# 1 Report

Rasmus Mosbech Jensen - 20105109

Sune Døssing - 20115515

Lars Christensen - 20115025

Our blocks are represented as a block object, which holds a block type and 4 vertices. The initial amount of blocks is coded as an X and Y amount in the javascript code, and the layout of these is made in proportion. Blocks are drawn as two triangles using triangle strip, which creates the triangles of v1,v2,v3 and v2,v3,v4. The block-outline and the stickman are hardcoded at the end of the vBuffer and cBuffer, after the blocks.

In our rendering function, we first iterates through each of our blocks, calling drawArrays individually for each of them. The mouse-over box is then highlighted, and then stickman.

Each blocktype has a predefined color which is used when we render each block. Color gradient is calculated in the fragment shader, where the darkness of the color, of a given pixel, is calculated depended on the distance from the pixel to the center of its parent block. The distance is calculated in the vertex shader and passed on to the fragment shader.

Our stickman uses a uniform vec4 (actually only vec2) which is added to each of the stickmans vertex positions to allow movement. This allows us to only put vertex positions into the buffer once and update it only with an offset. We also use a wavedistance variable which is incremented by the javascript code and determines how far the wave effect has travelled.

We use a uniform variable to determine which blocks are affected by the gradient effect, and by offset. And we use uniform variables to get corners, that we use to calculate a blocks middle position, and center of a mouse click within the vertex shader.

Necessary data from the vertex shader, such as wave distance (for wave effect) and the blocks middle position (for gradient), are shared with the fragment shader via varying variables.

The buffer has enough space for every blocks four vertices, the four vertices of the mouse-over box, and 8 vertices for the stickman figure. The two buffers is initially being filled with the blocks using the handleBuffer() function, which iterates through our array of blocks and adds the corners of each block, one block at a time.

In the render function, if a mouseclick has happened, the selected block will be recolored (by changing the color buffer) to match the new block type, that has been chosen in the menu on the GUI, and it has its object values updated to reflect this.

The four corners of the highlighted box is putted in the cBuffer every time the render function is called.

The stickman is placed into the buffer once and never again. Its vertices are never updated in the buffer, but are instead moved by adding an offset to them in the vertex shader.

We have not implemented the most optimal approach, since there is a few things that can be done to our project, so it will run more efficient (less calls to the buffer ect.). Air blocks are treated the same way as all other blocks, but could actually have been not-rendered to avoid having to do vertex and fragment shading for them, and just leave the blank canvas instead.

Our highlighting box is submitted to the buffer every time the render function is called. This is much slower than submitting it once initially and moving it around using a uniform vector offset.

The middle position for each block is calculated in every vertex, but only changes with every call to drawArrays.

Our solution uses only two buffers (one for positions and one for colors), one vertex shader, and one fragment shader. Many conditionals could have been averted if we used more than two buffers, which could be an additional one for the stickman and for the wireframe outlining the hovered box. We would not have to check in the vertex shader if we are dealing with stickman or a block, if we had a dedicated vertex shader for the stickman. This also means many uniform variables are handled even though they are not needed for every vertex.

The vertex shader checks if it is handling the stickman, and adds the offset to its position. It calculates the middle point of the vertexs parent block, and gives this to the fragment shader. It also handles wave effect, by getting the middle point between the mouseclick and the current vertex, and changing its position accordingly.

The fragment shader calculates the distance of each fragment to the center of parent block (given as centerpos by the vertex shader), and changes the color accordingly, to create gradient. It also changes color depending on how strong the wave effect is, which is also given by the vertex shader.

Our solution runs on Firefox (and Nightly), Internet Explorer (and Edge), and Chrome. Chrome runs very slowly, while the others do not. Edge generates an "unspecified error", which doesnt seem to cause any problems.