ITC363

Assignment Three

Sean Matkovich

Student ID: 11187033

Computer Graphic

Task 1: A 3D Function-Viewing Program

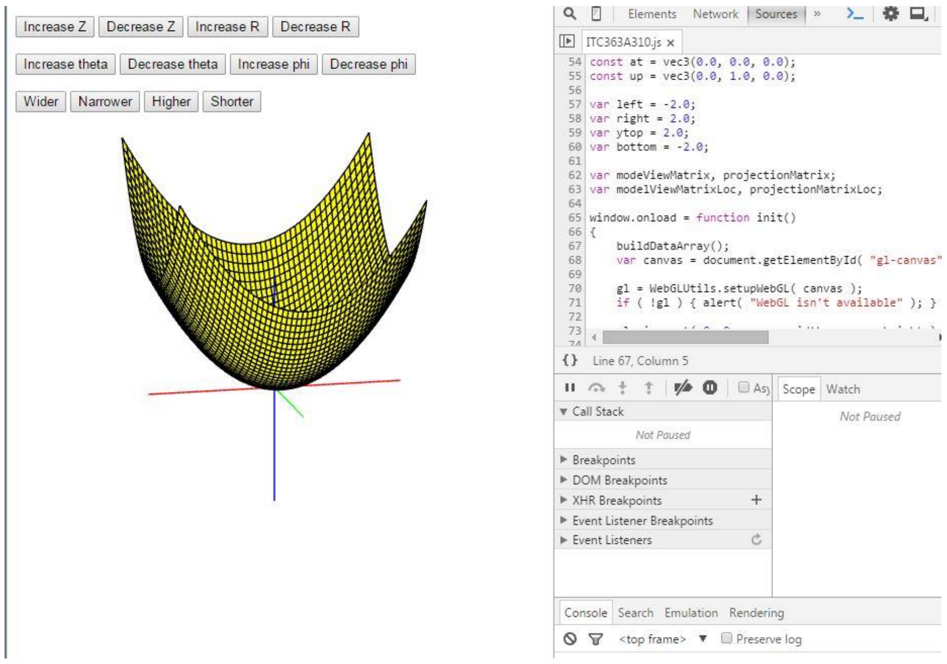
For this task you will create a *3D Function-Viewing Program* using WebGL. The program will allow the user to view a 3D plot of a particular function of two variables in 3D, displaying the surface representing the function as a mesh and allowing the viewer to change the viewing position to investigate different aspects of the function. The particular function to be used is

$$f(x, y) = y^2 - x^2$$

Within the region bounded by the cube of side-length 2 whose centre is the origin, i.e. in $-1 < x < 1$, $-1 < y < 1$, $-1 < z < 1$ with the function value being represented along the *z*-axis. The function is specified here, in the usual way of mathematicians and physicists, in terms of an *x*-axis and a *y*-axis in the horizontal plane and a vertical *z*-axis.

Solution Description

The solution to the task comprises of two files `ITC363A310.html` and `ITC363A311.js`. `ITC363A310.html` takes significant functionality from Angel (2015) hata.html through the use of the same buttons interface. The only exception being the addition of a slider, which allows the user to alter the number of displayed vertices.



```
const at = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);

var left = -2.0;
var right = 2.0;
var ytop = 2.0;
var bottom = -2.0;

var modeViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;

window.onload = function init()
{
    buildDataArray();
    var canvas = document.getElementById( "gl-canvas"

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }
```

Figure 1 ITC363A310.html

The actual canvas displays an orthographical projection of the 3D plot. The `BLACK` mesh structure is constructed from vertices displayed as `GL.LINE_LOOP`. The `GL.LINE_LOOP` is cast over a `YELLOW` surface created by a `GL.TRIANGLE_FAN`. A further three coloured lines are displayed which represent relative axis of the graph (Figure 2 shows `RED` X-axis, `GREEN` Y-axis, and `BLUE` Z-axis).

