Task

Your task is to complete the implementation of a 3D world. In this world you control a camera moving around a landscape which includes trees, hills and roads.

Files

The starter code is packaged with v0.18 of UNSWgraph (available here or via github). In the unsw.graphics.world package you will find a some skeleton classes. They implement some basic things you need, but are incomplete. The files provided are:

- **World.java** - this is the main entry point to your application
- **Terrain.java** - this class represents variable height terrain.
- **Tree.java** - this class represents a tree
- **Road.java** - this class represents a road as a bezier curve
- **LevelIO.java** - this class reads and writes game levels to and from JSON files.

Under res/worlds you will also find some sample level files. These specify various properties of a world. See test1.json to get a basic idea of the format.

You are free to change any of these files and any other files in UNSWgraph, as you see fit. We will not be testing individual functions. However, you should make sure that the established Level IO format works for your code, because we will be testing your level with standard level files.

World

This is the main class for your game. The main() method in this class will be used to test your game. It expects a single string specifying the name of the level file. If you want to specify any other parameters they should be part of the JSON file.

Initial milestone (end of week 10)

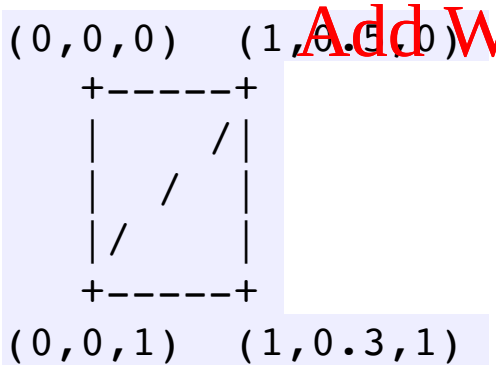
Terrain

The terrain is represented as a grid. The width and depth of the grid are specified in the level file. Each point in the grid has a specified altitude (height). **Your first task** is to draw the terrain as a mesh of triangles with vertices at each of the grid points with the corresponding altitude.
You can treat X,Z and altitude as a 3D coordinate. They should all have the same scale.
A 2x2 terrain with altitudes:
Note: the bold labels (x0,x1,z0,z1) are just to explain what the values mean and will not actually be part of the data

	x0	**x1**
z0	**0**	**0.5**
z1	**0**	**0.3**

A 2x2 terrain represents 4 vertices. The altitudes correspond to the Y values for the x,z co-ordinates.

Will create a mesh withe the following co-ordinates

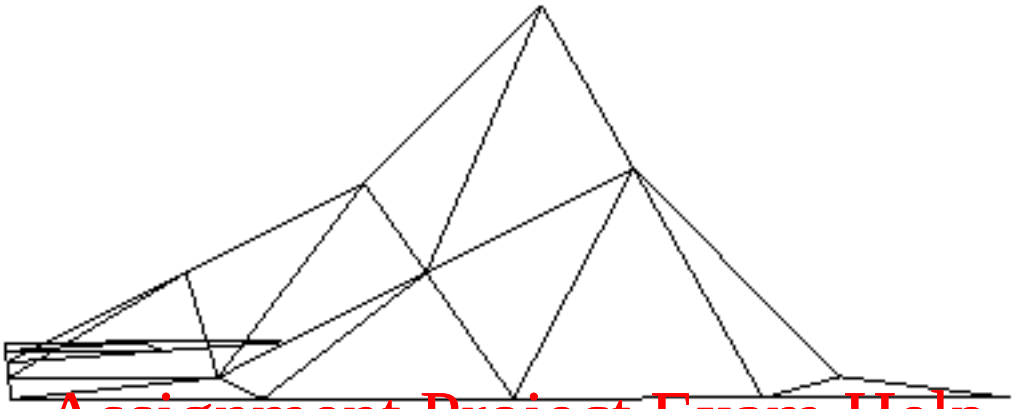


A 5x5 terrain with altitudes:

0	0	0	0	0
0	0	0.5	1	0
0	0.5	1	2	0
0	0	0.5	1	0
0	0	0	0	0

Will create a mesh that looks like this (this may look different depending on the angle/position you view it from and exactly how you set up your camera). This is taken from basically straight ahead

at around (0,0.5,9) in world co-ordinates and with a perspective camera:
Note: This screenshot was taken with back face culling on



Note: for the assignment you will texture your terrain and make it look pretty. We are just showing the lines here so you can clearly see the geometry. It is up to you whether you implement your terrain using face nomals or vertex normals.
You should consider efficiency when constructing and drawing the terrain.

