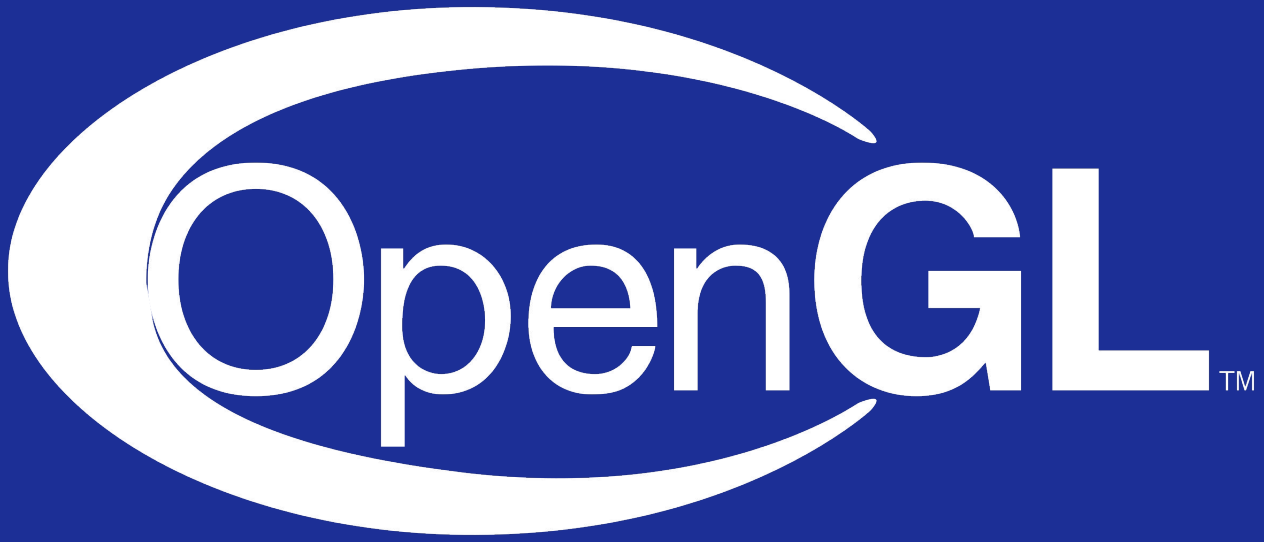
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

OpenGL : Oświetlenie



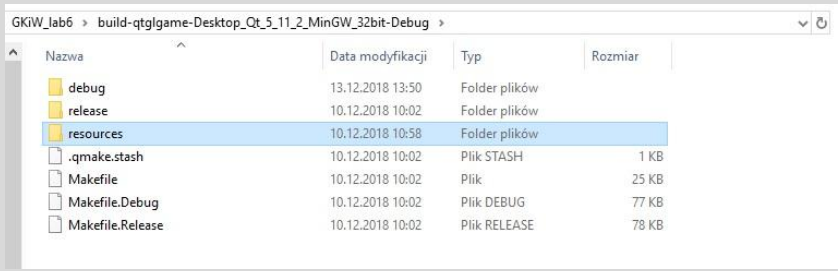
mgr inż. Michał Chwesiuk
mgr inż. Tomasz Sergej
inż. Patryk Piotrowski

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Folder z plikami zewnętrznymi (resources)

- Po odpaleniu przykładowego projektu, nie uruchomi się on poprawnie.
- Powodem jest brak programów cieniujących. Znajdują się one w plikach shader.vs (vertex shader) oraz shader.fs (fragment shader).
- Pliki te należy wkleić do zbudowanego projektu, do folderu resources, który należy utworzyć w folderze wskazywanym podczas kompilacji przez środowisko QT.



