**Członkowie zespołu:**

Grzegorz Świrski

Jakub Zembik

Marko Łaboszczak

Andrzej Datczuk

Paweł Domżalski

Projekt grupowy

Dydaktyczny system prezentujący możliwości renderowania grafiki 3D z wykorzystaniem OpenGL 4 i GLSL

1. **Cel projektu**

Celem projektu jest stworzenie dydaktycznego systemu prezentującego wybrane możliwości trójwymiarowej grafiki komputerowej z wykorzystaniem OpenGL 4 i języka cieniowania GLSL. System ilustruje najważniejsze zagadnienia związane z tworzeniem scen 3D, takie jak: modelowanie sceny, oświetlenie, kolory, cieniowanie, teksturowanie, przekształcenia. W systemie wykorzystane zostały biblioteki udostępnione wraz z podręcznikiem „OpenGL SuperBible 6th Edition”.

1. **Wykorzystane technologie, biblioteki i zasoby.**

Projekt został zrealizowany w środowisku *Microsoft Visual Studio 2013* w języku *C++.* Do programowania grafiki 3D wykorzystano interfejs *OpenGL* oraz język cieniowania *GLSL*. Do tworzenia okien wykorzystano bibliotekę *gl3w.* Wykorzystano również bibliotekę *sb6* udostępnioną wraz z podręcznikiem „OpenGL SuperBible” pozwalającą na wczytywanie modeli, shaderów oraz tekstur. Ponadto, wykorzystano również dedykowane modele (o rozszerzeniu .sbm) oraz tekstury w formacie .ktx.

1. **Główne okno aplikacji.**

Aplikacja składa się ze sceny 3D oraz graficznego interfejsu użytkownika. W scenie umieszczonych zostało 27 obiektów, rozmieszczonych względem siebie w taki sposób, że tworzą one „kostkę „ o wymiarach 3x3x3.

Każdy z obiektów ma przypisane indywidualne właściwości, które możemy, niezależnie od innych obiektów, zmieniać. Są to:

* Model,
* Tekstura,
* Wszystkie typy shaderów,
* Rotacja w trzech osiach,
* Skalowanie w trzech osiach.

Dodatkowo, dostępne są ustawienia globalne – dla całej sceny, zawierające:

* Pozycję światła na scenie,
* Albedo materiałów dla światła rozproszonego oraz ‘lustrzanego’,
* Siłę światła lustrzanego,
* Komentarz odnośnie sceny.

Wszystkie te parametry, wraz z obecną pozycją kamery, możemy zapisać do wybranego pliku XML, a następnie wczytać taką konfigurację w dowolnym momecie.

Plik XML jest zorganizowany w następujący sposób:

<app>

<light posX="98" posY="100" posZ="100"/>

<material\_properties diffuse\_albedoX="0.1" diffuse\_albedoY="0.1" diffuse\_albedoZ="0.1" specular\_albedo="0.3" specular\_power="100.5"/>

<camera posX="-10" posY="0" posZ="-4" eyeX="1" eyeY="-0.2666668" eyeZ="0" centerX="0" centerY="0" centerZ="0" upX="0" upY="1" upZ="0" hAngle="766.76941"/>