Trees

The levels include trees at different points on the terrain. **Your second task** is to draw the trees at the specified locations.
In the screenshots, the trees are drawn with two separate textured meshes: a cylinder for a trunk and a sphere for the leaves. For the base part of the assignment, you only need to have a tree that is a single textured mesh. We have provided a PLY file of a tree that you can use. However, if you want you can make your 'trees' more exotic: lampposts, candy-canes, chimneys, or whatever your

imagination dictates. The point is that they are placeable 3d models on the terrain.
Note that the level descriptions only specify the (x,z) location of the tree. You will need to use the terrain data to calculate the altitude of the tree and draw it appropriately. Trees are not guaranteed to be positioned at grid points, so you will need to interpolate altitude values if a tree is in the middle of a triangle.

Camera

You should implement a 3D camera which **moves** around the scene using the arrow keys:

- The up arrow moves forward
- The down arrow moves backward
- The left arrow turns left
- The right arrow turns right.

The camera should move up and down **following the terrain**. So if you move it forward up a hill, the camera should move up the hill.

Sunlight

Your world should be rendered using Phong shading. The level files include a "sunlight" field which is a 3D vector specifying a directional light to be included in the scene. The vector represents the direction **to** the source of the light.
You will need to create a new vertex and/or fragment shader to support defining a directional light instead of a point light. You can base it off one of the preexisting shaders if you wish. Note that the shader will also need to support texturing.
You will need to decide for yourself what are the appropriate other properties for the light and the appropriate material properties for the surfaces that will reflect it.

Final milestone (end of week 12)

Avatar

Add an avatar and make the camera follow behind the avatar in a 3rd person view. You should being able to switch from 1st person (with no avatar) to 3rd person (with the avatar) by pressing a key of your choice. For the base part the avatar does not need to be a complex model (a bunny is ok), but it does need to be textured.

Road

The level include roads. Each road is described as a 2D Bezier curve. I have provided a function for you which calculates the (x,z) location of points along the road. You will need to to use this function to extrude a road which follows this curve, with the width specified in the constructor.
You can assume, for the base portion of the assignment, the roads will only run over **flat terrain**, so you will not have to handle going up or down hills. You can assume the starting point of the road is at the altitude for the entire road. The should have an appropriate texture applied to it.

Night time and torch light

Your world should support a night mode triggered by a key press. In the night mode, the sunlight should dim (but not go away entirely) and a torch attached to the camera/avatar should switch on. This torch should be an attenuated spotlight. You will need to modify your shader to support this second light. It should be possible to pass properties of the spotlight (cutoff, attenuation, etc.) from the Java program to the shader.

You may have 2 separate shaders for the day and night mode if you wish, but keep in mind that when you load the new shader, you will need to pass all the parameters to it again.

Extensions

The base elements described above are worth 14 of the 20 marks. For the remaining 6 marks you can choose among the following extensions:

- Build a complex model or a model with walking animation or something beautiful or interesting for your avatar! (2..4 marks)
- Make the sun move and change colour according to the time of day (2 marks)
- Add distance attenuation to the torch light (2 marks)