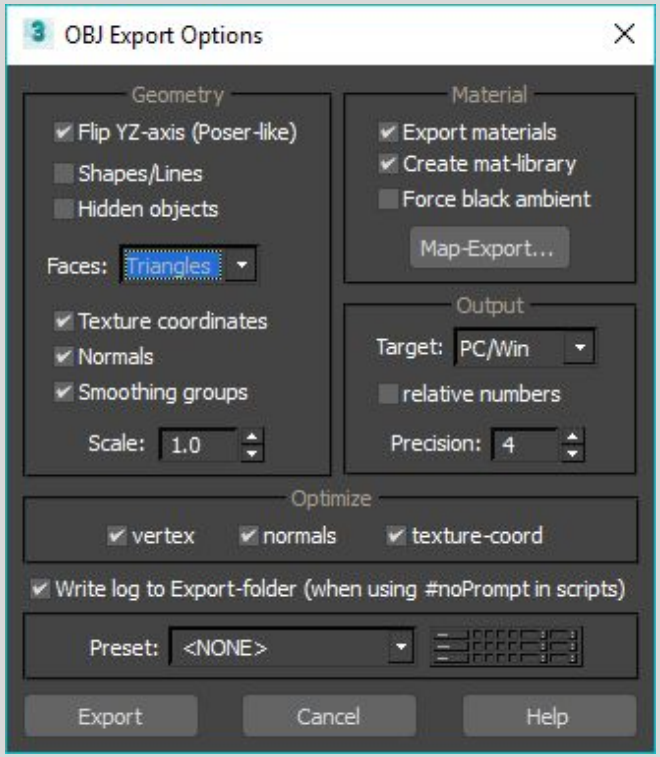
Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Konwertowanie pliku .3ds do .obj

- Do projektu został dołączony plik bunny.3ds i bunny.blend, który należy skonwertować do formatu .obj i dodać do stworzonego wcześniej folderu resources (wybrać plik 3ds albo blend w zależności od zainstalowanego środowiska).

File ->
Export ->
Wavefront (.obj)

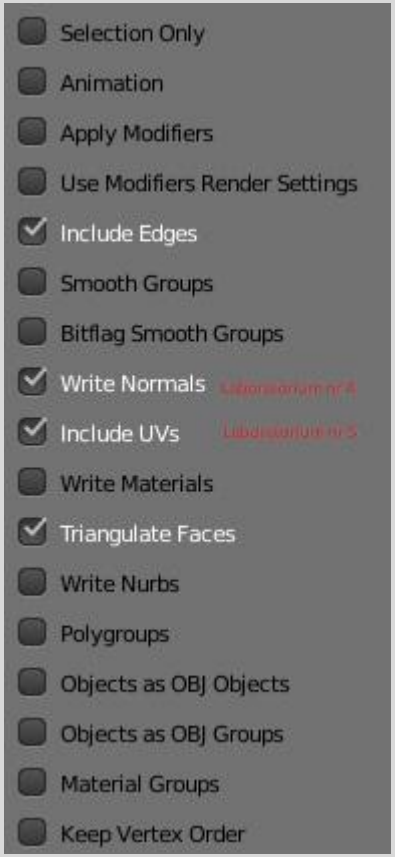


Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Konwertowanie pliku .blender do .obj

File ->
Export ->
Wavefront (.obj)



Wydział
Informatyki

Wczytanie OBJ

- Projekt został rozbudowany o możliwość wczytywania plików obj. Aby wczytać plik należy odpowiednio użyć funkcji generateMeshFromObjFile() wewnątrz **initializeGL()**