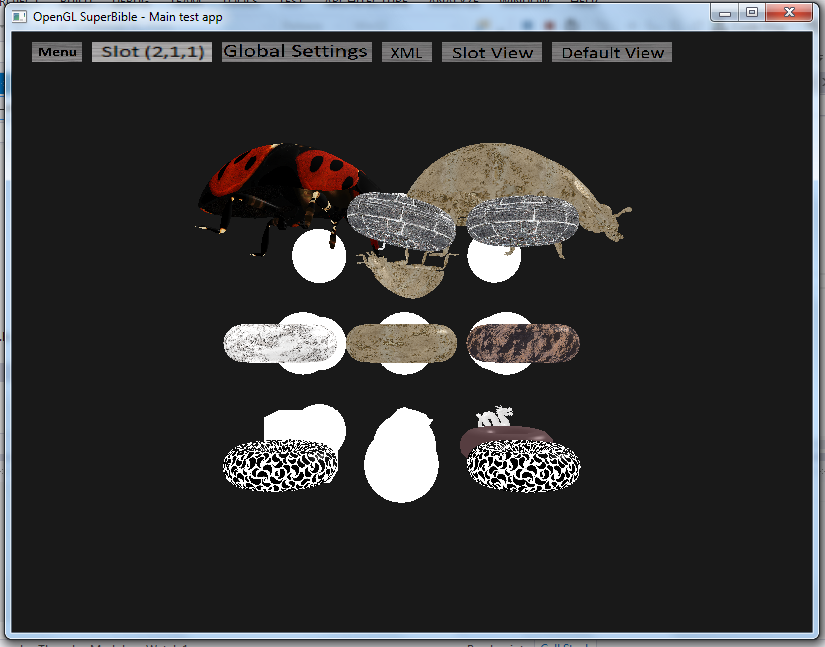
<info comment="FIRST COMMENT"/>

<objects>

<object coordX="0" coordY="0" coordZ="0" model="media/objects/bunny\_1k.sbm" texture="media/textures/floor.ktx" rotX="-0" rotY="0" rotZ="0" scaleX="1" scaleY="1" scaleZ="1" vsPath="media/shaders/default/default.vs.glsl" fsPath="media/shaders/DOF/RENDER.FS.GLSL" gsPath="" tesPath="" tcsPath="" csPath=""/>  
 <object coordX="1" (...)  
 </objects>

</app>

W górnej części górnego okna umieszczono elementy interfejsu użytkownika. Ogólny wygląd aplikacji przedstawiono na poniższym zrzucie ekranu.



Wywołanie głównego okna aplikacji realizowane jest przez poniższą funkcję:

void MainApp::startup()

{

LoadXmlConfig();

myTabPanel->Init(800,600);

// Init scene objects

for(int i = 0; i < OBJECT\_COUNT\_X; i++)

{

for(int j = 0; j < OBJECT\_COUNT\_Y; j++)

{

for(int k = 0; k < OBJECT\_COUNT\_Z; k++)

{

myTabPanel->setXmlParamsStruct( i\*9 + j\*3 + k, mSceneObjects[i][j][k].GetParams() );

myTabPanel->setXmlComment(xmlComment);

mSceneObjects[i][j][k].Startup();

}

}

}

setCameraPositions();

}

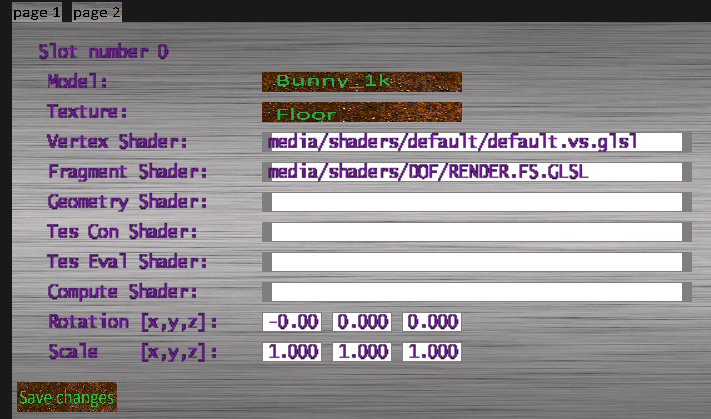
1. **Graficzny interfejs użytkownika oraz realizowane funkcje.**

Użytkownikowi aplikacji udostępnione zostały panele, które szczegółowo omówiono poniżej, wraz z realizowanymi przez nie funkcjami.



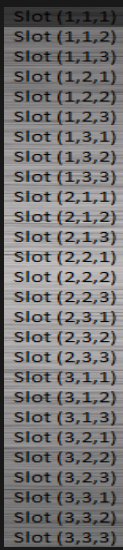
1. **Menu**

W menu użytkownik ma możliwość ustawienia parametrów dla aktualnie wybranego obiektu (wyboru obiektu dla którego ustawiane są parametry dokonuje się z wykorzystaniem listy rozwijanej „Slot”).