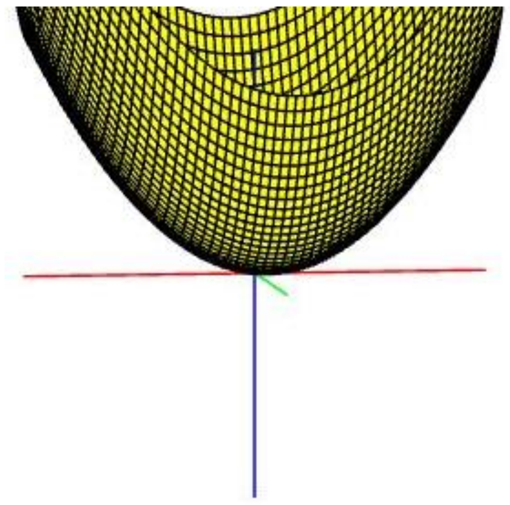


Figure 2 X, Y and Z axis display

In order to display the mesh clearly hidden surface removal and `Polygon offsetting` has been implemented through the use of `GL.DEPTH_TEST` and `GL.POLYGON_OFFSET_FILL`. As previously mention the controls to move the displayed function are based on those provide by Angel (2015) in the hata.js and hata.html. The 3D plot is rotated and scaled by modifications to the state variables which control the `eye` position and `projectionMatrix`. The `eye` vertex directly alters the `modelViewMatrix`; whilst the `projectionMatrix` is derived from the Angel (2015) `MV.js ortho` function; ultimately the transformation of the view is implemented in the vertex-shader by the matrix multiplication

```
projectionMatrix * modelViewMatrix *vPosition
```

Task 2: 3D Park Scene

For this task you will use WebGL to create a program displaying a 3D Park Scene. The simulated 3D scene of a park will contain pathways, grass regions, trees and huts, and interactivity will allow the user to navigate freely through the scene. A perspective projection will be used.

Specifications

The components of the scene will be:

- An extended flat rectangular area, coloured green, will provide the basic grass region on which objects are placed.
- Two straight pathways of constant width cross the region and intersect.
- A number of trees (at least 10) of different heights and widths are distributed through the scene. All trees are conifers, represented as a conical shape on a cylindrical trunk. Different shades of green distinguish different types, providing variety. The trunks should be a brownish grey.
- Two huts, perhaps used as shelters, appear in the scene. A basic hut is a cube surmounted by an overhanging square pyramid. Scaling along one direction can produce a more general hut. Both huts should be of the more general kind.

User interaction

The user should be able to manipulate aspects of the view as follows:

- The look-at point should be able to be changed by the user. An initial default will be set.
- The view-reference point (VRP) should be able to be moved through the scene. It is not constrained by pathways. In normal use the user should not have to alter the height of the VRP, but a height changing control could be included.
- It is important to include functionality for moving the VRP forwards and backwards and for swinging the look-at point right and left. It would be a mistake to allow the VRP to overrun the look-at point.
- In normal use the look-at point should be at the same height as the VRP, but a height adjustment for looking higher and lower could be included.
- The view-up vector should normally be maintained in the vertical direction normal to the extended flat rectangular area.

Implementation requirements

The implementation of your program should satisfy the following requirements:

- Instance transformations should be used. The various conifers will be instance transformations of a model conifer, whose vertices are held in GPU memory. The huts will be instance transformations of a model hut, whose vertices are held in GPU memory.
- Uniform qualified variables will be used to handle data specific to each instance.
- The scene will be implemented using a perspective projection.

Modelling

Appropriate geometric primitives for the model objects are:

- For the conifers, the conical top should be represented as a triangle fan that wraps around onto itself. The cylindrical trunk should be represented as a triangle strip that wraps around onto itself.
- For the huts, the base could be represented as a cube, described at length in the textbook, but since the top and bottom are not required, it would be simpler as a triangle strip that wraps around onto itself. The pyramidal top should be a triangle fan that wraps around onto itself.

Solution Description

The solution to the task comprises of four files `ITC363A320.html`, `ITC363A320.js`, `Hut.js and Tree.js.` As in previous task implementation the ITC363A320.js is the controlling component, maintaining state and handling all of the logic. The two file `Hut.js` and `Tree.js` are class object implementations for the respective components which are added to the view. The idea of separating these components from the main ITC363A320.js file was predominately from the view of file readability.

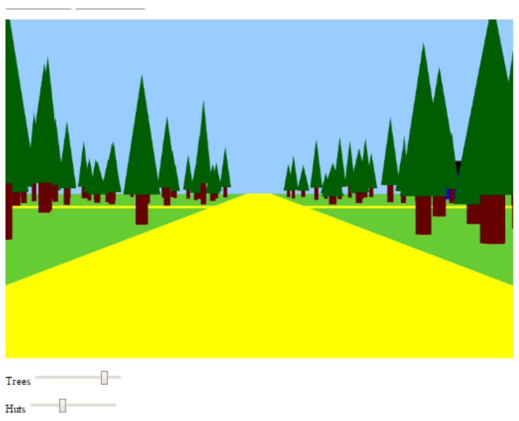


Figure 3 ITC363A320.html The Park scene.

The actual user interaction navigation and view controls has been based on Lodge (2015) implementation of a Stonehenge scene. The vertex buffer is loaded with the ground, path, tree and hut vertices. Each instantiation of tree and hut objects access the buffered vertices and applies a transformation for placement and sizing at render. This method maintains an efficient use of buffer memory and uses the power of the GPU to handle the transforms at each render.

