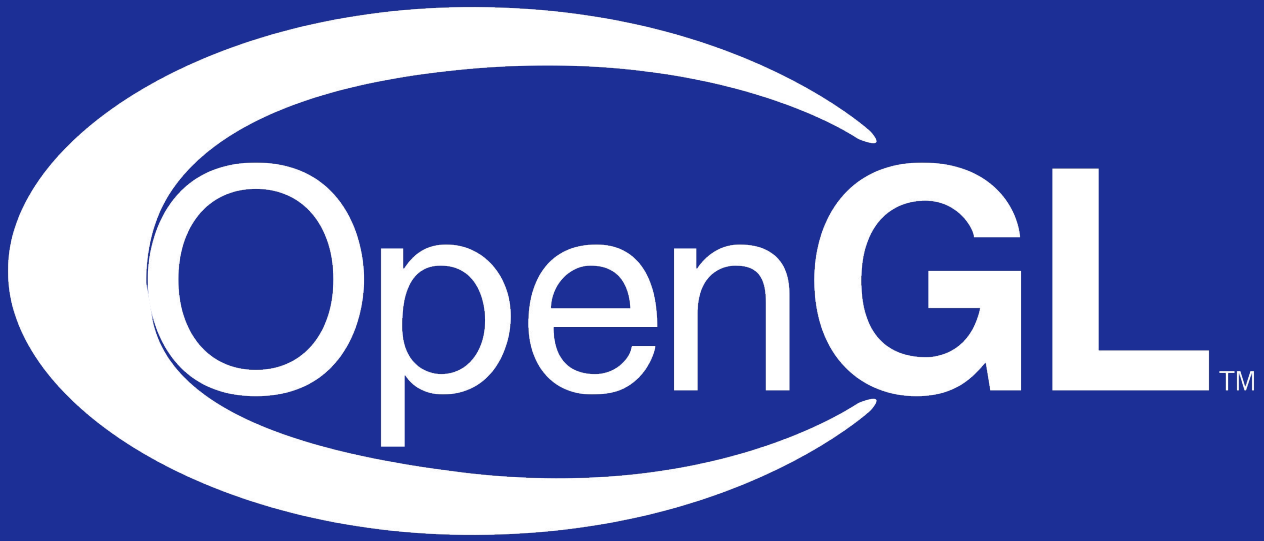
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Gry Komputerowe
Laboratorium 5

Oświetlenie
Head-up Display



Wydział
Informatyki

mgr inż. Michał Chwesiuk

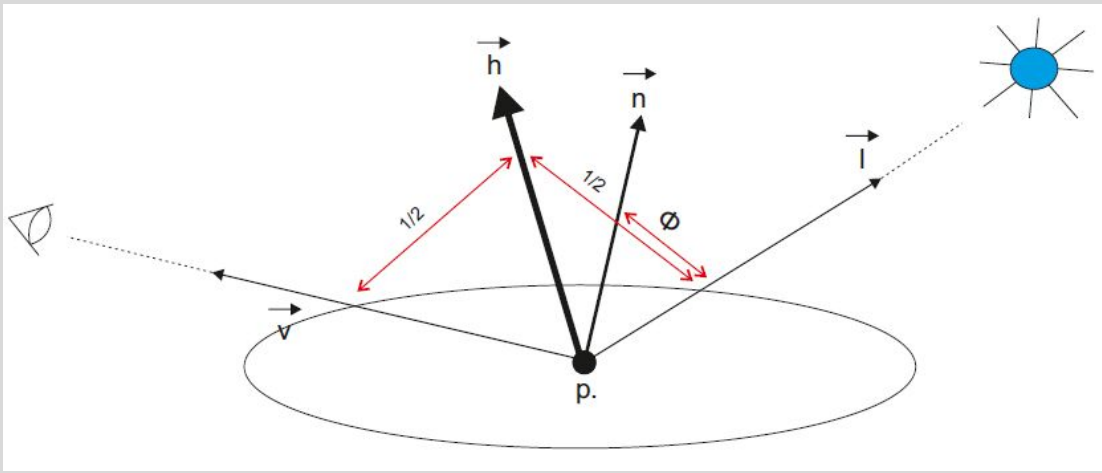
Model oświetlenia Blinna Phonga

$$I = \sum_{i=1}^{N} I_a + I_d + I_s$$

$$I_a = M_a \cdot L_a$$

$$I_d = M_d \cdot L_d \cdot (|\vec{n}| \cdot |\vec{l}|)$$

$$I_s = M_s \cdot L_s \cdot (|\vec{n}| \cdot |\vec{h}|)^{M_{shi}}$$



Zachodniopomorski Uniwersytet Technologiczny w Szczecinie

Wydział Informatyki

Oświetlenie - plan działania

- W projekcie qtglgame jest już zaimplementowany prosty system oświetlenia, zawierający tylko jedno źródło światła ze składowymi ambient i diffuse.
- W ramach dzisiejszych zajęć projektowych rozbudujemy aplikację :
  - Dodamy składową **specular** do źródła światła.
  - Rozbudujemy parametry materiału.
    - Na stan obecny w shaderach posiadamy tylko zmienną **modelColor**.
    - Tą zmienną trzeba zastąpić strukturą zawierającą pola **ambient**, **diffuse**, **specular** i **shininess**.
  - Możliwość ustawiania kilka źródeł światła.
    - Każde światło będzie można włączyć/wyłączyć.
    - Pętla w shaderze iterująca po włączonych źródłach światła.
    - Wymagane utworzenie wektora struktur zawierających lokację zmiennych w shaderze.

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Składowa specular

- Do programów cieniujących w strukturze **Light** musimy dodać nowy wektor trzyelementowy **specular**.
- Do wyliczenia składowej specular potrzebujemy pozycji kamery, dlatego dodamy także zmienną **cameraPosition**.

shader.vs

```
attribute vec4 vertex;
attribute vec3 normal;
attribute vec2 uvCoord;
uniform mat4 projMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
uniform int hasTexture;
uniform sampler2D texture;
uniform highp vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
};
uniform Light light;
```

shader.fs

```
uniform highp vec3 modelColor;
uniform int hasTexture;
uniform sampler2D texture;
uniform vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
};
uniform Light light;
```

Wydział
Informatyki

Składowa specular

- Stwórzmy zmienne w klasie **GLWidget** w **glwidget.h** przechowujące lokacje zmiennej cameraPosition (**m_cameraPositionLoc**) oraz pola specular w strukturze Light (**specular** w **LightLocStruct**).
- Następnie musimy w **GLWidget::InitializeGL()** w **glwidget.cpp** odnaleźć lokację tych zmiennych w shaderze i zapisać do utworzonych pól.

```
struct LightLocStruct
{
    int position;
    int ambient;
    int diffuse;
    int specular;
};

QPoint m_lastPos;
QOpenGLShaderProgram *m_program;
int m_projMatrixLoc;
int m_viewMatrixLoc;
int m_modelMatrixLoc;
int m_modelColorLoc;
int m_hasTextureLoc;
int m_cameraPositionLoc;
LightLocStruct m_lightLoc;

QMatrix4x4 m_proj;
QMatrix4x4 m_camera;
QMatrix4x4 m_world;

bool m_keyState[256]={0};
```

glwidget.h

```
void GLWidget::initializeGL()
{
    initializeOpenGLFunctions();
    glClearColor(0.1f, 0.2f, 0.3f, 1);
    glFrontFace(GL_CCW);
    glCullFace(GL_BACK);

    m_program = new QOpenGLShaderProgram;
    m_program->addShaderFromSourceFile(QOpenGLShader::Vertex, "resources/shader.vs");
    m_program->addShaderFromSourceFile(QOpenGLShader::Fragment, "resources/shader.fs");
    m_program->bindAttributeLocation("vertex", 0);
    m_program->bindAttributeLocation("normal", 1);
    m_program->bindAttributeLocation("uvCoord", 2);
    m_program->link();

    m_program->bind();
    m_projMatrixLoc = m_program->uniformLocation("projMatrix");
    m_viewMatrixLoc = m_program->uniformLocation("viewMatrix");
    m_modelMatrixLoc = m_program->uniformLocation("modelMatrix");
    m_modelColorLoc = m_program->uniformLocation("modelColor");
    m_hasTextureLoc = m_program->uniformLocation("hasTexture");
    m_cameraPositionLoc = m_program->uniformLocation("cameraPosition");
    m_lightLoc.position = m_program->uniformLocation("light.position");
    m_lightLoc.ambient = m_program->uniformLocation("light.ambient");
    m_lightLoc.diffuse = m_program->uniformLocation("light.diffuse");
    m_lightLoc.specular = m_program->uniformLocation("light.specular");

    m_program->release();
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Składowa specular

- W shaderze mamy prawie wszystkie zmienne potrzebne do wyliczenia składowej specular (brakuje jeszcze składowej zmiennej shininess materiału, tym zajmiemy się w dalszej części instrukcji).
- W **shader.fs** w funkcji **main()** rozbudowujemy algorytm wyliczenia koloru powierzchni.

