_[13]=0.5f;  
    out.matrix_[14]=0.5f;  
  
    return out;  
}
```

Macierz projekcji prostopadłej

Algorytm

Obiekt Buforu Ramki

Implementacja

❖ Shadery dla cienia

❖ Shadow
SunLightShader

❖ Window

❖ Framebuffer

❖ ShadowTexture

❖ ShadowProgram

❖ Shadow
SunLightProgram

❖ matma.h

```
Mat4 Mat4::CreateOrthoProjectionMatrix(
    GLfloat left, GLfloat right,
    GLfloat bottom, GLfloat top,
    GLfloat near_plane, GLfloat far_plane){
    Mat4 out;
    GLfloat tx = -(right + left) / (right - left);
    GLfloat ty = -(top + bottom) / (top - bottom);
    GLfloat tz = -(far_plane + near_plane)
                / (far_plane - near_plane);
    out.matrix_[0]=2 / (right - left);
    out.matrix_[5]=2 / (top - bottom);
    out.matrix_[10]=-2 / (far_plane - near_plane);
    out.matrix_[12]=tx;
    out.matrix_[13]=ty;
    out.matrix_[14]=tz;
    return out;
}
```