On each keyboard interaction the state variables are altered. These variables change the `eye` and `at` positions which consequently alter the `modelView` matrix that is passed into the vertex-shader at render. The two slider (Figure 4) components allow the user to dynamically change the number of object rendered within the scene, by sliding left or right objects are recalculated and placed randomly within the area.

The `ITC363A320.js` uses logic to randomly place each `tree` and `hut` object throughout a constant area of the scene. As a result each time the `html` page is refreshed or the number of objects is altered the scene changes. The selection of each object locations is centered around a nested function `inCircle(point, origin, radius)`. This function takes two points (`point` and `origin`) and a distance (`radius`) from which it calculates if the `point` encroaches on territory of the object found at the location `origin`. The territory is defined as the area of the circle inscribed by the `radius` dimension.

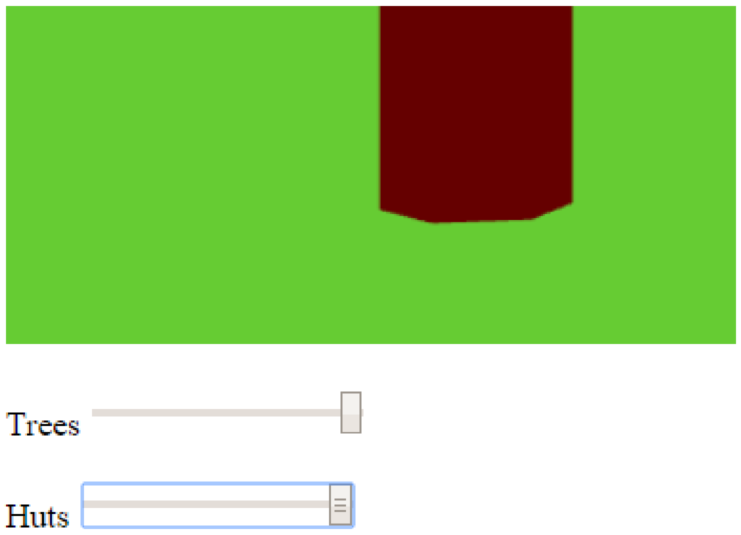
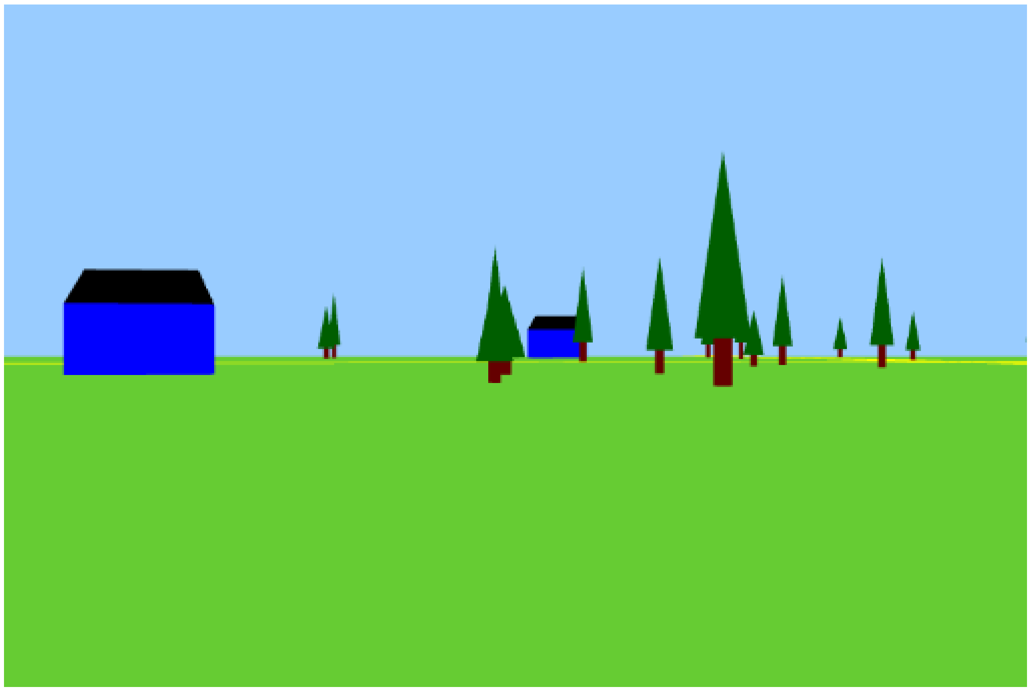
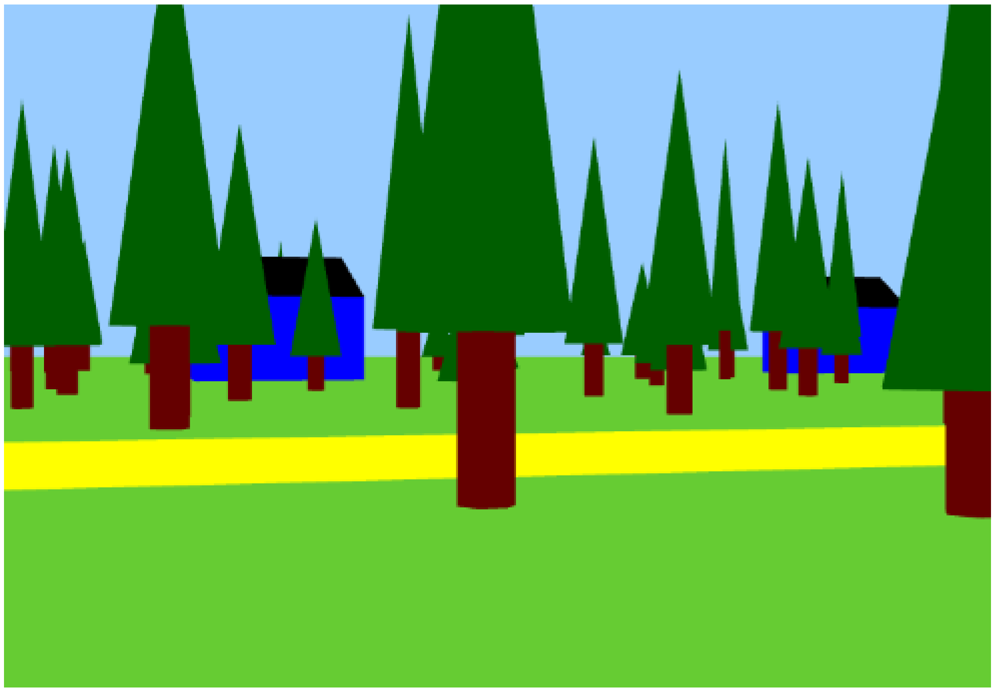
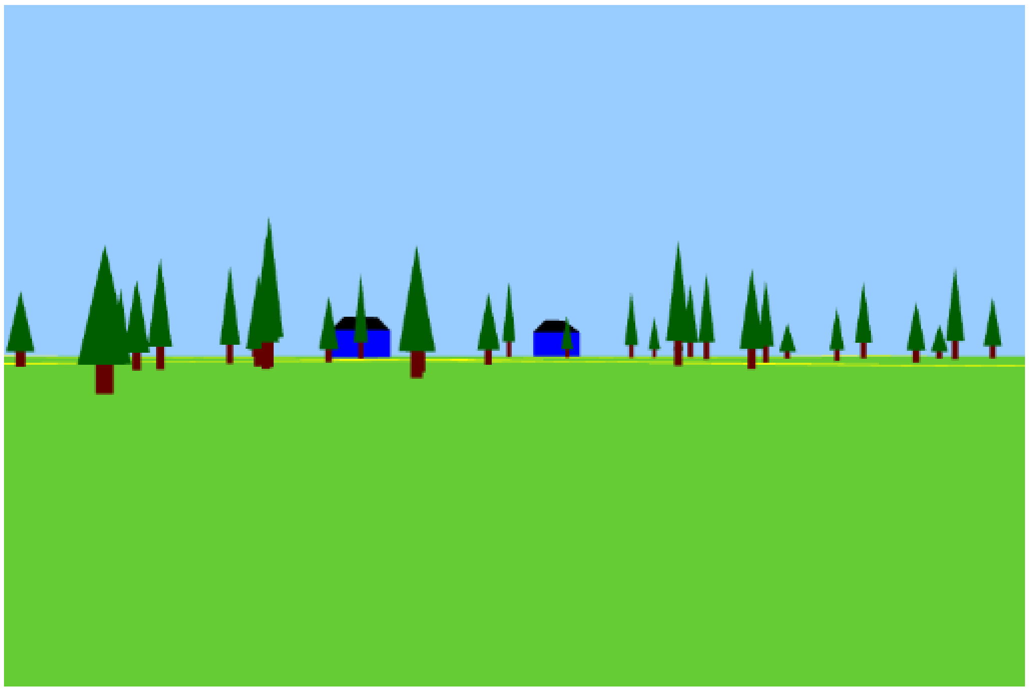


Figure 4 The HTML Sliders for altering the number of objects within the scene







References

Angel, E., Shreiner, D., Bhattacharjee, A., & Mukherjee, S. (2015). Interactive computer graphics. Boston [etc.]: Pearson.

Lodge, K. (2015) Stonehenge.js