```
void main() {

    highp vec3 tex = texture2D(texture, fragUV).xyz;
    highp vec3 colorFull = vec3(0, 0, 0);

    highp vec3 N = normalize(fragNormal);
    highp vec3 V = normalize(vertexWorldSpace - cameraPosition);

    highp vec3 L = normalize(light.position - vertexWorldSpace);
    highp vec3 H = normalize(L + V);

    highp float cosNL = dot(N, L);
    cosNL = clamp(cosNL, 0.0, 1.0);

    highp float cosVH = dot(V, H);
    cosVH = clamp(cosVH, 0.0, 1.0);

    highp vec3 colorAmb = modelColor * light.ambient;
    highp vec3 colorDif = modelColor * light.diffuse * cosNL;
    highp vec3 colorSpe = modelColor * light.specular * cosVH;

    if(hasTexture == 1)
    {
        colorAmb = colorAmb * tex;
        colorDif = colorDif * tex;
    }

    colorFull = clamp(colorAmb + colorDif + colorSpe, 0.0, 1.0);

    gl_FragColor = vec4(colorFull, 1.0);
}
```

Wydział Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Składowa specular

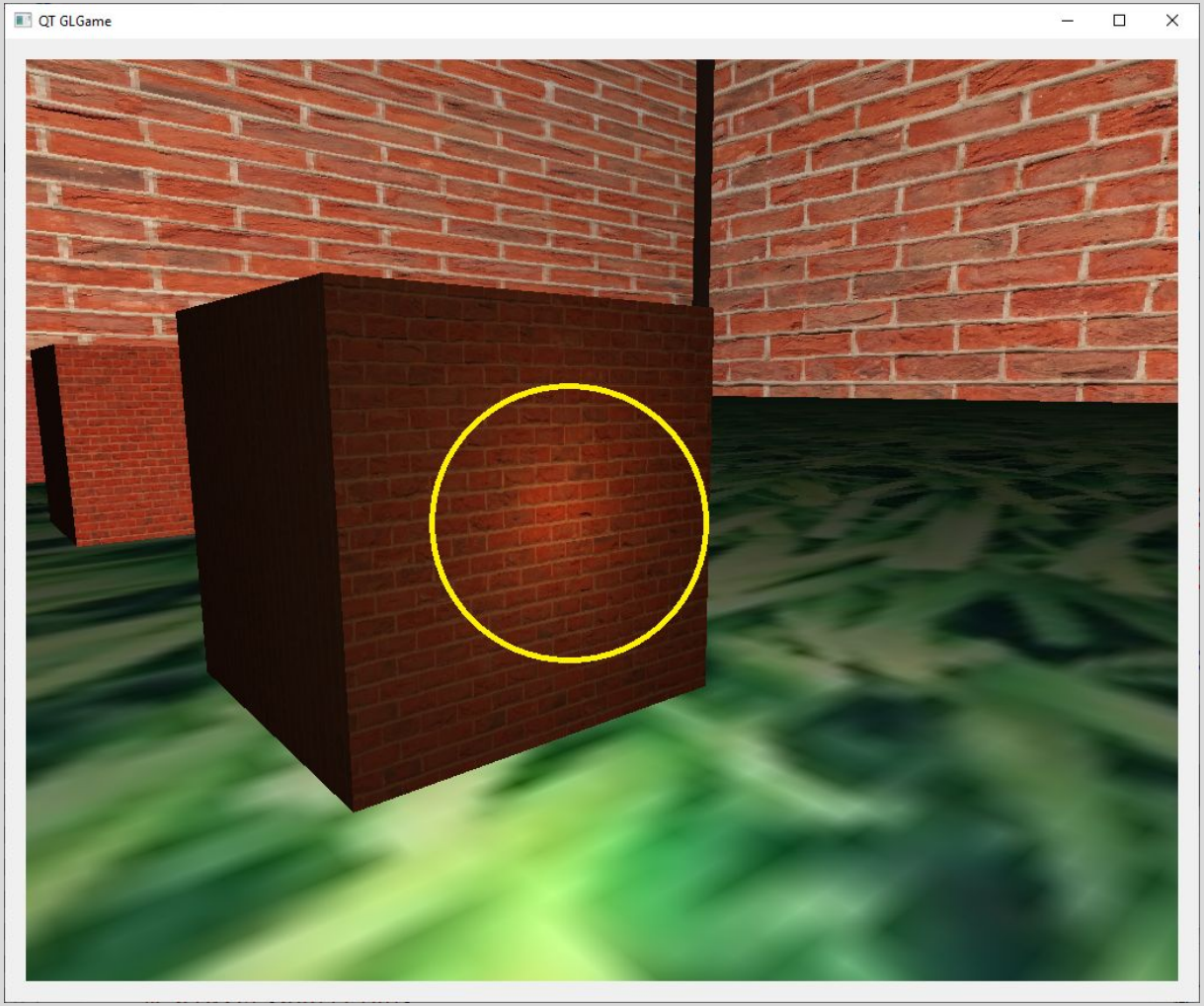
- Teraz wystarczy ustawić składową specular w shaderze na jakąś wartość w **GLWidget::paintGL()** w **glwidget.cpp** (polecam ustawienie każdej składowej na tą samą, małą wartość, np. 0.1f lub 0.2f).
- Trzeba także przekazać pozycję kamery do shadera.

```
void GLWidget::paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    QStack<QMatrix4x4> worldMatrixStack;

    m_program->bind();

    m_program->setUniformValue(m_cameraPositionLoc, m_player.position - m_camDistance * m_player.direction);

    m_program->setUniformValue(m_lightLoc.position, m_player.position - m_player.direction);
    m_program->setUniformValue(m_lightLoc.ambient, QVector3D(0.1f, 0.1f, 0.1f));
    m_program->setUniformValue(m_lightLoc.diffuse, QVector3D(0.9f, 0.9f, 0.9f));
    m_program->setUniformValue(m_lightLoc.specular, QVector3D(0.1f, 0.1f, 0.1f));

    m_world.setToIdentity();
    m_camera.setToIdentity();

    m_camera.lookAt(
    m_player.position - m_camDistance * m_player.direction,
    m_player.position,
    QVector3D(0, 1, 0) );
```

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Składowa specular



Wydział
Informatyki

Materiały obiektów

- Aktualnie za kolor obiektu odpowiada zmienna **material_color**.
- Trzeba tę zmienną zastąpić strukturą **Material** zawierającą składowe **ambient**, **diffuse**, **specular** oraz **shininess**.
- Stwórzmy taką strukturę w klasie **GameObject** w **gameobject.h**.

```
class GameObject
{
public:
    GameObject();

    QVector3D position = QVector3D(0.0f, 0.0f, 0.0f);
    QVector3D rotation = QVector3D(0.0f, 0.0f, 0.0f);
    QVector3D scale = QVector3D(1.0f, 1.0f, 1.0f);

    QVector3D previousPosition = QVector3D(0.0f, 0.0f, 0.0f);

    float m_radius = 1.0f;

    QVector3D material_color = QVector3D(1.0f, 1.0f, 1.0f);

    struct Material
    {
        QVector3D ambient = QVector3D(1.0f, 1.0f, 1.0f);
        QVector3D diffuse = QVector3D(1.0f, 1.0f, 1.0f);
        QVector3D specular = QVector3D(1.0f, 1.0f, 1.0f);
        float shininess = 1.0f;
    };

    Material material;
```

Materiały obiektów

- Dodając strukturę **Material** usuwając poprzednią zmienną **modelColor**, przy dodawaniu obiektów należy zmodyfikować sposób ustawiania koloru.
- Przykład z **GLWidget::initializeGL()** z **glwidget.cpp** przy generowania Cube'ów.

```
addObject(&m_player);

for(int i = 0 ; i < 5 ; i++)
    {
        for(int j = 0 ; j < 7 ; j++)
        {
            Cube* cube = new Cube();

            cube->position.setX(j * 1 - 3);
            cube->position.setY(0);
            cube->position.setZ(i * 1 - 6);

            cube->material_color.setX(i * 0.2f);
            cube->material_color.setY(0.5f);
            cube->material_color.setZ(j * 0.1f);

            cube->material.diffuse.setX(i * 0.2f);
            cube->material.diffuse.setY(0.5f);
            cube->material.diffuse.setZ(j * 0.1f);

            cube->scale = QVector3D(0.3f, 0.3f, 0.3f);

            cube->m_radius = 0.5 * sqrt(3 * cube->scale.x() * cube->scale.x());

            cube->m_texture = TextureManager::getTexture("brick");

            addObject(cube);
        }
    }
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Materiały obiektów

- Kolejnym krokiem jest dodanie struktury **Material** w shaderach, usuwając przy okazji zmienną **modelColor**.

shader.vs

```
attribute vec4 vertex;
attribute vec3 normal;
attribute vec2 uvCoord;
uniform mat4 projMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
uniform int hasTexture;
uniform sampler2D texture;
uniform highp vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
};
uniform Light light;

struct Material {
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
    highp float shininess;
};
uniform Material material;
```

shader.fs

```
uniform highp vec3 modelColor;
uniform int hasTexture;
uniform sampler2D texture;
uniform vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
};
uniform Light light;

struct Material {
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
    highp float shininess;
};
uniform Material material;
```