- Add rain using particle effects (4 marks) For the full marks this would need to include alpha blended billboarded particles, creation and destruction ,some kind of evolution over time (position, size, colour, as is appropriate for your kind of particles).
- Add ponds with animated textures to your world (4 marks) Ponds need only lie on flat terrain like roads but should include animated textures showing ripples or waves.
- Add an L-system for fractal tree generation (4 marks) To get full marks for this you would need to implement a proper rewite system. You would not need to load the grammar for the L-system from JSON, but it should be possible to alter the grammar just by changing values in the code.
  You should also provide a way to increase/ decrease the number of iterations either interactively or from reading in the number of iterations from a json file. By default you should set it to the number of iterations that looks best/runs best. It does not matter if the tree does not look as good when iterations are increased/decreased. It is also ok if performance drops for high numbers of iterations. This is to be expected.
- Fix road extrusion so roads can go up and down hills (4 marks). To get full marks for this, your road must perfectly follow the terrain without intersecting or rising above it.
- Add lakes or ponds with a ripple effect. The ripple should be animated and realised by distorting the mesh in the vertex shader (4 marks)
- Add shadows to the trees and terrain (4 marks)
- Implement normal mapping on one of your models(4 marks)
- Add reflection mapping (using cube mapping) to one of your models (4 marks)
- Implement some form of NPR shading (6 marks)
- BSP trees for terrain rendering (6 marks)
- Add level-of-detail support for rendering distant objects with lower-resolution models (6 marks)
- Implement Cook-Torrance shading or some other form of physically based rendering (6 marks)
- Implement grenades or bombs that cause the terrain to deform when they explode.
- Add Portal style portals.
  - Portals you can walk through (4 marks)

    - Portals you can walk and see through (8 marks)
- Implement the terrain as a Bezier or NURBS surface (8 marks)

If you have other ideas for extensions please ask on the forum. If there are any I like, I will add them to the list. I'm looking for extensions which test your use of different rendering techniques rather than just adding more stuff to the world.

Note: The marks above increase roughly logarithmically with the amount of work required. So a task worth 6 marks is about 16 times harder than a task worth 2 marks. The higher mark extensions will also require you to do your own research as they may require ideas and techniques not covered in this course.

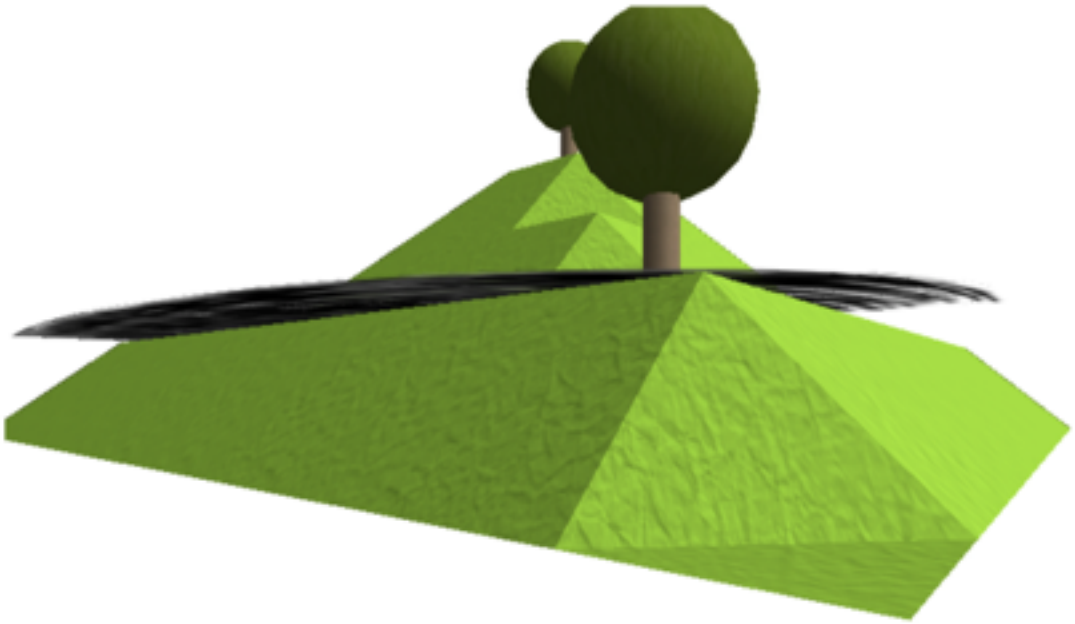
Assignment 2: Sample Example worlds

These are the some screenshots of the example worlds provided. Note that screenshots were taken from whatever angle best captured the scene. Your output doesn't have to look exactly like this, but the following elements should match:

1. the shape of the terrain
2. the direction of the lighting
3. the shape of the road

