Mandatory Assignment 3

Anders Fog Bunzel, 20112293
Mark Medum Bundgaard, 20112423
Mikkel Brun Jakobsen, 20114457

Tuesday 11th August, 2015

1 Usage

You can access the game at `http://silentfeet.rocks/`.

You can select different types of blocks by pressing '1' through '7' on the keyboard.
You can toggle flying mode by pressing 'f'.
You can toggle map mode by pressing 'm'.

2 Overview of our code

Our code consists of three Javascript files.

2.1 camera.js

Keeps track of our projection and our view matrix. It can be toggled to be in "map mode" to switch between perspective (1st person) or orthogonal (map). We have a very basic collision system that interprets the camera as a point and projects it to the closest "fluid" (air, fire, water) point outside of a solid block.

2.2 whale.js

Responsible for creating a whale from an OBJ file using the OBJ loader from `https://github.com/frenchtoast747/webgl-obj-loader` as well as drawing them.

2.3 minecraft3D.js

Responsible for everything else.

- Window creation.
- Object creation (shader programs, cubes, wireframes, textures)
- Control handling (mouse, keyboard)

- Picking using offscreen rendering onto a texture.
- Drawing cubes/whales.
- etc.

3 Buffers

Our world is split into 16x16x16 chunks, each of which has a blockBufferId (containing block vertices) and a lineBufferId (containing wireframe vertices). The idea was only to update part of the world when blocks are inserted/removed. There is a trade off between the amount of draw calls and the size of the data to upload when the world changes.

We have two buffers containing data for drawing a single tilted cube (one for triangles and one for lines). We use a two draw calls for each of the spinning cubes. Since cubes can be picked up in an arbitrary order, this was the most straight forward approach we could think of.

We create a single buffer containing the "block placement wireframe".

4 Draw Calls

We have two draw call for each chunk (one for blocks and one for block wireframes). Optimization: If we could somehow determine whether a chunk is actually visible we could omit some draw calls. A chunk has three vertex attributes, namely positions, normal vectors and texture coordinates.

We have two draw call for each spinning cube (one for the block and one for the block wireframe). It has the same vertex attributes as a chunk.

We have a single draw call for the "block placement wireframe".

In order to perform picking using offscreen rendering, we render all of our chunks (again) using a special shader program. We only use the position and the normal attribute.

5 Shaders

We have the following shader programs:

- Our "cube program" is used to render chunks and spinning cubes.
- Our "cube wireframe program" is used to render chunk block wireframes, spinning cube wireframes and the "block placement wireframe".
- Our "whale program" is used to render the sky whales. The fragment shader simply chooses a color based on the whale's normal vectors.
- Our "picking program" creates unique colors for each cube. The color is composed of the position of the fragment in world space, as well as which side of the cube the fragment is from.

6 Parameters

All of our shaders gets three different matrices.

A model matrix for transforming from "model space" to "world space". A view matrix for transforming from "world space" to "view space". A projection matrix for transforming from "view space" to "clip space".

Our cube program gets loads of parameters for the Phong lighting model. We give it a position for our torch and direction vectors for the sun and the moon as well as colors for each of the lights.

7 Final remarks

Instead of putting every single cube in a chunk into the chunks buffer, we only upload that are completely covered by other blocks. We can do a lot better by omitting more of the occluded faces. We could combine faces next to each other into the same "face".

We have tested our game using Firefox, Chrome and Safari.