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Materiały obiektów

- Następnie trzeba zmodyfikować obliczenia w **shader.fs**, aby korzystała ze zmiennych struktury **Material**.

```
void main() {

    highp vec3 tex = texture2D(texture, fragUV).xyz;
    highp vec3 colorFull = vec3(0, 0, 0);

    highp vec3 N = normalize(fragNormal);
    highp vec3 V = normalize(vertexWorldSpace - cameraPosition);

    highp vec3 L = normalize(light.position - vertexWorldSpace);
    highp vec3 H = normalize(L + V);

    highp float cosNL = dot(N, L);
    cosNL = clamp(cosNL, 0.0, 1.0);

    highp float cosVH = dot(V, H);
    cosVH = clamp(cosVH, 0.0, 1.0);

    highp vec3 colorAmb = material.ambient  * light.ambient;
    highp vec3 colorDif = material.diffuse  * light.diffuse * cosNL;
    highp vec3 colorSpe = material.specular * light.specular * pow(cosVH, material.shininess);

    if(hasTexture == 1)
    {
        colorAmb = colorAmb * tex;
        colorDif = colorDif * tex;
    }

    colorFull = clamp(colorAmb + colorDif + colorSpe, 0.0, 1.0);

    gl_FragColor = vec4(colorFull, 1.0);
}
```

Wydział
Informatyki

Materiały obiektów

- W klasie **GLWidget** w **glwidget.h** należy utworzyć strukturę **MaterialLocStruct** wraz z polem jej typu przechowującą lokację zmiennych struktury **Material** w shaderze.
- Usuńmy także zmienną **m_modelColorLoc**.

```
struct LightLocStruct
{
    int position;
    int ambient;
    int diffuse;
    int specular;
};

struct MaterialLocStruct
{
    int ambient;
    int diffuse;
    int specular;
    int shininess;
};

QPoint m_lastPos;
QOpenGLShaderProgram *m_program;
int m_projMatrixLoc;
int m_viewMatrixLoc;
int m_modelMatrixLoc;
int m_modelColorLoc;
int m_hasTextureLoc;
int m_cameraPositionLoc;
LightLocStruct m_lightLoc;
MaterialLocStruct m_materialLoc;
```

Materiały obiektów

- W **GLWidget::initializeGL()** w **glwidget.cpp** należy polom struktury **MaterialLocStruct** przypisać lokację odpowiadających zmiennych w shaderze.
- W tym samym miejscu należy usunąć odwołanie do **m_modelColorLoc**.

```
void GLWidget::initializeGL()
{
    initializeOpenGLFunctions();
    glClearColor(0.1f, 0.2f, 0.3f, 1);
    glFrontFace(GL_CCW);
    glCullFace(GL_BACK);

    m_program = new QOpenGLShaderProgram;
    m_program->addShaderFromSourceFile(QOpenGLShader::Vertex, "resources/shader.vs");
    m_program->addShaderFromSourceFile(QOpenGLShader::Fragment, "resources/shader.fs");
    m_program->bindAttributeLocation("vertex", 0);
    m_program->bindAttributeLocation("normal", 1);
    m_program->bindAttributeLocation("uvCoord", 2);
    m_program->link();

    m_program->bind();
    m_projMatrixLoc = m_program->uniformLocation("projMatrix");
    m_viewMatrixLoc = m_program->uniformLocation("viewMatrix");
    m_modelMatrixLoc = m_program->uniformLocation("modelMatrix");
    m_hasTextureLoc = m_program->uniformLocation("hasTexture");
    m_cameraPositionLoc = m_program->uniformLocation("cameraPosition");

    m_materialLoc.ambient = m_program->uniformLocation("material.ambient");
    m_materialLoc.diffuse = m_program->uniformLocation("material.diffuse");
    m_materialLoc.specular = m_program->uniformLocation("material.specular");
    m_materialLoc.shininess = m_program->uniformLocation("material.shininess");

    m_lightLoc.position = m_program->uniformLocation("light.position");
    m_lightLoc.ambient = m_program->uniformLocation("light.ambient");
    m_lightLoc.diffuse = m_program->uniformLocation("light.diffuse");
    m_lightLoc.specular = m_program->uniformLocation("light.specular");
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Materiały obiektów

- Ostatnim krokiem, jest przypisanie wartości struktury **Material** w shaderze właściwości materiałów obiektów gry.
- W pętli renderowania w **GLWidget::paintGL()** w **glwidget.cpp** należy zastąpić przypisanie koloru do zmiennej **modelColor** nowo utworzonymi zmiennymi.

```
void GLWidget::paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    QStack<QMatrix4x4> worldMatrixStack;

    m_program->bind();

    m_program->setUniformValue(m_cameraPositionLoc, m_player.position - m_camDistance * m_player.direction);

    m_program->setUniformValue(m_lightLoc.position, m_player.position - m_player.direction);
    m_program->setUniformValue(m_lightLoc.ambient, QVector3D(0.1f, 0.1f, 0.1f));
    m_program->setUniformValue(m_lightLoc.diffuse, QVector3D(0.9f, 0.9f, 0.9f));
    m_program->setUniformValue(m_lightLoc.specular, QVector3D(0.1f, 0.1f, 0.1f));

    m_world.setToIdentity();
    m_camera.setToIdentity();

    m_camera.lookAt(
    m_player.position - m_camDistance * m_player.direction,
    m_player.position,
    QVector3D(0, 1, 0) );

    for(int i = 0 ; i < m_gameObjects.size() ; i++)
    {
        GameObject* obj = m_gameObjects[i];

        m_program->setUniformValue(m_materialLoc.ambient, obj->material.ambient);
        m_program->setUniformValue(m_materialLoc.diffuse, obj->material.diffuse);
        m_program->setUniformValue(m_materialLoc.specular, obj->material.specular);
        m_program->setUniformValue(m_materialLoc.shininess, obj->material.shininess);

        if(obj->m_texture != nullptr)
        {
            m_program->setUniformValue(m_hasTextureLoc, 1);
            obj->m_texture->bind();
        }
```

Wydział Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Materiały obiektów

- To samo należy zrobić z renderowaniem trójkątów kolizyjnych, także w **GLWidget::paintGL()** w **glwidget.cpp**.

```
for(int i = 0 ; i < collisionTriangles.size()  ; /*nic*/) {

    Triangle triangle = collisionTriangles[i];

    m_program->setUniformValue(m_modelToWorldLoc, QVector3D(1, 1, 1));

    m_program->setUniformValue(m_materialLoc.ambient, QVector3D(1.0f, 1.0f, 1.0f));
    m_program->setUniformValue(m_materialLoc.diffuse, QVector3D(1.0f, 1.0f, 1.0f));
    m_program->setUniformValue(m_materialLoc.specular, QVector3D(1.0f, 1.0f, 1.0f));
    m_program->setUniformValue(m_materialLoc.shininess, 1.0f);

    if(triangle.texture != nullptr)
    {
        m_program->setUniformValue(m_hasTextureLoc, 1);
        triangle.texture->bind();
    }
    else
    {
        m_program->setUniformValue(m_hasTextureLoc, 0);
    }

    worldMatrixStack.push(m_world);
        m_world.translate(QVector3D(0, 0, 0));
        m_world.rotate(0, 1, 0, 0);
        m_world.rotate(0, 0, 1, 0);
        m_world.rotate(0, 0, 0, 1);
        m_world.scale(QVector3D(1 ,1 ,1));
        setTransforms();
        collisionTrianglesMesh.render(this, i * 3, triangle.groupSize * 3);
    m_world = worldMatrixStack.pop();

    i += triangle.groupSize;
}
```

- W tym miejscu przypisujemy stałe właściwości materiału. Dla chętnych można pokusić się o ich implementację w strukturze **Triangle**.

Wydział
Informatyki

Wiele źródeł światła

- Mamy już zaimplementowany pełny model oświetlenia Blinna-Phonga, ogranicza nas tylko jedno źródło światła. Aby dodać obsługę kilku źródeł światła, w klasie **GLWidget** w **glwidget.h** należy dodać kilka rzeczy:
  - Stworzyć strukturę **Light** zawierającą pola **position**, **ambient**, **diffuse**, **specular**, **isActive** (stan włączony/wyłączony) oraz **attenuation** (wygaszanie).
  - Rozszerzmy strukturę przechowującą lokację zmiennych w shaderze o dwa dodatkowe pola (**isActive** i **attenuation**).
  - Stwórzmy zmienną **MAX_LIGHT**, która będzie przechowywać maksymalną ilość źródeł światła (np. 5).
  - Stwórzmy tablicę zmiennych typu **Light** przechowujących informację o parametrach źródeł światła.
  - Zmieńmy pole typu **LightLocStruct** na tablicę.
  - Dodajmy metodę **setLights()**, która wyśle informację o parametrach źródłach światła do shadera.
- Zrzut ekranu zawierających tę listę zmian na kolejnym slajdzie.

Wiele źródeł światła

- Klasa **GLWidget** w **glwidget.h** :
- Należy zwrócić uwagę na zmianę z

  LightLocStruct m_lightLoc;

  na

  LightLocStruct m_lightLoc[MAX_LIGHTS];
- Ustawimy także domyślne pole **isActive** na **false** w celu początkowego wyłączenia wszystkich źródeł światła.
- Zmienną **MAX_LIGHTS** można ustawić na inną wartość, ale trzeba będzie pamiętać, żeby ustawić tą samą wartość w shaderach.