Some of them include the line outlines of the triangles so you can see the geometry of the terrain. You should not draw such lines in your implementation

Test 1

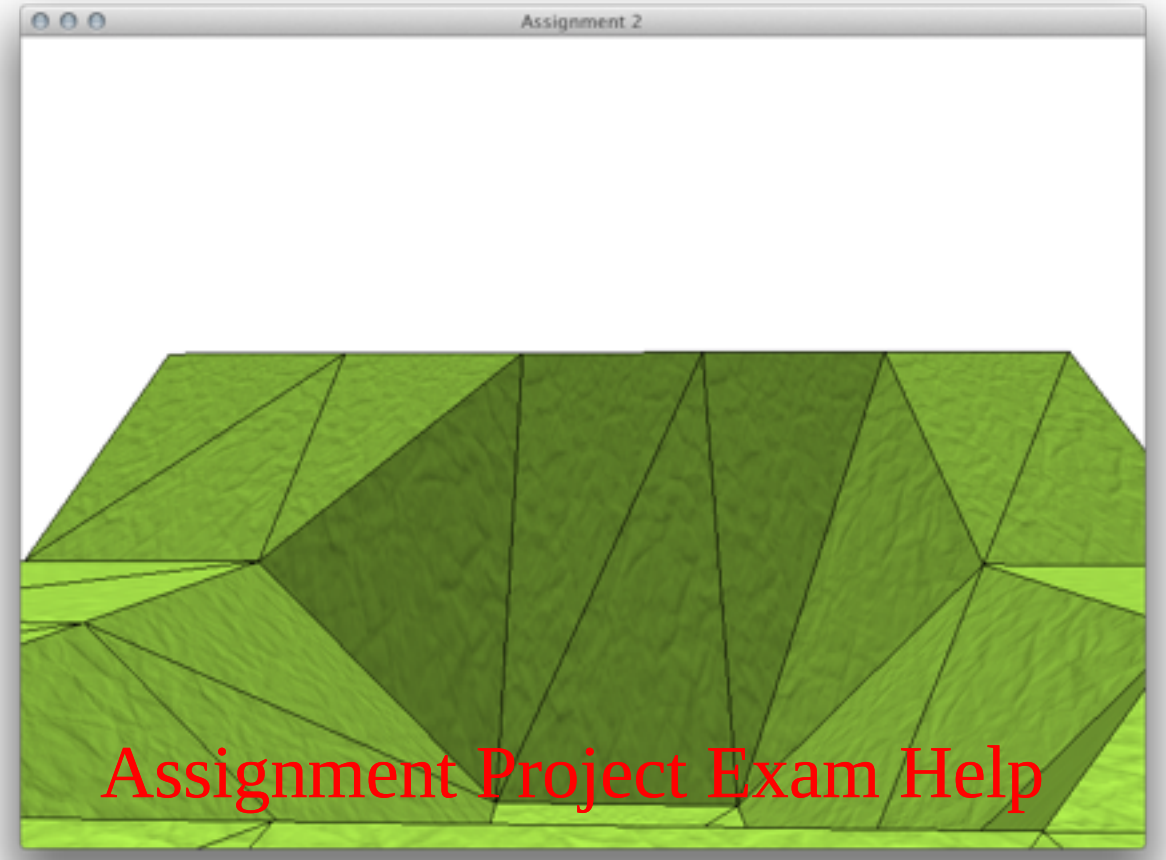


The road hangs over the edge of the terrain.

