1 Report

Rasmus Mosbech Jensen - 20105109

Lars Christensen - 20115025

Sune Døssing - 20115515

1.1 An overview of your code (files, structures and responsibilities)

In our code , most of ours global variable is been initialized in the top. Later in the initialization part we handle the content for the buffer arrays, then we initialize the buffers and the eventlisteners. Afterward we start the rendering. All this we do with the help of some helper functions.

Our world is made of 10x10x10 blocks, which is filled with dirt-blocks, stone-blocks, water-blocks, bedrock-blocks(lowest layer of the world) and some air blocks, which are not rendered. The starting view is placed on the top of the blocks, and moves along with the player, as the user rotate or moves.

1.2 Which buffer do you have (and why)

We are some buffers. Those are: vBuffer - This is for vertice positions. cBuffer - This is for the colors (one for each vertex). iBuffer - It is a buffer of indices for drawing the ordinary blocks and wireframe indices afterwards. sBuffer - It is a buffer of indices for drawing the small rotating blocks. swBuffer - It is a buffer of indices for drawing wireframes around the small rotating blocks. We use the index buffers for the drawing with gl.drawElements method. This allows us to avoid having to change the attribute-buffers.

The sun and moon have a vPosition-buffer and a vColor-buffer, but also stub-buffers for the remaining attributes, so we don't require a new shader program.

Finally, we have one cPosition buffer, named centBuffer, that gives each vertex the position of the center of its parent-block, which allows small-blocks-vertices to be scaled and rotated around their block center. This is done, so we can avoid having to do individual draw calls for every small block.

centColBuffer is used for picking, and will let each block be colored based on where its centerposition is. This gives each block a unique color that can we can look up.

1.3 Which drawcalls do you have (and why?)

We use one drawElements call for the blocks, one for the block-wireframes, one for the selection-wireframe, one for the sun and moon, one for small blocks and one for small-block-wireframes.

These drawcalls could be combined into one, but are seperated logically to make it easier to keep an overview of.

1.4 Which shaders do you have (and why?)

We have one vertex shader and one fragment shader.

Our vertex shader applies modelview and projection matrices, and also does scaling and rotations for small-blocks. It also calculates vectors for lighting effects. Our fragment shader deals with Phong lighting, it also has a branch which allows textures and lighting to be turned off, for picking.

1.5 Which parameters are transferred between javascript, vertex and fragment shaders (and why?)

Our attributes consist of positions, colors, center-positions for parent blocks, texture coordinates, and a normal. Although vColor was initially meant to consist of normals, we ended up using this for picking, and so had to make another attribute to get normals for every vertex.

We have a projection matrix, a modelview matrix and a small-block rotation matrix, which are sent as uniform variables. The modelview matrix defines the direction and origin of the view, while the projection matrix defines what kind of view it is.

The small-block rotation matrix is used to achieve the rotating effect for the small-blocks. The matrix is prepared on the CPU side, and is changed to a new rotation each rendering. It is set to a standard mat4() while drawing ordinary blocks, to avoid rotating or scaling those.

Lighting effects have their own uniforms. Ambient is set as a global effect that is independent of light sources. Each light-source has their own diffuse, specular and position uniforms. Lighting related positional data (L, E, N) is moved from vertex shader to fragment shader, to avoid having to do them for every fragment.

1.6 Is you solution the optimal one, or are there alternatives?

Currently, because we only have one vertex shader, we move every vertex back to 0,0,0, and apply the small block rotating matrix to them. If we had used two shaders, we could avoid having to move vertexes to 0,0,0, if they didn't need to be rotated as small-blocks do.

Our solution uses 36 vertices for every block, even if those blocks share corners with other blocks. The idea of using drawElements is that each vertex only appears once, and can be referred to multiple times using indices. However, because the color buffer is matched to the vertex buffer, we ended up having multiple vertices for the same point, to have different colored faces and black for wireframe. It would have been more efficient if we could somehow have used one color per index instead of vertex.

Our world has a constant-max size of 10x10x10, due to being built based on clip coordinates, if we had instead given each block a constant-size, and let them extend beyond the -1,-1,-1 to 1,1,1 range, we could have had a much greater amount of blocks. Because of this limitation, it is not possible for the user to add blocks outside of the predetermined area.

Our implementation of the Blinn-Phong lighting model requires several uniforms and varying variables for every light source, and quickly bloats up. The small rotating blocks do not receive the correct lighting effects.