```
struct LightLocStruct
{
    int position;
    int ambient;
    int diffuse;
    int specular;
    int isActive;
    int attenuation;
};

struct MaterialLocStruct
{
    int ambient;
    int diffuse;
    int specular;
    int shininess;
};

struct Light
{
    QVector3D position;
    QVector3D ambient;
    QVector3D diffuse;
    QVector3D specular;
    bool isActive = false;
    float attenuation;
};

static const int MAX_LIGHTS = 5;
Light m_lights[MAX_LIGHTS];
void setLights();

QPoint m_lastPos;
QOpenGLShaderProgram *m_program;
int m_projMatrixLoc;
int m_viewMatrixLoc;
int m_modelMatrixLoc;
int m_cameraPositionLoc;
int m_hasTextureLoc;
LightLocStruct m_lightLoc[MAX_LIGHTS];
MaterialLocStruct m_materialLoc;
```

Wiele źródeł światła

- W shaderach także należy zmienić zmienną typu struktury **Light** na tablicę (wystarczy dopisać **[MAX_LIGHTS]** do **uniform Light light;**.
- Należy także stworzyć zmienną **const int MAX_LIGHTS = 5;**.
- s
- Do struktury **Light** dodajmy także pola **isAlive** i **attenuation**.

shader.vs

```
attribute vec4 vertex;
attribute vec3 normal;
attribute vec2 uvCoord;
uniform mat4 projMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelMatrix;
uniform int hasTexture;
uniform sampler2D texture;
uniform highp vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

const int MAX_LIGHTS = 5;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
    int isActive;
    float attenuation;
};
uniform Light light[MAX_LIGHTS];
```

shader.fs

```
uniform highp vec3 modelColor;
uniform int hasTexture;
uniform sampler2D texture;
uniform vec3 cameraPosition;
varying highp vec3 fragNormal;
varying highp vec3 vertexWorldSpace;
varying highp vec2 fragUV;

const int MAX_LIGHTS = 5;

struct Light {
    highp vec3 position;
    highp vec3 ambient;
    highp vec3 diffuse;
    highp vec3 specular;
    int isActive;
    float attenuation;
};
uniform Light light[MAX_LIGHTS];
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wiele źródeł światła

- Kolejnym krokiem jest odnalezienie lokacji zmiennych w shaderze i przypisanie ich do naszej struktury w GLWidget.
- Odnajdywanie lokacji zmiennych w shaderze przechowywanych w tablicy może może wydawać się kłopotliwe, ale po pierwszej implementacji nie powinno sprawiać problemu :)
- Przypisywanie lokacji zmiennych w shaderze implementujemy tak samo jak w poprzednich przypadkach w **GLWidget::initializeGL()**.

```
m_materialLoc.ambient = m_program->uniformLocation("material.ambient");
m_materialLoc.diffuse = m_program->uniformLocation("material.diffuse");
m_materialLoc.specular = m_program->uniformLocation("material.specular");
m_materialLoc.shininess = m_program->uniformLocation("material.shininess");

m_lightLoc.position = m_program->uniformLocation("light.position");
m_lightLoc.ambient = m_program->uniformLocation("light.ambient");
m_lightLoc.diffuse = m_program->uniformLocation("light.diffuse");
m_lightLoc.specular = m_program->uniformLocation("light.specular");

for(int i = 0 ; i < MAX_LIGHTS; i++)
{
    m_lightLoc[i].position = m_program->uniformLocation(QString().asprintf("light[%d].position", i);
    m_lightLoc[i].ambient = m_program->uniformLocation(QString().asprintf("light[%d].ambient", i);
    m_lightLoc[i].diffuse = m_program->uniformLocation(QString().asprintf("light[%d].diffuse", i);
    m_lightLoc[i].specular = m_program->uniformLocation(QString().asprintf("light[%d].specular", i);
    m_lightLoc[i].isActive = m_program->uniformLocation(QString().asprintf("light[%d].isActive", i);
    m_lightLoc[i].attenuation = m_program->uniformLocation(QString().asprintf("light[%d].attenuation", i);
}

m_program->release();
```

Wydział Informatyki

Wiele źródeł światła

- Należy także zaimplementować metodę **GLWidget::setLights()** w **glwidget.cpp**, która będzie odpowiedzialna za wysłanie wszyskich informacji o parametrach źródeł światła do shaderów.

```
void GLWidget::setLights()
{
    for(int i = 0; i < MAX_LIGHTS; i++)
    {
        m_program->setUniformValue(m_lightLoc[i].position, m_lights[i].position);
        m_program->setUniformValue(m_lightLoc[i].ambient, m_lights[i].ambient);
        m_program->setUniformValue(m_lightLoc[i].diffuse, m_lights[i].diffuse);
        m_program->setUniformValue(m_lightLoc[i].specular, m_lights[i].specular);
        m_program->setUniformValue(m_lightLoc[i].isActive, m_lights[i].isActive);
        m_program->setUniformValue(m_lightLoc[i].attenuation, m_lights[i].attenuation);
    }
}
```

- Tę metodę będziemy używać w **GLWidget::paintGL()** w **glwidget.cpp** zamiast poprzedniego wysyłania parametrów źródeł światła.

```
void GLWidget::paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    QStack<QMatrix4x4> worldMatrixStack;

    m_program->bind();

    m_program->setUniformValue(m_cameraPositionLoc, m_player.position - m_camDistance * m_player.direction);

    setLights();

    m_program->setUniformValue(m_lightLoc.position, m_player.position - m_player.direction);
    m_program->setUniformValue(m_lightLoc.ambient, QVector3D(0.1f, 0.1f, 0.1f));
    m_program->setUniformValue(m_lightLoc.diffuse, QVector3D(0.9f, 0.9f, 0.9f));
    m_program->setUniformValue(m_lightLoc.specular, QVector3D(0.1f, 0.1f, 0.1f));

    m_world.setToIdentity();
    m_camera.setToIdentity();

    m_camera.lookAt(
    m_player.position - m_camDistance * m_player.direction,
    m_player.position,
    QVector3D(0, 1, 0) );
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wiele źródeł światła

- Ostatnim krokiem jest zastosowanie wielu źródeł światła w shaderach.
- Dodamy także wygaszanie źródła światła zależnie od odległości.
- Wszystkie obliczenia dokonujemy w **shader.fs**.

```
void main() {

    highp vec3 tex = texture2D(texture, fragUV).xyz;
    highp vec3 colorFull = vec3(0, 0, 0);

    highp vec3 N = normalize(fragNormal);
    highp vec3 V = normalize(vertexWorldSpace - cameraPosition);

    for(int i = 0 ; i < MAX_LIGHTS ; i++)
    {
        if(light[i].isActive == 1)
        {

            highp vec3 L = normalize(light[i].position - vertexWorldSpace);
            highp vec3 H = normalize(L + V);

            highp float D = length(light[i].position - vertexWorldSpace);


            highp float cosNL = dot(N, L);
            cosNL = clamp(cosNL, 0.0, 1.0);

            highp float cosVH = dot(V, H);
            cosVH = clamp(cosVH, 0.0, 1.0);

            highp vec3 colorAmb = material.ambient  * light[i].ambient;
            highp vec3 colorDif = material.diffuse  * light[i].diffuse * cosNL;
            highp vec3 colorSpe = material.specular * light[i].specular * pow(cosVH, material.shininess);

            if(hasTexture == 1)
            {
                colorAmb = colorAmb * tex;
                colorDif = colorDif * tex;
            }

            colorAmb = colorAmb / pow(D, light[i].attenuation);
            colorDif = colorDif / pow(D, light[i].attenuation);
            colorSpe = colorSpe / pow(D, light[i].attenuation);

            colorFull = clamp(colorFull + colorAmb + colorDif + colorSpe, 0.0, 1.0);
        }
    }