Test 2

Various slopes lit from above

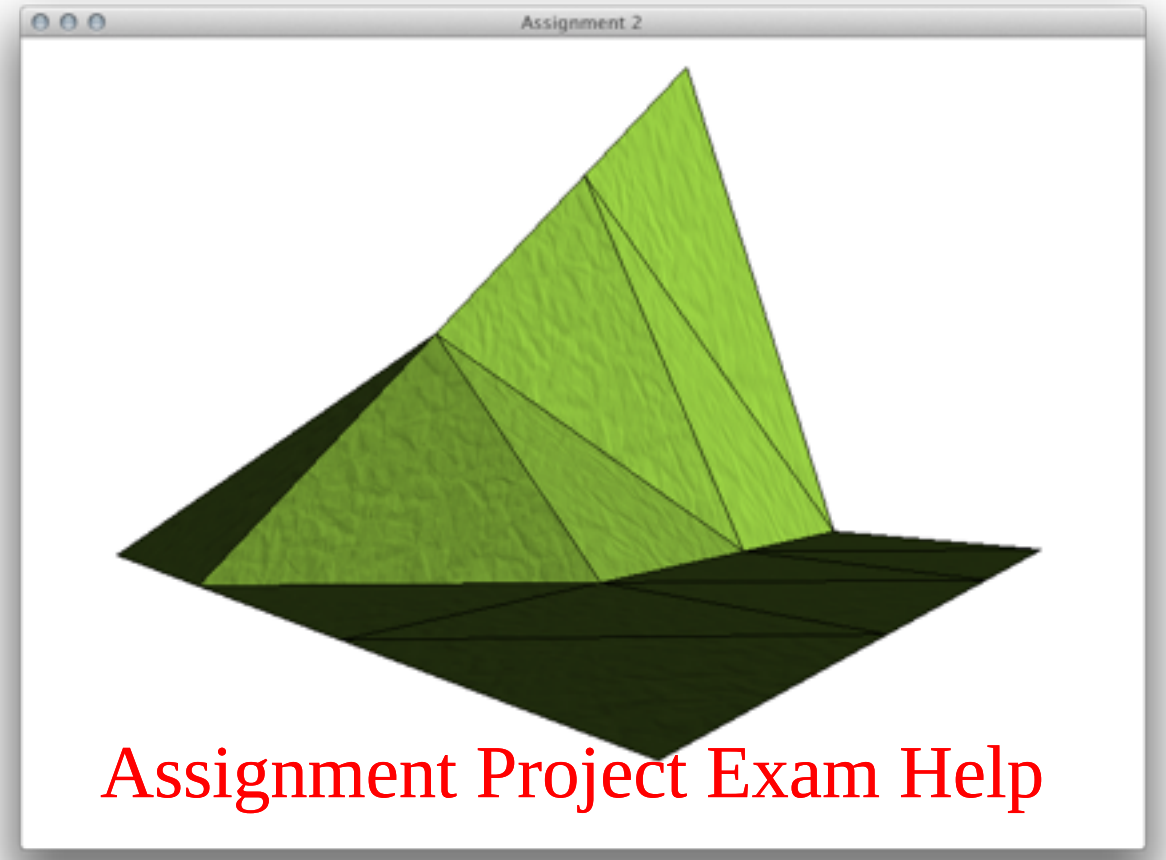
```
{
    "width" : 6,
    "depth" : 6,

    "sunlight" : [ 0, 1, 0 ],

    "altitude" : [
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 0, 0, 1, 1,
        1, 1, 0, 0, 1, 1,
        2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2,
    ]
}
```



Test 3

A hill with sunlight coming from the negative x direction.


```
{
    "width" : 4,
    "depth" : 4,

    "sunlight" : [ -1, 0, 0 ],

    "altitude" : [
        0, 0, 0, 0,
        0, 0, 1, 0,
        0, 0, 2, 0,
        0, 0, 3, 0
    ]
}
```



Test 4

Trees at various heights testing interpolation

```
{
    "width" : 4,
    "depth" : 4,

    "sunlight" : [ -1, 1, 0 ],

    "altitude" : [
        0, 0, 3, 3,
        0, 0, 3, 3,
        0, 0, 0, 0,
        0, 0, 0, 0
    ],

    "trees" : [
        { "x" : 0.5, "z" : 0.5 },
        { "x" : 1.5, "z" : 0.5 },
```

```
        { "x" : 1.5, "z" : 1.5 },
        { "x" : 2.5, "z" : 0.5 },
        { "x" : 2.5, "z" : 1.5 },
        { "x" : 2.5, "z" : 2.5 },
    ]
}
```

Test 5

A single bezier curve of road

```
{
    "width" : 4,
    "depth" : 4,

    "sunlight" : [ -1, 1, 0 ],

    "altitude" : [
        0, 0, 0, 0,
        0, 0, 0, 0,
```