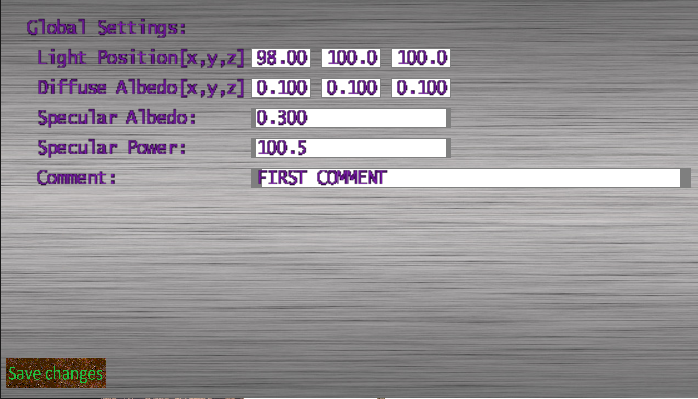
1. **Slot (x, y, z)**

Wybierając obiekt o określonych współrzędnych z listy rozwijanej użytkownik określa, dla którego modelu mają być ustawiane parametry, dostępne w zakładce „Menu”. Listę rozwijaną zaprezentowano na poniższym zrzucie ekranu.



1. **Global Settings**

Użytkownik ma możliwość ustawienia współrzędnych źródła światła („Light Position[x, y, z]), albedo materiałów dla światła rozproszonego oraz ‘lustrzanego’ oraz siłę światła lustrzanego. Możliwe jest również wpisanie komentarza, który zostanie zapisany w pliku konfiguracyjnym xml. Wszystkie parametry ustawiane z tego okna zapisane zostaną w pliku konfiguracyjnym po kliknięciu przycisku „Save changes”.



1. **XML**

Klikając w pole „XML” wyświetlane jest okno, w którym użytkownik ma możliwość wyboru pliku konfiguracyjnego xml z dysku. Istnieje również możliwość zapisania parametrów sceny w nowym pliku xml. Funkcje te uruchamiane są po naciśnięciu przycisków odpowiednio „Load XML” oraz „Save XML”.



1. **Slot View**

Przycisk służy do ustawienia kamery w taki sposób, aby wskazywała ona aktualnie wybrany obiekt (lista „Slot (x,y,z)).

1. **Default View**

Przycisk służy do ustawienia kamery w ustawieniu domyślnym, obejmującym całą scenę 3D.

1. **Wczytywanie plików konfiguracyjnych**

TinyXML 2 jest prostym, małym, wydajnym, parserem plików xml, który łatwo się integruje w aplikacje. TinyXML 1 używał pamięć nie efektywnie i nie sprawdzał się dobrze dla urządzeń mobilnych

Oba parsery posiadają: podobne API, są oparte o DOM i wspierają UTF-8.

Natomiast TinyXML 2 alokuję mniej pamięci (około 40% co TinyXML 1), odczytuję dane 5 razy szybciej niż jego poprzednik, zawiera nowocześniejsze wykorzystanie c++ (przestrzenie nazw), oraz właściwą obsługa białych znaków

TinyXML 1 może raportować błędy parsowania oraz jest dobrze przetestowany I sprawdzony.

1. **Wczytywanie tekstur.**

Wczytywanie tekstur odbywa się z wykorzystaniem biblioteki sb6. Tekstury zapisane są w formacie .ktx, który jest formatem utworzonym specjalnie dla celów dydaktycznych podręcznika „OpenGL SuperBible”. Aby możliwe było nałożenie tekstury w tym formacie na model sbm, musi on zawierać współrzędne tekseli dla odpowiadających im wierzchołków. Niestety, wśród wykorzystywanych przez nas modeli, współrzędne te udostępnione są jedynie w modelach: „Torus\_nrms\_tc” oraz „Ladybug”. Wczytywanie tekstur odbywa się z wykorzystaniem poniższych funkcji:

void SceneObject::LoadTexture()

{

if (!mParams.TexturePath.empty())

mTexture = LoadTextureFromFile(mParams.TexturePath.c\_str());

if (mTexture != 0)

IsTextureLoaded = GL\_TRUE;

}

void SceneObject::UnloadTexture()

{

glDeleteTextures(1, &mTexture);

mTexture = 0;

IsTextureLoaded = GL\_FALSE;

}

GLuint SceneObject::LoadTextureFromFile(string path)

{

GLuint textureID = 0;