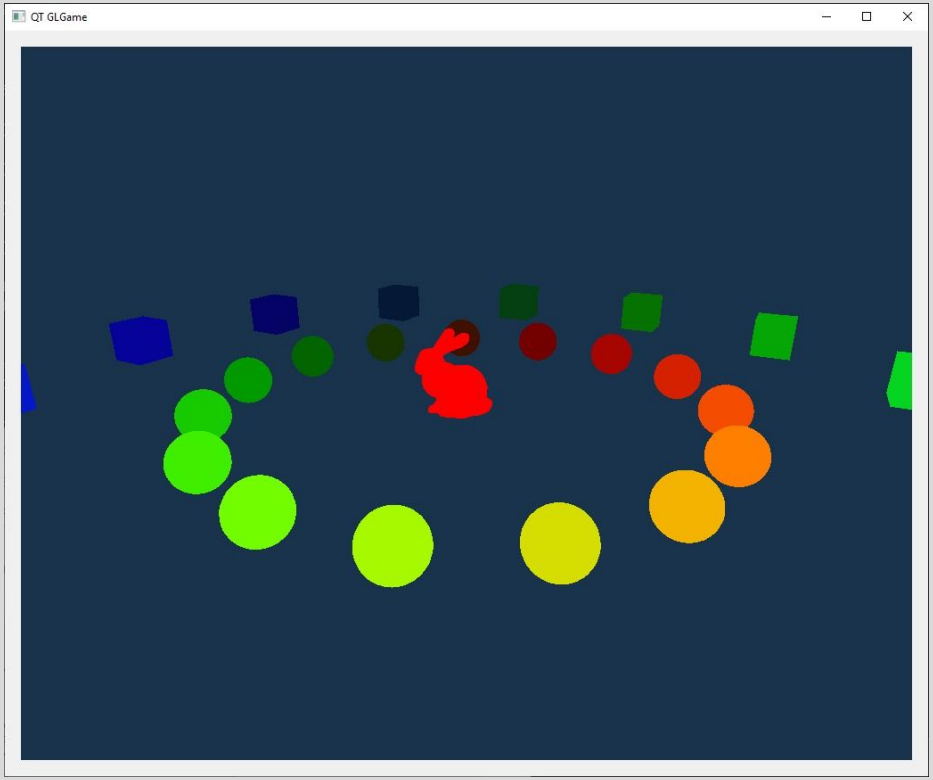
```
m_meshes.insert("Bunny", new CMesh);
m_meshes["Bunny"]->generateMeshFromObjFile("resources/bunny.obj");
```