    gl_FragColor = vec4(colorFull, 1.0);
}
```


Wydział
Informatyki

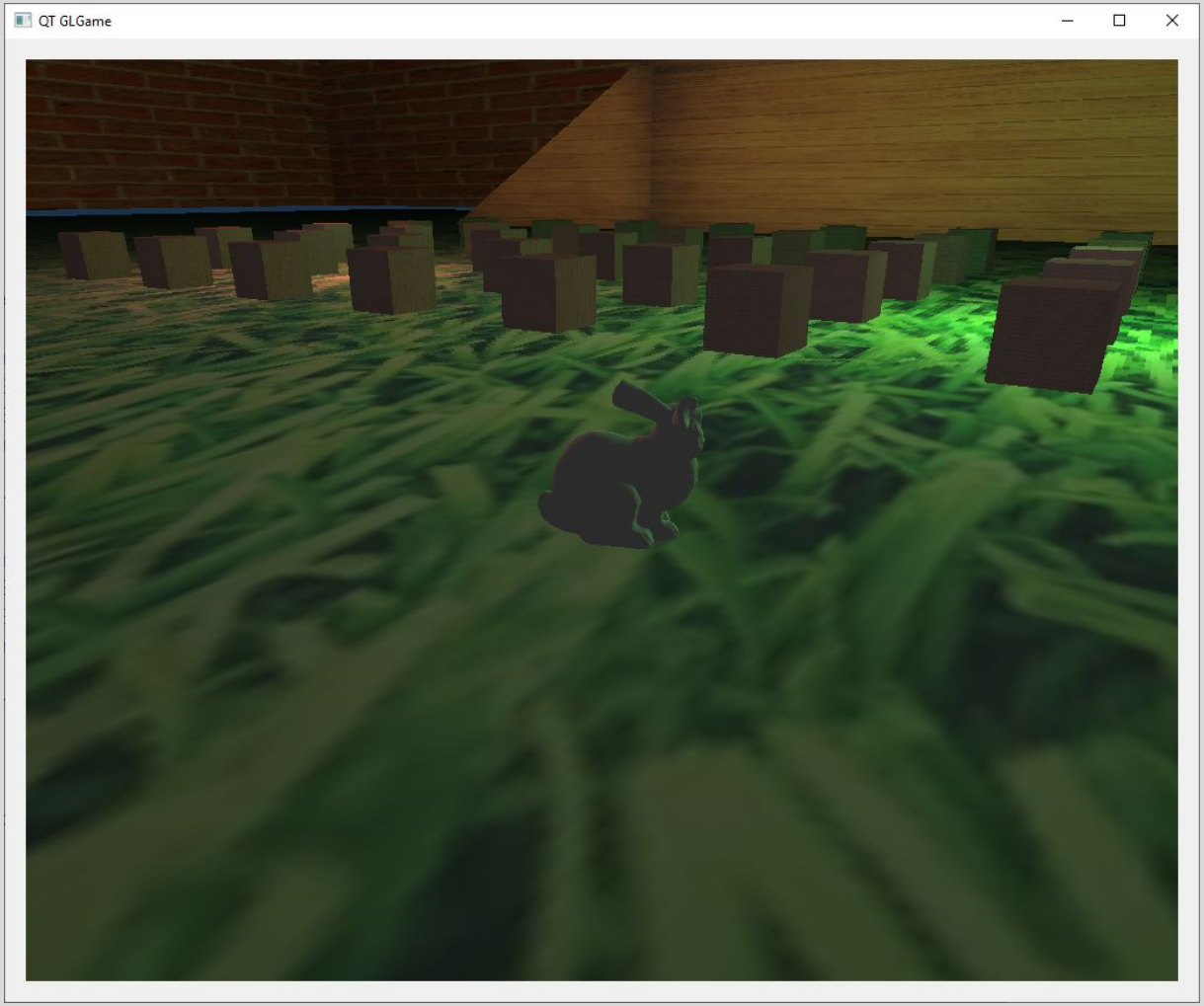
Wiele źródeł światła

- Udało się zaimplementować wiele źródeł światła! :)
- Teraz należy je wstawić w **GLWidget::paintGL()** w **glwidget.cpp**.


```
void GLWidget::paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    QStack<QMatrix4x4> worldMatrixStack;

    m_lights[0].position = QVector3D(-3.0f, 1.0f, -4.0f);
    m_lights[0].ambient = QVector3D(0.1f, 0.1f, 0.1f);
    m_lights[0].diffuse = QVector3D(1.0f, 0.3f, 0.3f);
    m_lights[0].specular = QVector3D(0.1f, 0.1f, 0.1f);
    m_lights[0].isActive = true;
    m_lights[0].attenuation = 0.5f;

    m_lights[1].position = QVector3D(3.0f, 1.0f, -4.0f);
    m_lights[1].ambient = QVector3D(0.1f, 0.1f, 0.1f);
    m_lights[1].diffuse = QVector3D(0.3f, 1.0f, 0.3f);
    m_lights[1].specular = QVector3D(0.1f, 0.1f, 0.1f);
    m_lights[1].isActive = true;
    m_lights[1].attenuation = 0.5f;

    m_program->bind();

    m_program->setUniformValue(m_cameraPositionLoc, m_player.position - m_camDistance * m_player.direction);

    setLights();
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wiele źródeł światła



Wydział
Informatyki

HUD - Head-Up Display

- Celem **Head-Up Display** jest wyświetlenie elementów na ekranie, które opisują stan gry (**GUI - graphic user interface**). Przykładem może być :
  - Pasek symbolizujący poziom zdrowia gracza (ilość żyć).
  - Wyświetlenie ikon symbolizujących elementy gry (np. zdolności, przedmioty).
- Wyświetlanie takich elementów odbywa się poprzez wykorzystanie **rzutu ortograficznego**.
- Aby wyświetlić elementy **HUDu** należy stworzyć **nowy shader** (w instrukcji nazwiemy go **shader_hud**).
- Każdy element HUDu będzie prostokątem (**rectangle**). Musimy stworzyć nowy mesh, który będzie zawierał w sobie wierzchołki prostokąta,

HUD - Rectangle Mesh

- Najpierw zajmiemy się utworzeniem prostokąta. Stwórzmy deklaracje metody **generateRectangle()** wewnątrz klasy **CMesh** w **cmesh.h**.

```
class CMesh
{
public:
    CMesh();
    ~CMesh();
    const GLfloat *constData() const { return m_data.constData(); }
    int vertexCount() const { return m_count; }
    GLenum primitive() {return m_primitive; }

    void generateCube(GLfloat ww, GLfloat hh, GLfloat dd);
    void generateSphere(float r, int N);
    void generateRectangle();
    void generateMeshFromObjFile(QString filename);
```

- oraz jej definicję w **cmesh.cpp**

```
void CMesh::generateRectangle()
{
    add(QVector3D(0, 0, 0), QVector3D(0, 0, 0), QVector2D(0, 0));
    add(QVector3D(0, 1, 0), QVector3D(0, 0, 0), QVector2D(0, 1));
    add(QVector3D(1, 1, 0), QVector3D(0, 0, 0), QVector2D(1, 1));
    add(QVector3D(1, 0, 0), QVector3D(0, 0, 0), QVector2D(1, 0));

    m_primitive = GL_TRIANGLE_FAN;

    initVboAndVao();
}
```

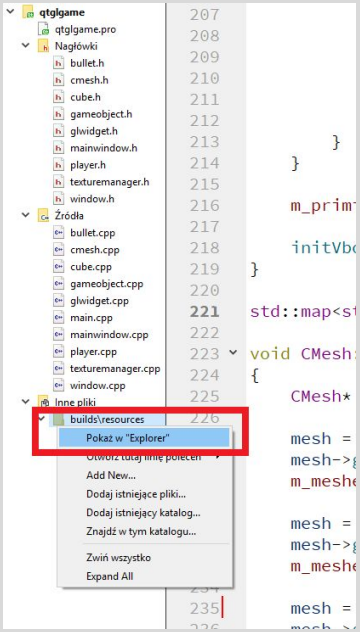
HUD - Rectangle Mesh

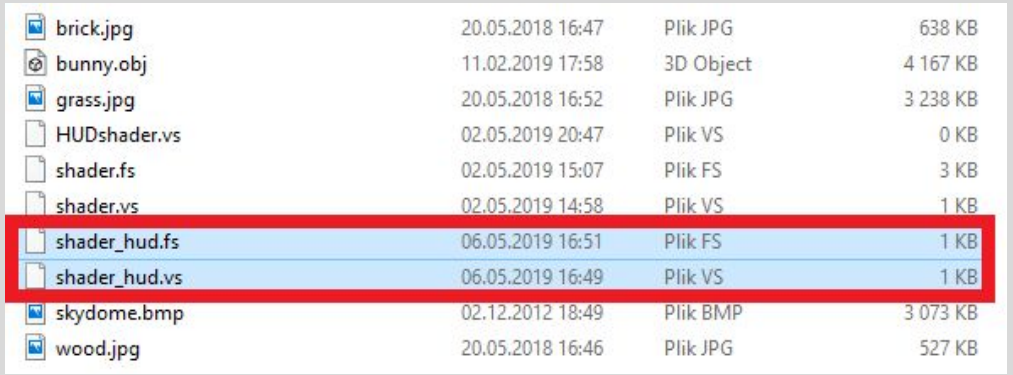
- Mając już przygotowaną metodę generującą prostokąt, możemy ją wykorzystać do w celu dodania nowej siatki do naszego programu.
- Wewnątrz stworzonej wcześniej metodzie **CMesh::loadAllMeshes()** dodajemy wczytanie kolejnej siatki (deklaracja tej metody w **cmesh.cpp**).

```
void CMesh::loadAllMeshes()
{
    CMesh* mesh;

    mesh = new CMesh;
    mesh->generateCube(1.0f, 1.0f, 1.0f);
    m_meshes["cube"] = mesh;

    mesh = new CMesh;
    mesh->generateSphere(0.5f, 24);
    m_meshes["sphere"] = mesh;

    mesh = new CMesh;
    mesh->generateRectangle();
    m_meshes["rect"] = mesh;

    mesh = new CMesh;
    mesh->generateMeshFromObjFile("resources/bunny.obj");
    m_meshes["bunny"] = mesh;
}
```

