1 Report

Rasmus Mosbech Jensen - 20105109

Lars Christensen - 20115025

Sune Døssing - 20115515

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

Our code is split up into parameters in the top, initialization, eventlisteners, rendering and helper functions.

Our world is filled with dirt-blocks from -1,-1,-1 to 1,1,1. The starting view is placed ontop of the blocks, and moves along with the player. We first render blocks, then block wireframes, then small-blocks, and then small-block-wireframes. Finally, we render the sun and the moon.

1.2 Which buffer do you have (and why)

vBuffer for vertice positions, cBuffer for colors (one for each vertex). iBuffer is a buffer of indices for drawing ordinary blocks and wireframe indices afterwards. sBuffer is a buffer of indices for drawing small rotating blocks. swBuffer is a buffer of indices for drawing wireframes around small rotating blocks. Index buffers are used for drawing with drawElements, and allow us to avoid having to change attribute-buffers.

The sun and moon have a vPosition-buffer and a vColor-buffer, but also stub-buffers for the remaining attributes, so it doesnt 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 to avoid having to do individual draw calls for every small block.

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

1.3 Which drawcalls do you have (and why?)

We use one drawElements call for blocks, for block-wireframes, one for 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 matrixes, 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 projectil matrix defines what kind of view it is.

The small-block rotation matrix is used to achieve the rotating effect for 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 independant 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 have only one vertex shader, we move every vertex back to 0,0,0, and apply the small block rotating matrix to them. If we used two shaders, we could avoid having to move vertexes to 0,0,0, if they dont need to 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 refered 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 indice 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.

Our implementation of the blin-phong lighting model requires several uniforms and varying variables for every lightsource, and quickly bloats up.