- Aby wyrenderować wczytany obiekt trzeba postąpić tak samo jak z innymi wygenerowanymi obiektami.

```
worldMatrixStack.push(m_world);
    m_world.translate(0.0f, 0.0f, 0.0f);
    m_world.scale(QVector3D(0.1f, 0.1f, 0.1f));
    setTransforms();
    m_program->setUniformValue(m_modelColorLoc,QVector3D(1.0f, 1.0, 1.0));
    m_meshes["Bunny"]->render(this);
m_world = worldMatrixStack.pop();
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wczytanie OBJ - efekt końcowy



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

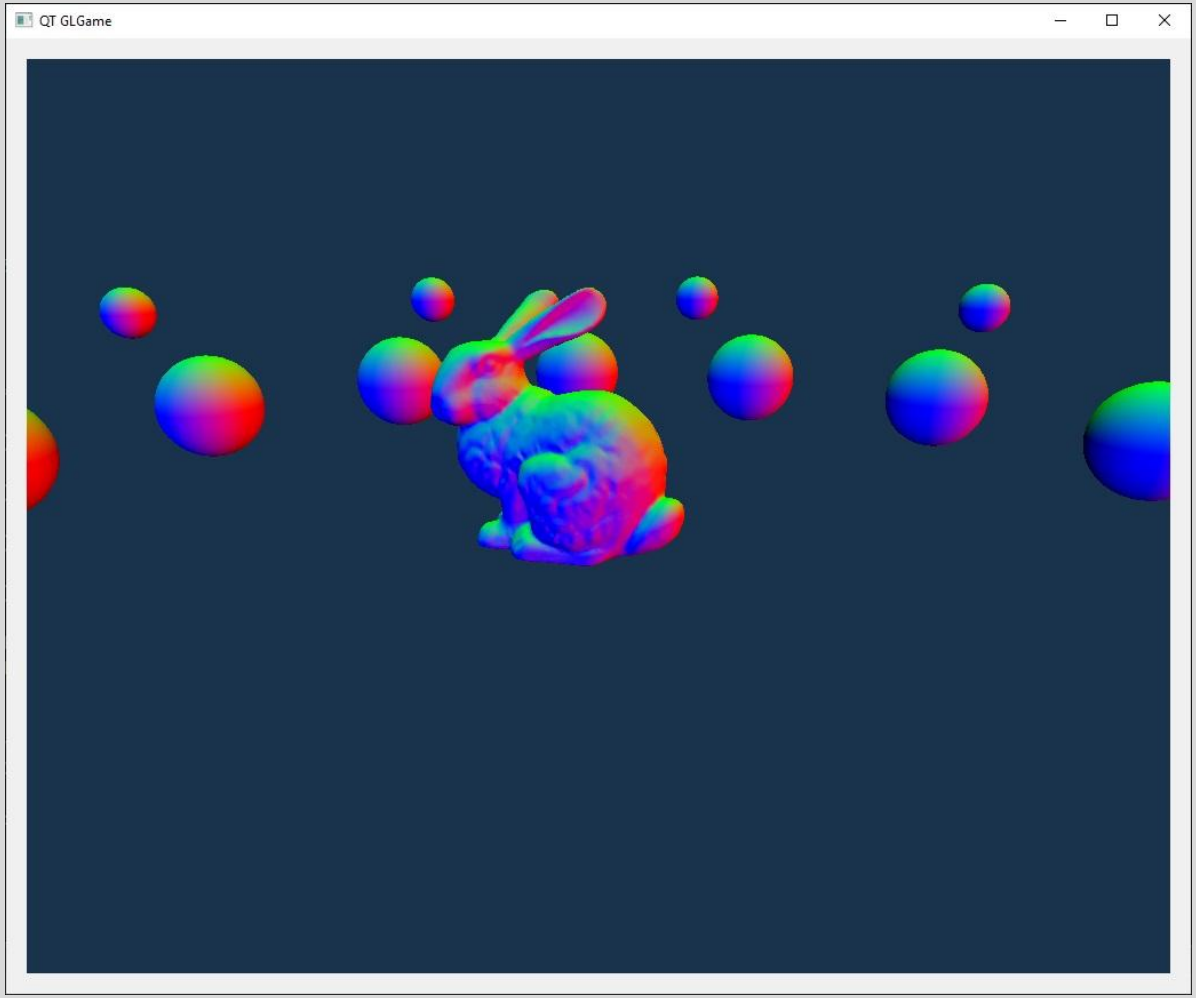
Wektory normalne

- Pierwszym etapem jest uwzględnienie wektorów normalnych. Są one już zaimplementowane w generowaniu sześcianu, sfery, a także w wczytanym projekcie. W programie cieniującym jest on wczytywany jako atrybut normal.
- Atrybuty występują jedynie wewnątrz vertex shader'a, aby zmodyfikować na podstawie jego kolor, należy przenieść go do fragment shader'a, interpolując go.
- Do vertex oraz fragment shadera należy dodać nową zmienną typu varying, czyli taką którą można przekazywać między shaderami.
  - `varying highp vec3 fragNormal;`
- Następnie przekazujemy w vertex shaderze atrybut *normal* wewnątrz funkcji *main()*
  - `fragNormal = normal;`
- Następnie wewnątrz funkcji *main()* w fragment shaderze przypisujemy kolor piksela na wektor normalny.
  - `gl_FragColor = vec4(fragNormal, 1.0);`
- Skopiować edytowane shadery do folderu resource.

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wektory normalne - efekt końcowy



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Oświetlenie 1/3

- Mając wektory normalne wewnątrz fragment shadera, można je wykorzystać aby kolor obiektu był zależny od położenia względem źródła światła.
- Do vertex i fragment shader'a należy dodać strukturę *Light* oraz zmienną uniform jej typu.