HUD - Shader

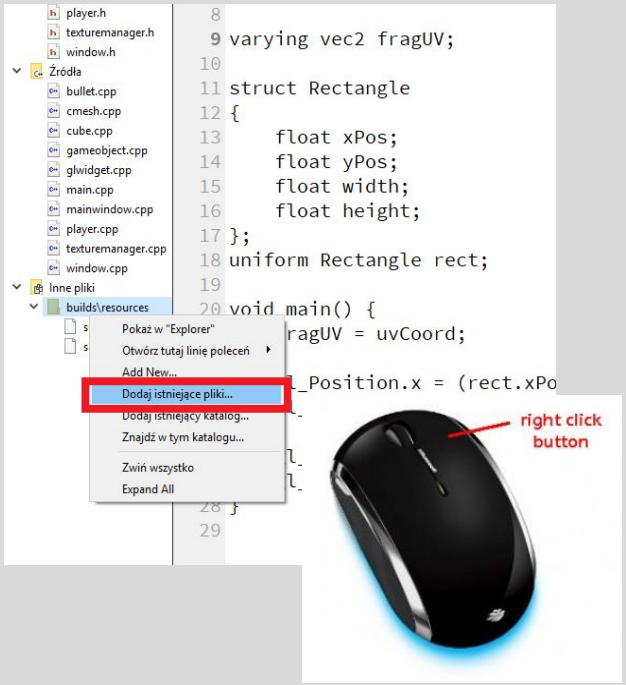
- Kolejnym krokiem jest stworzenie nowego shadera.
- Shader nazwiemy **shader_hud**, będzie się on zawierał w dwóch plikach - **shader_hud.vs** (vertex shader) i **shader_hud.fs** (fragment shader).
- Pliki te trzeba własnoręcznie stworzyć w katalogu **/resources**. Aby dostać się do tego katalogu można klinąć **prawy przycisk myszy** na tym katalogu wewnątrz Qt. Będąc w katalogu tworzymy potrzebne pliki.

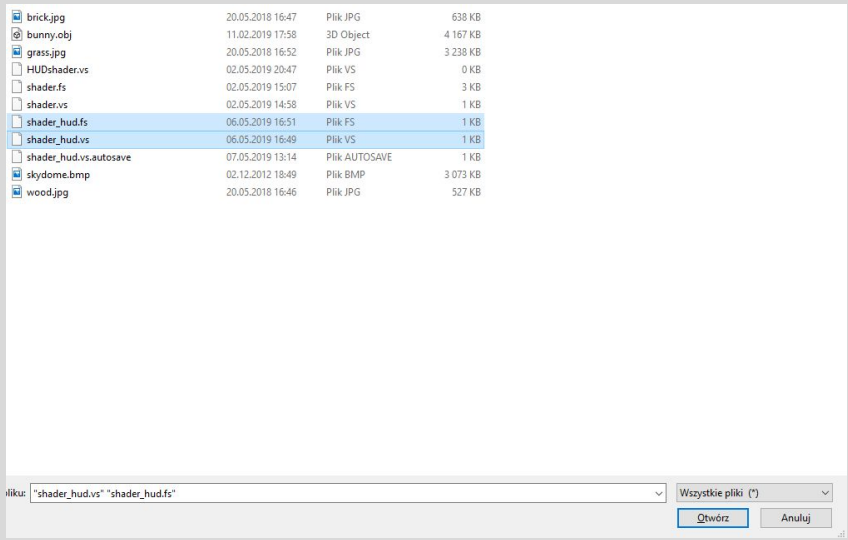




HUD - Shader

- Utworzone pliki można dodać do edytora Qt.
- Aby to zrobić, należy jak poprzednio wcisnąć prawy przycisk myszy na **builds/resources** i wybrać opcję **Dodaj Istniejące Pliki**.





Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

HUD - Vertex shader

- Utworzony vertex shader (**shader_hud.vs**) trzeba wypełnić poniższym kodem.

```
attribute vec4 vertex;
attribute vec3 normal;
attribute vec2 uvCoord;
uniform highp vec3 color;
uniform highp int hasTexture;
uniform sampler2D texture;
uniform highp vec2 resolution;

varying vec2 fragUV;

struct Rectangle
{
  float xPos;
  float yPos;
  float width;
  float height;
};
uniform Rectangle rect;

void main() {
        fragUV = uvCoord;

        gl_Position.x = (rect.xPos + rect.width  * vertex.x) / resolution.x * 2.0 - 1.0;
        gl_Position.y = (rect.yPos + rect.height * vertex.y) / resolution.y * (-2.0) + 1.0;

        gl_Position.z = 0.0;
        gl_Position.w = 1.0;
}
```


Wydział
Informatyki

HUD - Fragment shader

- Utworzony fragment shader (**shader_hud.fs**) trzeba wypełnić poniższym kodem.

```
uniform highp vec3 color;
uniform highp int hasTexture;
uniform sampler2D texture;
uniform highp vec2 resolution;

varying vec2 fragUV;

struct Rectangle
{
   float xPos;
   float yPos;
   float width;
   float height;
};
uniform Rectangle rect;

void main() {

   highp vec4 tex = texture2D(texture, fragUV).xyzw;

   if(hasTexture == 1)
      gl_FragColor = vec4(color * tex.xyz, tex.w);
   else
      gl_FragColor = vec4(color, 1.0);
}
```

- **Dalsze trzy slajdy zawierają wyłącznie opis tych shaderów !**
  - **Dla zainteresowanych :)**

HUD - Opis shaderów

- W kodzie shaderów widać, że wierzchołki posiadają trzy wierzchołki :

```
attribute vec4 vertex;
attribute vec3 normal;
attribute vec2 uvCoord;
```

- Pozycja wierzchołka (**vertex**) i współrzędne tekstury (**uvCoord**) będziemy wykorzystywać, natomiast wektor normalny (**normal**) jest zbędny. Mimo to jest on dodany, ponieważ klasa **CMesh** jest przystosowana do przekazywania tych trzech atrybutów. Z tej klasy będzie zaraz korzystać, więc wygodniej nam będzie przekazywać niewykorzystywany atrybut, niż rozbudować klasę CMesh o przekazywanie wybranych atrybutów.

- Kolejno mamy kilka zmiennych **uniform**.

```
uniform highp vec3 color;
uniform int hasTexture;
uniform sampler2D texture;
uniform vec2 resolution;
```

  - **vec3 color** - kolor naszego prostokąta.
  - **int hasTexture** - zmienna przechowująca informację, czy nałożyć teksturę.
  - **sampler2D texture** - obraz 2D przechowujący teksturę.
  - **vec2 resolution** - rozdzielczość w pikselach zawierający obraz.

HUD - Opis shaderów

- W shaderach zawarta jest także struktura **Rectangle**

```
struct Rectangle
{
  float xPos;
  float yPos;
  float width;
  float height;
};
```

- oraz zmienna uniform **rect** jej typu.

```
uniform Rectangle rect;
```

- Struktura **Rectangle** zawiera cztery pola dotyczące prostokąta : pozycję X, pozycję Y, szerokość i wysokość.
- Ostatnią zmienną poza funkcjami main() jest zmienna **vec2 fragUV**.

```
varying vec2 fragUV;
```

- Dopiska **varying** oznacza, że przypisana wartość w **vertex shaderze** będzie **interpolowana** do **fragmentów** w **fragment shaderze**.

HUD - Opis shaderów

- Funkcja **main()** w **vertex shaderze** posiada następujące linijki kodu :
  - Przepisanie współrzędnych tekstury do fragment shadera.

```
fragUV = uvCoord;
```

  - Ustawienie współrzędnych prostokąta do tych przekazanych w zmiennej **rect** wraz z normalizacją względem rozdzielczości.

```
gl_Position.x = (rect.xPos + rect.width  * vertex.x) / resolution.x * 2.0 - 1.0;
gl_Position.y = (rect.yPos + rect.height * vertex.y) / resolution.y * (-2.0) + 1.0;
```

  - Ustawienie pozostałych wartości wbudowanej zmiennej **gl_Position** na domyślne wartości.

```
gl_Position.z = 0.0;
gl_Position.w = 1.0;
```

- Natomiast funkcja **main()** we **fragment shaderze** posiada :
  - Pobranie koloru tekstury w pozycji **fragUV**

```
highp vec4 tex = texture2D(texture, fragUV).xyzw;
```

  - Oraz ustawienie koloru piksela na kolor tekstury pomnożony przez przekazanych kolor w zmiennej **color** jeśli obiekt ma teksturę.

```
if(hasTexture == 1)
    gl_FragColor = vec4(color * tex.xyz, tex.w);
```

  - Lub jeśli nie posiada tekstury, to tylko na wartości ze zmiennej **color**.

```
else
    gl_FragColor = vec4(color, 1.0);
```

HUD - GLWidget

- Mamy już shadery, teraz należy je wczytać wewnątrz klasy **GLWidget.**
- W klasie **GLWidget** w **glwidget.h** dodajemy poniższe linijki kodu :

```
int m_hasTextureLoc;
int m_cameraPositionLoc;
LightLocStruct m_lightLoc[MAX_LIGHTS];
MaterialLocStruct m_materialLoc;

QOpenGLShaderProgram *m_program_hud;
int m_resolutionLoc_hud;
int m_color_hud;
int m_hasTextureLoc_hud;
RectangleLocStruct m_rectangleLoc_hud;
void setRectangle(float xPos, float yPos, float width, float height, QVector3D color, QOpenGLTexture* texture);
QVector2D m_resolution;

QMatrix4x4 m_proj;
QMatrix4x4 m_camera;
QMatrix4x4 m_world;

bool m_keyState[256]={0};

float m_camDistance = 1.5f;

QElapsedTimer timer;
```

HUD - GLWidget

- QOpenGLShaderProgram *m_program_hud;
  - Wskaźnik do shaderów, które utworzyliśmy
- int m_resolutionLoc_hud;
  int m_color_hud;
  int m_hasTextureLoc_hud;
  RectangleLocStruct m_rectangleLoc_hud;
  - Zmienne przechowujące **lokacje** (uchwyty) zmiennych **uniform** w utworzonym shaderze.
- void setRectangle(float xPos, float yPos, float width, float height, QVector3D color, QOpenGLTexture* texture);
  - Funkcja, która przekaże parametry prostokąta do shadera.
- QVector2D m_resolution;
  - Zmienna, która będzie przechowywać rozdzielczość ekranu.