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

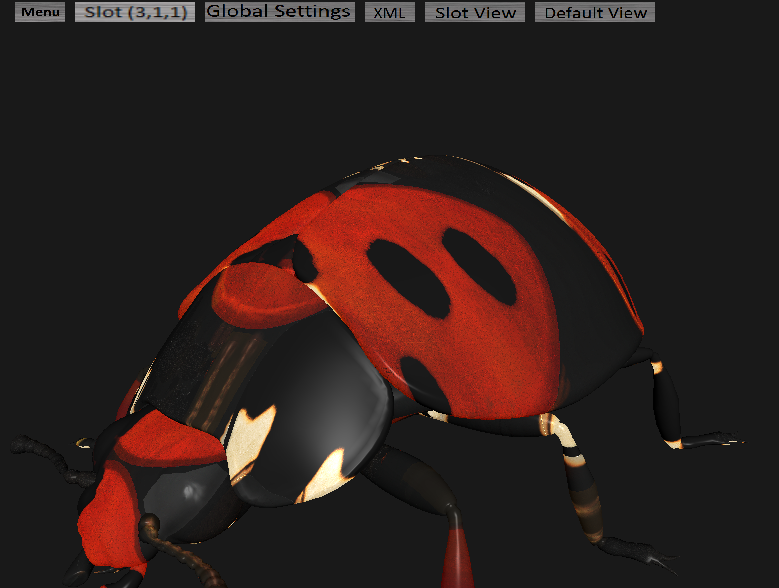
glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

textureID = sb6::ktx::file::load(path.c\_str());

return textureID;

}

Przykładowa tekstura nałożona na model wygląda następująco:



1. **Wczytywanie jednostek cieniowania (shaderów).**

Wczytywanie shaderów wykonywane jest z wykorzystaniem następujących funkcji:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: LoadShaders

// FullName: SceneObject::LoadShaders

// Access: public

// Returns: void

// Qualifier:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool SceneObject::LoadShaders()

{

GLuint fs = 0, vs = 0, gs = 0, tes = 0, tcs = 0;

bool ret = true;

if( !mParams.VertexShaderPath.empty() )

vs = LoadShaderFromFile( mParams.VertexShaderPath, GL\_VERTEX\_SHADER );

if( !mParams.FragmentShaderPath.empty() )

fs = LoadShaderFromFile( mParams.FragmentShaderPath, GL\_FRAGMENT\_SHADER );

if( !mParams.GeometryShaderPath.empty() )

gs = LoadShaderFromFile( mParams.GeometryShaderPath, GL\_GEOMETRY\_SHADER );

if( !mParams.TessControlShaderPath.empty() )

tcs = LoadShaderFromFile( mParams.TessControlShaderPath, GL\_TESS\_CONTROL\_SHADER );

if( !mParams.TessEvaluationShaderPath.empty() )

tes = LoadShaderFromFile( mParams.TessEvaluationShaderPath, GL\_TESS\_EVALUATION\_SHADER );

if (mProgram)

glDeleteProgram(mProgram);

mProgram = glCreateProgram();

if( vs )

glAttachShader( mProgram, vs );

if( fs )

glAttachShader( mProgram, fs );

if( gs )

glAttachShader( mProgram, gs );

if( tcs )

glAttachShader( mProgram, tcs );

if( tes )

glAttachShader( mProgram, tes );

glLinkProgram(mProgram);

//Check for errors

GLint programSuccess = GL\_TRUE;

glGetProgramiv( mProgram, GL\_LINK\_STATUS, &programSuccess );

if( programSuccess != GL\_TRUE )

{

// Linking errors

glDeleteProgram( mProgram );

mProgram = 0;

ret = false;

goto shaderCleanup;

}

uniforms[0].diffuse\_albedo = glGetUniformLocation(mProgram, "diffuse\_albedo");

uniforms[0].specular\_albedo = glGetUniformLocation(mProgram, "specular\_albedo");

uniforms[0].specular\_power = glGetUniformLocation(mProgram, "specular\_power");

shaderCleanup:

if( vs ) glDeleteShader( vs );

if( fs ) glDeleteShader( fs );

if( gs ) glDeleteShader( gs );

if( tcs ) glDeleteShader( tcs );

if( tes ) glDeleteShader( tes );

return ret;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: LoadShaderFromFile

// FullName: SceneObject::LoadShaderFromFile

// Access: public

// Returns: GLuint

// Qualifier:

// Parameter: const string path

// Parameter: GLenum shaderType

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GLuint SceneObject::LoadShaderFromFile( const string path, GLenum shaderType )

{

GLuint shaderID = 0;

string shaderString;

ifstream sourceFile( path.c\_str() );

if( sourceFile )