```
struct Light {
        highp vec3 position; // Pozycja źródła światła
        highp vec3 ambient; // Składowa światła niezależna od kąta padania na powierzchnię
        highp vec3 diffuse;   // Składowa światła zależna od kąta padania na powierzchnię
};
uniform Light light;
```

- Następnie w celu przeprowadzenia dalszych obliczeń, musimy obliczyć wierzchołków w przestrzeni świata. Aby to uczynić dodajemy zmienną *variable* w obydwóch shaderach

```
varying highp vec3 vertexWorldSpace;
```

- W funkcji *main()* w vertex shaderze interpolujemy tą zmienną do fragment shadera (aby były w przestrzeni świata, mnożymy wierzchołek przez macierz modelu).

```
vertexWorldSpace = (modelMatrix * vertex).xyz;
```

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Oświetlenie 2/3

- Mamy już wszystkie dane potrzebne do wykonania prostego równania oświetlenia.
- W funkcji *main()* we fragment shaderze wykonujemy obliczenia.
  - Obliczamy wektor N, znormalizowany wektor normalny.
    ```
    highp vec3 N = normalize(fragNormal);
    ```
  - Obliczamy wektor L, znormalizowany wektor od pozycji wierzchołka do pozycji źrodła światła.
    ```
    highp vec3 L = normalize(light.position - vertexWorldSpace);
    ```
  - Obliczamy “cosinus” kąta pomiędzy wektorami N i L za pomocą iloczynu skalarnego.
    ```
    highp float cosNL = dot(N, L);
    ```
  - Nanosimy poprawkę na obliczony “cosinus” dzięki użyciu operacji tzw. clamplingu (przycinania), aby był w przedziale [0, 1].
    ```
    cosNL = clamp(cosNL, 0.0, 1.0);
    ```
  - Obliczamy składową ambient dla fragmentu.
    ```
    highp vec3 colorAmb = modelColor * light.ambient;
    ```
  - Obliczamy składową diffuse dla fragmentu.
    ```
    highp vec3 colorDif = modelColor * light.diffuse * cosNL;
    ```
  - Sumujemy te dwie składowe, przy okazji je clampując.
    ```
    highp vec3 colorFull = clamp(colorAmb + colorDif, 0.0, 1.0);
    ```
  - Przypisujemy wyliczoną sumę do końcowego koloru fragmentu.
    ```
    gl_FragColor = vec4(colorFull, 1.0);
    ```
- Pamiętamy o kopiowaniu shaderów do folderu resource.

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Oświetlenie 3/3

- Po napisaniu równania oświetlenia w vertex i fragment shaderze, należy przekazać do nich odpowiednie dane.
- Najpierw pobieramy lokację tych zmiennych w shaderze.
  - W klasie *glWidget* deklarujemy w pliku *glWidget.h* strukturę.

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

  - I obiekt tej struktury.

