MAYARI

Controls

Controls are implemented with polling and a keybuffer. The movement uses the deltatime of the last frame to preserve frame independency.

Key	Effect	Use
W,A,S,D	Move Mayari	game
Space	Float (keep pressed)	game
Right Mouse Button	Control Camera	game
C	Change Day/Night	game
ESC	Quit Game	game
F2	show FPS	debug
F3	toggle wireframe	debug
F4	toggle texture sampling quality	debug
F5	toggle mipmapping quality	debug
F6	toggle BulletDebugShader	debug
F7	toggle postProcessing effects	debug
F8	toggle viewfrustum-culling	debug
F9	toggle transparency	debug
F10	toggle shadows	debug
B	toggle sun intensity (white - red)	test
M	faster move speed	test

Lights

Two directional light sources are implemented (sun, moon) with different colors (only attributes: vec3 direction, vec3 intensity). Only one can be active during the game (day, night). The skybox will be adjusted to the current time of the day.

Camera

One camera object that is locked behind Mayari, following it in all movement. Camera can be rotated in pitch (limited) and yaw by clicking the right mouse button and dragging in the corresponding direction.

Camera rotation is implemented by using quaternions.

Models

ASSIMP is used for loading .obj together with .mtl files as well as .dae files for rigged models. A model consists of several meshes. For each mesh retrieved by ASSIMP, the provided structure is converted into a simple Mesh object and added to the meshes inside the model. Every mesh saves the vertices (position + normal + texCoords), indices and the textures itself. The textures are loaded via SOIL. When loaded, the textureID used by openGL is saved inside the struct and used later to be drawn.

All models were created using Maya 2016 (including the terrain). Simple lambert Materials were assigned since only the texture is needed in toonShading.

Effects

- Vertex Skinning (2)
- Cel-Shading (0.5)

- + Contours: Edge Detection (1)
- Shadow Volumes (1.5)
- => 5 points

Moving Objects

The main character is movable by pressing the W/A/S/D keys in order to move it forward/backwards and rotate left/right (implemented with quaternions).

Moving is delegated by Bullet to keep the models position and its corresponding collisionBox updated.

Vertex Skinning

Skeletal Animation is showcased with the main character, which is moving her 'tail' constantly. The implementation follows the oglDev tutorial (http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html) by using assimp for importing skeletal information. The objects animation and bone information are stored in seperate datastructures, mapping bone IDs and weights to vertices IDs. Each frame, the bone transformations are calculated by iterating over the entire node tree of a mesh concatenating the matrices from leaves up to the root node. A single animation is shown, but multiple animations are possible.

Shading and Textures

The main approach for shading scene objects is the xToon shader, as described in [BARLA P., THOLLOT J., MARKOSIAN L.; "X-Toon: An Extended Toon Shader", in *International Symposium on Non-Photorealistic Animation and Rendering*, 2006.]. It extends classic cel-shading techniques, which can be described as using a dimensional texture as a color lookup according to the fragments position relative to a light source, by adding a second dimension for describing the detail of the current fragment. This requires 2D textures with mostly colored gradients, with the x-Dimension defining the tone (relativ to light position) and the y-Dimension describing the detail. The calculation of the detail differs for different effects.

shader Uses Phong Illumination model in the fragment shader that works on standard textures. Is only used with the big statue at the beginning of the world, to showcase the requirements for texture mapping/sampling quality.

xToonSlt Uses xToon to simulate a silhouette/backlighting effect. Detail is calculated by the fragments normal and the view vector: $|n \cdot v|^r$, where r describes the magnitude of the effect (higher for the main character at night, letting it 'glow').

terrainShader Uses the xToon to simulate a Level-of-Detail effect. The calculation for the detail is depth-based, defining a distance from the eye for when to decrease the level of detail.

textShader Is used to draw quads on the screen, containing the letters and an alpha component.

debugShader Used to debug bullet physics. Shows the collision boxes for scene objects.

skyboxShader used to draw the skybox.

screenShader Renders a screenfilling quad to the default framebuffer. Used for postprocessing effects (edge detection), using the custom framebuffer as its texture.

quadShader Same as screenShader but without postProcessing effect. Simply render the texture of the custom framebuffer.

skinningShader Used for vertex skinning. Calculates the vertices position according to their influencing bones in the vertex shader. Fragment shader stays the same xToonSlt.

shadowShader Used for drawing the shadowVolumes on the screen.

Edge Detection

To draw contours to the screen, post processing effects for edge detection are used. First, the scene is rendered as usual in a custom framebuffer. This is then used as a texture for the default framebuffer. Everything else is calculated in the the screenShader. The vertex shader just passes on the tex coordinates and the position (discarding the depth info since we are rendering to a quad). The fragment shader then uses a sobel operator to determine edges in the passed texture. The length of the magnitude vector (gradient for all colors) determines whether or not to draw a line.

Shadow Volumes

For drawing shadows we use the volumes approach proposed by Everitt and Kilgard 2002 [EVERITT C. and KILGARD M. J.; *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*, online: http://arxiv.org/ftp/cs/papers/0301/0301002.pdf, last visited: 2015-06-20,2002.]. First, the scene is drawn with only ambient light, not taking into account any light sources. Then, all shadow volumes are drawn into the stencil buffer to later decide which areas are illuminated or not. By using additive blending, the scene is drawn again with lighting information, only on allowed pixels defined by the stencil buffer.

Currently the stencil buffer seems to be incremented too much. We ask to get one more week to fix this problem until the game event.

Externals

ASSIMP http://www.assimp.org/
Importer for .obj models

GLFW http://www.glfw.org/
Used for window handling

GLEW http://glew.sourceforge.net/
OpenGl Extension Wrangler

Bullet http://bulletphysics.org/wordpress/
Physics simulation for collision, ray-casting, interactions

GLM http://glm.g-truc.net/0.9.7/index.html
Math library for OpenGl applications

FreeType http://www.freetype.org/
For rendering true type fonts

SOIL http://www.lonesock.net/soil.html
For texture loading from image files

FMOD http://www.fmod.org/
Sound library, not used yet

Gameplay

Mayari has to find her way out of the world that holds her captured.

- win condition: reach the portal on the mountain top in the far right corner (as seen from starting point: straight ahead)
- loose condition: not existing (if you fall in the pit under the bridge, mayari is reseted
- to get out of the starting area: press C to change to night, so the plant vanishes for you to pass
- when going ahead, another plant blocks the way
- instead turn left before that to the pit

- change to night: a bridge appears
- cross the bridge
- in the upperleft corner, a hint tells you, that you can shortly float by pressing and holding space
- go up the plateau in the middle with the boulders on it
- roll the boulders down in the pit before the upper right hill
- float up on the boulders
- change to night to let the stairs appear
- walk up fast and float while day, and with it a plant comes back to float upon.
- float on the hill and walk sideways to get up
- touch the orange portal

Debug:

- press M to change mayaris move speed; thanks to physics, you are kind of flying when you run up any kind of ramp
  - this way you can get an overview of the terrain and fly right to the portal