{

// Copy to string

shaderString.assign( istreambuf\_iterator<char>(sourceFile), istreambuf\_iterator<char>() );

const GLchar\* shaderSource = shaderString.c\_str();

shaderID = glCreateShader( shaderType );

glShaderSource( shaderID, 1, (const GLchar\*\*)&shaderSource, NULL );

glCompileShader( shaderID );

//Check shader for errors

GLint shaderCompiled = GL\_FALSE;

glGetShaderiv( shaderID, GL\_COMPILE\_STATUS, &shaderCompiled );

if( shaderCompiled != GL\_TRUE )

{

// Compiling error

glDeleteShader( shaderID );

shaderID = 0;

}

}

else

{

// Error opening file

}

return shaderID;

}

1. **Zmiana parametrów oświetlenia, zmiana ustawienia kamery, zmiana skali i rotacji modeli.**

Zmiany wymienionych parametrów wykonywane są poprzez odczytanie odpowiednich wartości z pliku xml. Wartości tych parametrów wpisywane są do odpowiednich pól struktury „SceneObjectParams” zaprezentowanej poniżej.

struct SceneObjectParams

{

// General

string ModelPath;

string TexturePath;

vmath::vec3 Rotation;

vmath::vec3 Scale;

vmath::uvec3 Coords; // Position in the 'cube'

// Light

vmath::vec3 LightPosition;

vmath::vec3 DiffuseAlbedo;

float SpecularAlbedo;

float SpecularPower;

// Shaders

string VertexShaderPath;

string FragmentShaderPath;

string GeometryShaderPath;

string TessControlShaderPath;

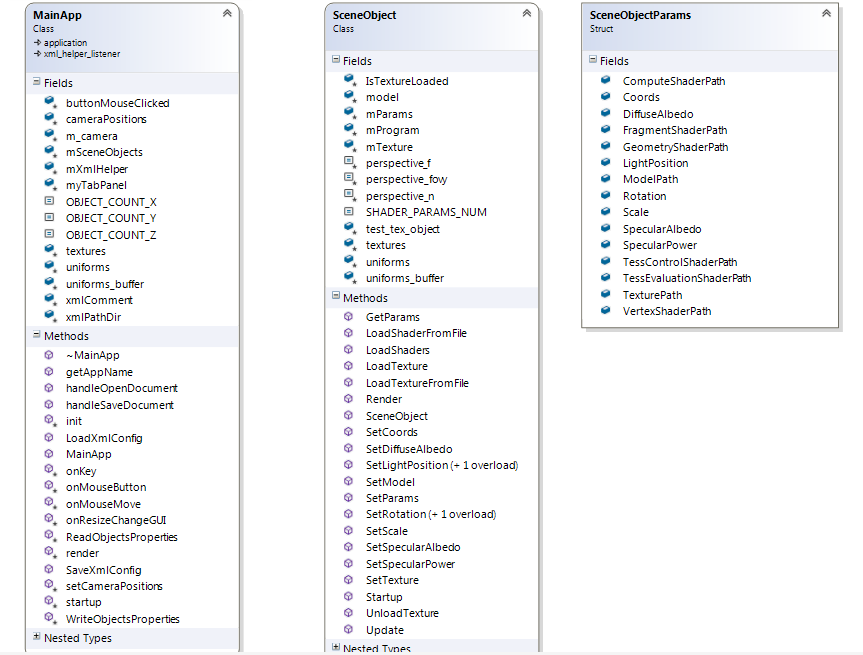
string TessEvaluationShaderPath;

string ComputeShaderPath;

};

1. **Architektura aplikacji**

Na rdzeń aplikacji, wyłączając GUI, składają się dwie główne klasy: MainApp i SceneObject, reprezentujące odpowiednio aplikację i indywidualny obiekt na scenie, struktura SceneObjectParams przetrzymująca parametry każdego z obiektów oraz wcześniej wspomniana pomocnicza klasa xml\_helper, ułatwiająca obsługę plików XML.



1. **Podsumowanie**

Zostały zrealizowane założenia projektowe dotyczące aplikacji. Zaprezentowano podstawowe aspekty programowania grafiki 3D z wykorzystaniem OpenGL oraz GLSL, takie jak: wczytywanie modeli, shaderów, tekstur, modelowanie oświetlenia.