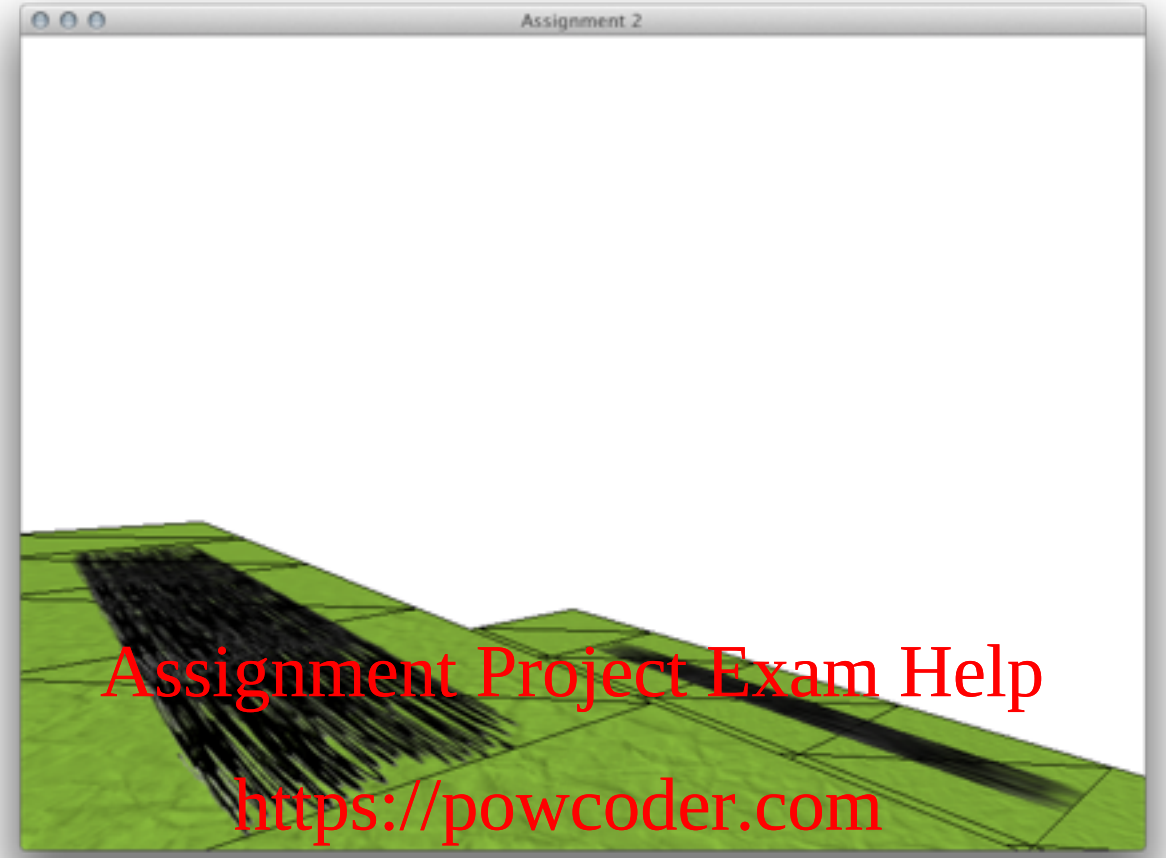
```
            0, 0, 0, 0,
            0, 0, 0, 0
        ],

        "roads" : [
            {
                "width" : 1,
                "spine" : [
                    0, 0.5,
                    3, 0.5,
                    3, 2.5,
                    0, 2.5
                ]
            }
        ]
}
```


Test 6

Two roads at different altitudes

```
{
    "width" : 6,
    "depth" : 6,

    "sunlight" : [ -1, 1, 0 ],

    "altitude" : [
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1,
        0, 0, 0, 0, 0, 0,
        0, 0, 0, 0,    0, 0,
        0, 0, 0, 0,    0, 0,
    ],

    "roads" : [
        {
            "width" : 0.5,
            "spine" : [
                1, 1.5,
                2, 1.5,
                3, 1.5,
                4, 1.5
            ]
        },

        {
            "width" : 0.3,
            "spine" : [
                1, 4.5,
                2, 4.5,
                3, 4.5,
                4, 4.5
            ]
        }

    ]
```

```
}
```

