# Globe Crime Visualization

Roy Portas - s4356084

May 14, 2017

## 1 Introduction

The aim of this project is to create a web based visualization of global crime data. This will be done through a 3D model of a globe with crime data superimposed on top of it.

The visualization aims t be a easy to use tool for the general public to view crime trends around the world over time.

## 2 Methods

### 2.1 Acquiring Data and Domain Research

Data was acquired from the website [website].

Before building the web app, research was done on existing solutions, such as Google Earth and ...

### 2.2 Designing the Object Hierarchy

The object hierarchy allows us to define groups of objects to manage the transformations and rotations more easily. It usually takes the form of a tree structure with leaf nodes being the smallest parts of the model.

The complete model will consist of the following elements:

- The world
- The sun (light source)
- The moon, which rotates around the world
- Crime heat maps, which are displaying on the world.

Thus the model can be representing in the following hierarchy:

INSERT TREE DIAGRAM

### 2.3 Version 1: Pure WebGL

The initial prototype was created in pure WebGL, which is graphics library for web browsers with an API similar to OpenGL ES2.0. The first step to creating the WebGL program is initialize WebGL

and setup the shaders and buffers. To start with a simple cube was created, the shape of the cube was defined as vertices.

```
const cubeVertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexBuffer);
const vertices = [
    // Front face
    -1.0, -1.0,  1.0,
    1.0, -1.0,  1.0,
    1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,

    // Back face
    -1.0, -1.0, -1.0,
    -1.0,  1.0, -1.0,
    1.0,  1.0, -1.0,
    1.0, -1.0, -1.0,

    .... // Do for all faces
];

gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

// We are using 3 dimensions
cubeVertexBuffer.itemSize = 3;
// We need 24 vertices to represent a cube
cubeVertexBuffer.numItems = 24;
```

This buffer contains all the points required to create a cube. WebGL then takes this JavaScript and converts it into GLSL, which is ran on the GPU.

After defining the position vertices, the colour buffer was created, for this example the colours was kept simple, a solid green colour for all the cube.

Once these buffers are setup, the next thing to do is to setup the shaders. For this prototype two shaders where used, a fragment shader and a vertex shader.

The fragment shader in this case just sets up the GPU to use the correct data types for the rest of the code. The vertex shader generates coordinates in the clipspace, which is used by the GPU to render the objects correctly.

Once these shaders were created two matrices were set up, one for the model view matrix, the other for the projection matrix. The model view matrix transforms coordinates into view coordinates, which is used by the GPU to display the objects. The projection matrix is used to used to manage the perspective of the scene.

### 2.4 Version 2: ThreeJS

As the project became more complex, it became hard to manage all the WebGL code.

## 3 Results

## 4 Discussion

## 5 Conclusion