HUD - GLWidget

- W **GLWidget::initializeGL()** w **glwidget.cpp** implementujemy wczytywanie shadera, linkowanie go, oraz pobranie lokacji zmiennych **uniform**.

```
    m_lightLoc[i].ambient = m_program->uniformLocation(QString().asprintf("light[%d].ambient", i));
    m_lightLoc[i].diffuse = m_program->uniformLocation(QString().asprintf("light[%d].diffuse", i));
    m_lightLoc[i].specular = m_program->uniformLocation(QString().asprintf("light[%d].specular", i));
    m_lightLoc[i].isActive = m_program->uniformLocation(QString().asprintf("light[%d].isActive", i));
    m_lightLoc[i].attenuation = m_program->uniformLocation(QString().asprintf("light[%d].attenuation", i));
}

m_program->release();

m_program_hud = new QOpenGLShaderProgram;
m_program_hud->addShaderFromSourceFile(QOpenGLShader::Vertex, "resources/shader_hud.vs");
m_program_hud->addShaderFromSourceFile(QOpenGLShader::Fragment, "resources/shader_hud.fs");
m_program_hud->bindAttributeLocation("vertex", 0);
m_program_hud->bindAttributeLocation("normal", 1);
m_program_hud->bindAttributeLocation("uvCoord", 2);
m_program_hud->link();

m_program_hud->bind();

m_resolutionLoc_hud = m_program_hud->uniformLocation("resolution");
m_color_hud = m_program_hud->uniformLocation("color");
m_hasTextureLoc_hud = m_program_hud->uniformLocation("hasTexture");
m_rectangleLoc_hud.xPos = m_program_hud->uniformLocation("rect.xPos");
m_rectangleLoc_hud.yPos = m_program_hud->uniformLocation("rect.yPos");
m_rectangleLoc_hud.width = m_program_hud->uniformLocation("rect.width");
m_rectangleLoc_hud.height = m_program_hud->uniformLocation("rect.height");

m_program_hud->release();

lastUpdateTime = 0;
timer.start();

CMesh::loadAllMeshes();
```

HUD - GLWidget

- Należy także dodać strukturę **RectangleLocStruct** do klasy **GLWidget** w **glwidget.cpp**.

```
struct MaterialLocStruct
{
    int ambient;
    int diffuse;
    int specular;
    int shininess;
};

struct RectangleLocStruct
{
    int xPos;
    int yPos;
    int width;
    int height;
};

struct Light
{
    QVector3D position;
    QVector3D ambient;
    QVector3D diffuse;
    QVector3D specular;
    bool isActive = false;
    float attenuation;
};
```

HUD - GLWidget

- Zaimplementujmy funkcję **GLWidget::setRectangle()** w **glwidget.cpp**.

```
void GLWidget::setRectangle(float xPos, float yPos, float width, float height, QVector3D color, QOpenGLTexture* texture)
{
    m_program_hud->setUniformValue(m_rectangleLoc_hud.xPos, xPos);
    m_program_hud->setUniformValue(m_rectangleLoc_hud.yPos, yPos);
    m_program_hud->setUniformValue(m_rectangleLoc_hud.width, width);
    m_program_hud->setUniformValue(m_rectangleLoc_hud.height, height);
    m_program_hud->setUniformValue(m_color_hud, color);

    if(texture != nullptr)
    {
        m_program_hud->setUniformValue(m_hasTextureLoc_hud, 1);
        texture->bind();
    }
    else
    {
        m_program_hud->setUniformValue(m_hasTextureLoc_hud, 0);
    }
}
```

- Potrzebuje jeszcze uzupełnić pole **m_resolution** faktyczną rozdzielczością okna. Możemy ją wydobyć w metodzie **GLWidget::resizeGL()** w pliku **glwidget.cpp**

```
void GLWidget::resizeGL(int w, int h)
{
    m_proj.setToIdentity();
    m_proj.perspective(60.0f, GLfloat(w) / h, 0.01f, 100.0f);

    m_resolution = QVector2D(w, h);
}
```

HUD - paintHUD

- Ostatnim krokiem jest implementacja metody **paintHUD()** w klasie **GLWidget**, która będzie renderowała nasz HUD.
- Funkcję deklarujemy w klasie **GLWidget** w **glwidget.h**.

```
void setTransforms(void);
void updateGL();
void paintHUD();
```

- Oraz tworzymy jej deklarację w **glwidget.cpp**.

```
void GLWidget::paintHUD()
{
    CMesh* rectMesh = CMesh::m_meshes["rect"];

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC0_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    m_program_hud->bind();
    m_program_hud->setUniformValue(m_resolutionLoc_hud, m_resolution);

    // Elementy HUDu

    glDisable(GL_BLEND);
    m_program_hud->release();
}
```

HUD - paintHUD

- Utworzoną funkcję **GLWidget::paintHUD()** używamy na końcu funkcji **GLWidget::paintGL()** w **glwidget.cpp**.

```
            m_world.rotate(0, 0, 0, 1);
            m_world.scale(QVector3D(1 ,1 ,1));
            setTransforms();
            collisionTrianglesMesh.render(this, i * 3, triangle.groupSize * 3);
        m_world = worldMatrixStack.pop();

        i += triangle.groupSize;
    }

    m_program->release();

    float timerTime = timer.elapsed() * 0.001f;
    float deltaTime = timerTime - lastUpdateTime;
    if(deltaTime >= (1.0f / FPS)) {
        updateGL();
        lastUpdateTime = timerTime;
    }

    paintHUD();

    update();
}
```

HUD - paintHUD

- Teraz wystarczy w funkcji **GLWidget::paintHUD()** dodawać elementy HUD.
- Przykład prostego prostokątu na pozycji [30, 10] o rozmiarze 200 na 100 o kolorze żółtym:

```
void GLWidget::paintHUD()
{
    CMesh* rectMesh = CMesh::m_meshes["rect"];

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    m_program_hud->bind();
    m_program->setUniformValue(m_resolutionLoc_hud, m_resolution);

    setRectangle(30, 30, 200, 100, QVector3D(1, 1, 0), nullptr);
    rectMesh->render(this);

    glDisable(GL_BLEND);
    m_program_hud->release();
}
```

HUD - paintHUD

HUD - paintHUD

- Przykład wyświetlenia ikony
  - ikona pobrana ze strony https://game-icons.net/