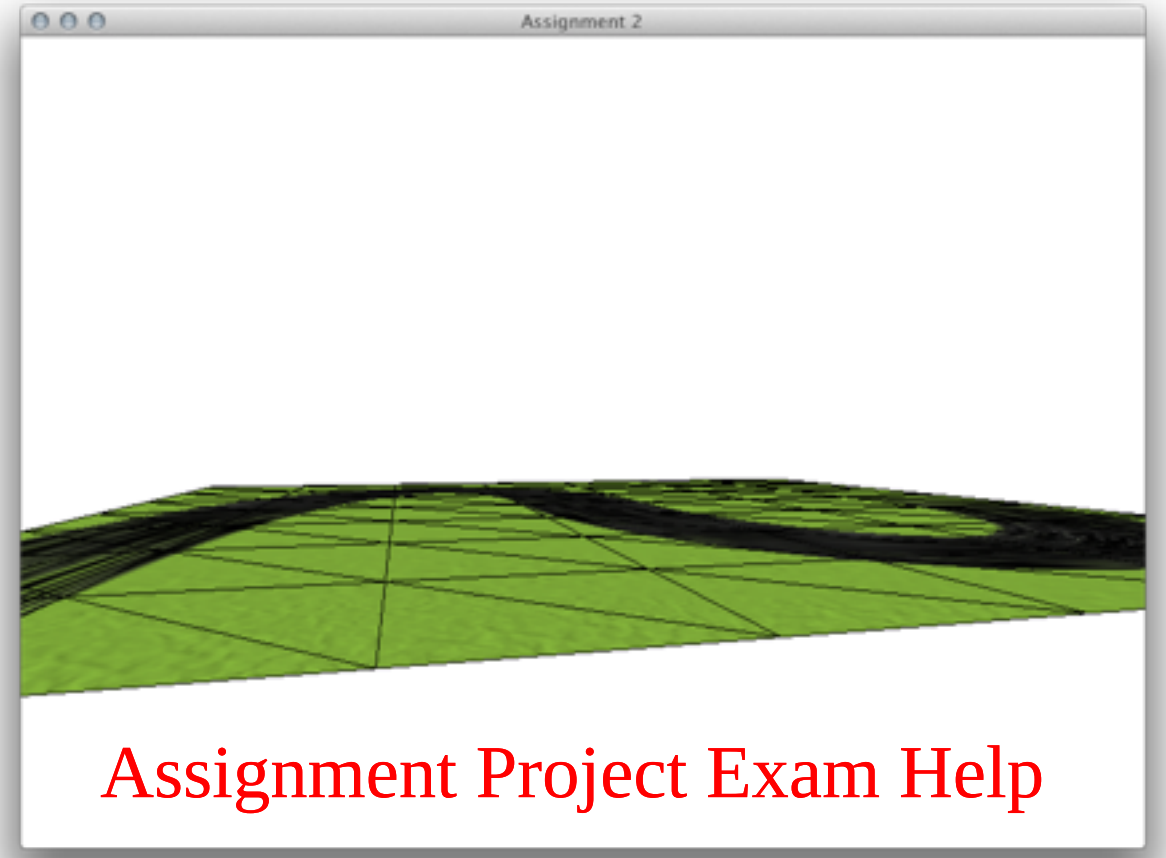


Test 7

A two piece bezier spline road

```
{
    "width" : 8,
    "depth" : 8,

    "sunlight" : [ -1, 1, 0 ],

    "altitude" : [
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0,
```

```
        0, 0, 0, 0, 0, 0, 0, 0,
    ],

    "roads" : [
        {
            "width" : 1,
            "spine" : [
                0, 0.5,
                7, 0.5,
                7, 3.5,
                4, 3.5,
                0, 3.5,
                0, 6.5,
                4, 6.5
            ]
        }
    ]
}
```



Tree Interpolation

It is ok to place the centre of the trunk at the interpolated y-value. You will not lose any marks for this, but it is obviously better to fix it