```
LightLocStruct m_lightLoc;
```

  - W funkcji *initializeGL()* w pliku *glWidget.cpp* pobieramy lokacje i zapisujemy ją do stworzonej struktury.

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

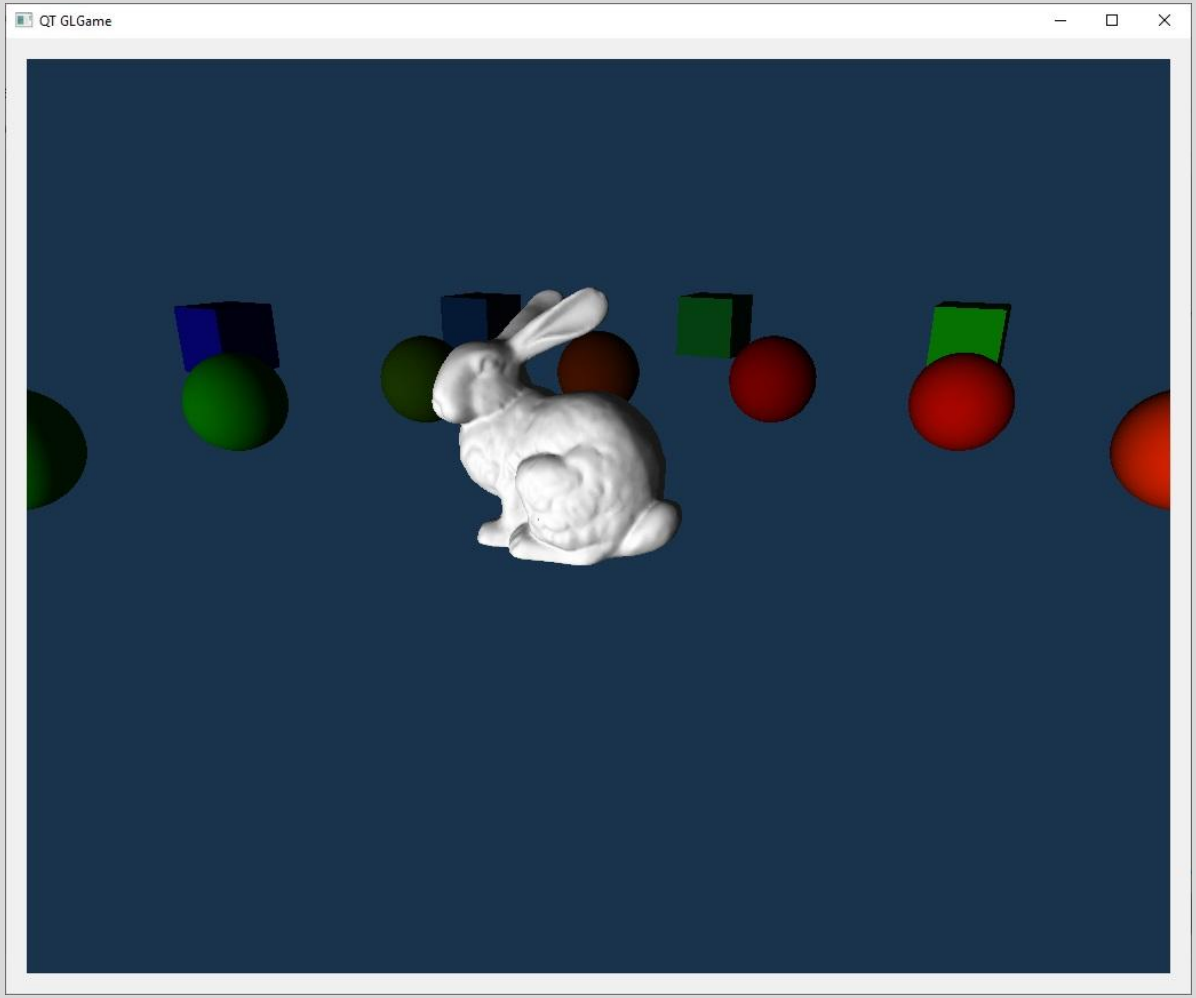
  - W funkcji *paintGL()* (na początku) możemy ustawić parametry światła.

```
m_program->setUniformValue(m_lightLoc.position, QVector3D(0.0f, 0.0f, 15.0f));
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));
```

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Oświetlenie - efekt końcowy



Wydział
Informatyki

Zadanie

Wersja podstawowa - 0.75pkt

- Oświetlenie ambient i diffuse.
- Wczytanie pliku bunny.obj i wyrenderowanie go.
- Wczytanie własnego modelu.
- Poruszające się źródło światła.

Wersja rozszerzona - 1.00pkt

- Wersja podstawowa.
- Zaimplementować więcej źródeł światła.