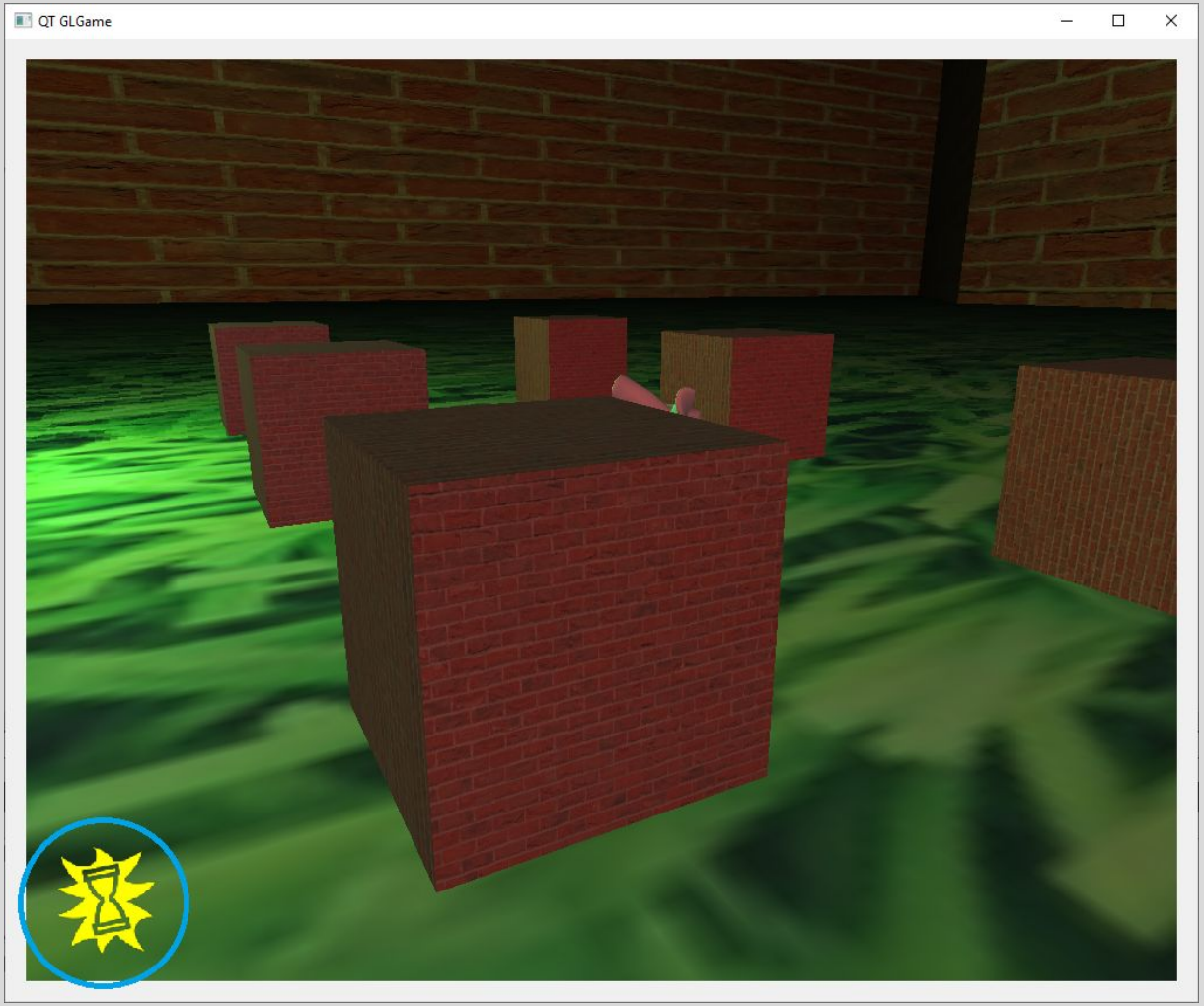
```
void GLWidget::paintHUD()
{
    CMesh* rectMesh = CMesh::m_meshes["rect"];

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    m_program_hud->bind();
    m_program->setUniformValue(m_resolutionLoc_hud, m_resolution);

    setRectangle(20, 680, 100, 100, QVector3D(1, 1, 0), TextureManager::getTexture("hourglass"));
    rectMesh->render(this);

    glDisable(GL_BLEND);
    m_program_hud->release();
}
```

  - Należy pamiętać o dodaniu tekstury do folderu **builds/resources/** oraz wczytaniu jej w **TextureManager::init()**.

```
void TextureManager::init()
{
    m_textures["brick"] = new QOpenGLTexture(QImage("resources/brick.jpg"));
    m_textures["grass"] = new QOpenGLTexture(QImage("resources/grass.jpg"));
    m_textures["wood"] = new QOpenGLTexture(QImage("resources/wood.jpg"));
    m_textures["hourglass"] = new QOpenGLTexture(QImage("resources/hourglass.png"));
}
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

HUD - paintHUD



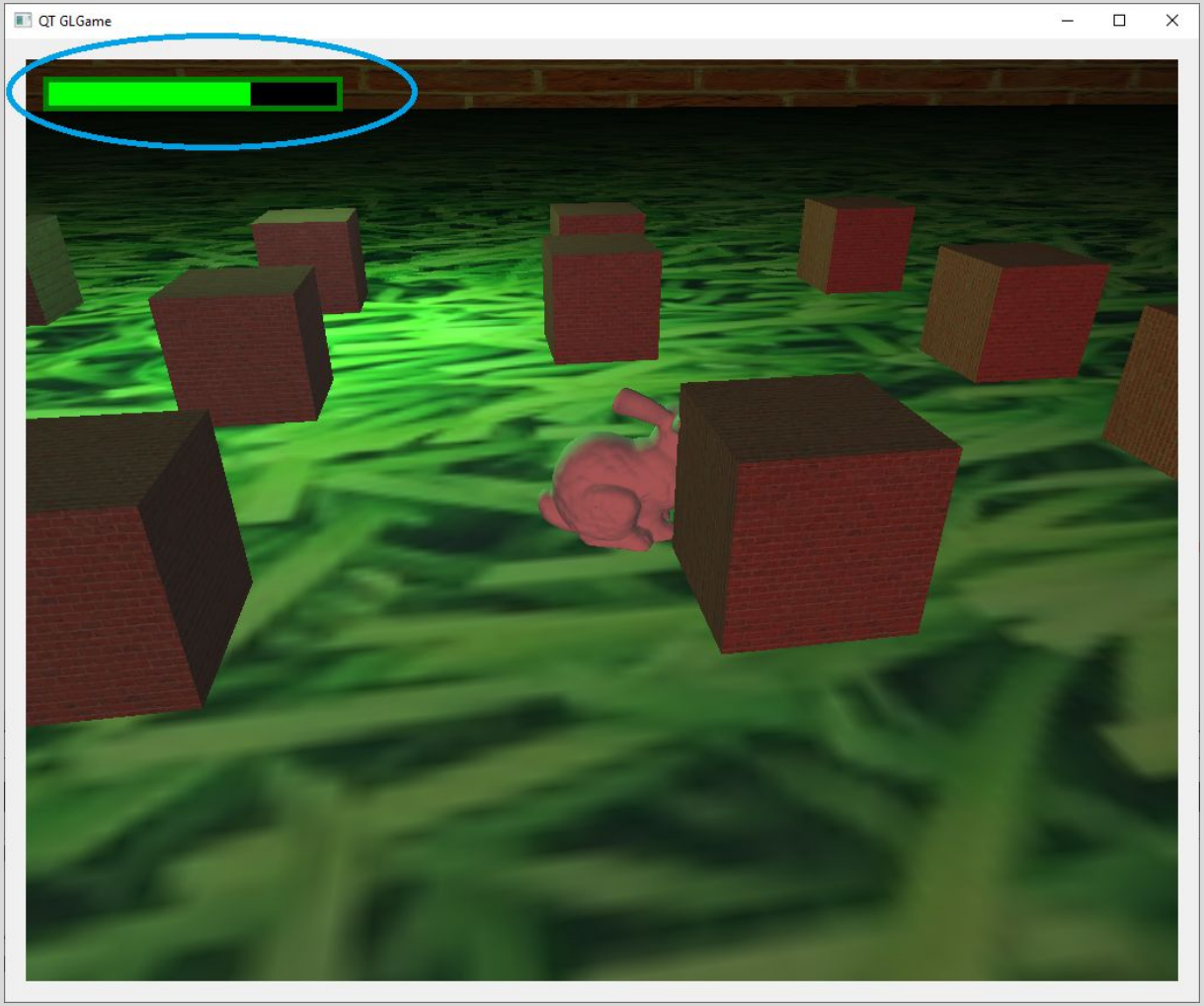
Wydział
Informatyki

HUD - paintHUD

- Przykład paska życia dla gracza :)


```
void GLWidget::paintHUD()
{
    CMesh* rectMesh = CMesh::m_meshes["rect"];

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    m_program_hud->bind();
    m_program->setUniformValue(m_resolutionLoc_hud, m_resolution);

    float playerHP = 70;
    float playerHPMax = 100;
    float posX = 20;
    float posY = 20;
    float width = 250;
    float height = 20;
    float frame = 5;

    setRectangle(posX, posY, width*playerHP/playerHPMax, height, QVector3D(0, 1, 0), nullptr);
    rectMesh->render(this);

    setRectangle(posX+width*playerHP/playerHPMax, posY, width*(1-playerHP/playerHPMax), height, QVector3D(0, 0, 0), nullptr);
    rectMesh->render(this);

    setRectangle(posX-frame, posY-frame, width+frame*2, height+frame*2, QVector3D(0, 0.5, 0), nullptr);
    rectMesh->render(this);

    glDisable(GL_BLEND);
    m_program_hud->release();
}
```

HUD - paintHUD



HUD - paintHUD

- Przykład tekstu.
- Należy do funkcji **CMesh::render()** dopisać zwolnienie Vertex Array Object.

```
void GLWidget::paintHUD()
{
    CMesh* rectMesh = CMesh::m_meshes["rect"];

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    m_program_hud->bind();
    m_program->setUniformValue(m_resolutionLoc_hud, m_resolution);

    QPainter painter(this);
    painter.setPen(QColor(255.0f, 255.0f, 255.0f, 255.0f));
    painter.setFont(QFont("Helvetica", 26));
    painter.drawText(100, 100, "TEST");
    painter.end();

    glDisable(GL_BLEND);
    m_program_hud->release();
}
```

HUD - paintHUD

- Przykład tekstu.
- Należy do funkcji **CMesh::render()** dopisać zwolnienie Vertex Array Object.

```
void CMesh::render(GLWidget* glWidget)
{
    m_vao_binder->rebind();
    glWidget->glDrawArrays(m_primitive, 0, vertexCount());
    m_vao_binder->release();
}

void CMesh::render(GLWidget* glWidget, int offset, int count)
{
    m_vao_binder->rebind();
    glWidget->glDrawArrays(m_primitive, offset, count);
    m_vao_binder->release();
}
```

HUD - paintHUD