Computer Defender
Documentation for Assignment 1

Markus Scherer, Lukas Köll

Implementation

Freely movable Camera

For a freely movable Camera we implemented a Camera class that has methods such as "calculateViewMatrix()" and "calculateProjectionMatrix()" depending on the Cameras current position, orientation and game state. Right now, it generally follows the movement of the player character with an offset so that it always looks over the player characters shoulders. It is also possible to drag the mouse on the screen on the X-Axis to make the camera orbit around the player. It will then automatically sweep back into its original position.

Moving Objects

We have a class called GeometryObject which has vec3 attributes such as position, orientation and scale. When an object moves, its according properties are changed. During the rendering process we then calculate the according transformation matrices and apply them.

Texture Mapping

We are modeling our characters in Blender and generate the UV Maps there. In our Engine, we then read out the UV coordinates using Assimp and store them together with the vertex positions.

Simple Lighting and Materials

Every GeometryObject is associated to a Material. Our Engine uses a Material base class which is associated with a Shader program and has methods preDraw(), postDraw() and setUniforms(). These methods are called in the GeometryObject in the draw() method. This design should assure that the required uniforms for a certain shader are always set and still be flexible.

Controls

Control	Action
WASD	Move the player forward/backward turn left/right
TAB	starts Debug mode
CTRL-TAB	Disables Debug mode
Right-Click	Shoot in look direction
Spacebar	jump
Drag mouse horizontally	Make Camera orbit around android

Features

- Multiple light sources
- Shadow Mapping
- Shooting
- Orbiting Camera (automatically returns to original position)
- Flexible Level Design
- easily extendable due to Event System
- Collision Detection
- Jumping
- Projectile Explosion

Lighting

We currently use 4 independant point light sources which illuminate all objects. A Light object does basically inherit all properties from a SceneObject and adds things like vec3 color. You can decide wether or not an object will be textured or not by using the according Material. The Material name is read from the imported model file. (StandardMaterial for example will render a white Object with blinn/phong illumination but no texture while LightedTextureMaterial will use a Texture and also use the lighting information).

Illumination is done via Blinn model and shading via Phong right now. Currently, all parameters such as ambient, diffuse and specular reflection coefficient are static, but we plan to read them from the material properties in the future.

Additional Libraries

We used the following external libraries:

Library	**Purpose**	**URL**
Assimp	Model and Material Loading	http://assimp.sourceforge.net/
GLM	OpenGL compatible vectors and matrices	http://glm.g-truc.net/
FreeImage	Image loading	http://freeimage.sourceforge.net/
GLFW	Window Handling	http://www.glfw.org/
GLEW	easier handling of OpenGL	http://glew.sourceforge.net/