COSC422 Computer Graphics

Introduction to Terrain Programming

Aim:

This programming exercise introduces you to the fascinating field of terrain rendering through an example that generates a terrain model using tessellation shaders.

Note: Tessellation stages are available only in OpenGL 4.0 and later versions. Your program may generate run-time errors on systems with older versions of OpenGL.

Terrain.cpp:

(1) The program `Terrain.cpp` draws a set of quads arranged in a rectangular 10x10 grid containing 100 vertices (Fig. 1a). The quads represent the terrain's base plane. The program generates a display of the terrain base as in Fig. 1b.

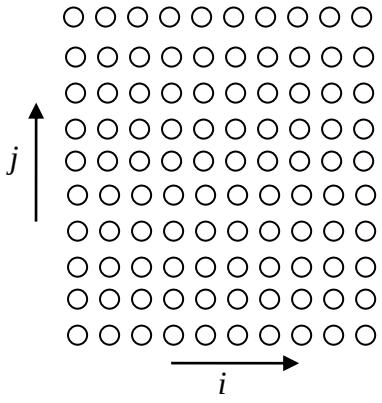

Fig. 1a

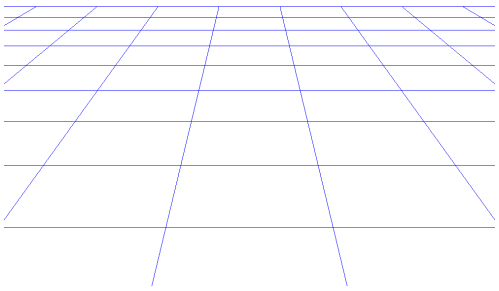

Fig. 1b

(2) In the display() function, modify the primitive type specified in the glDrawElements() function from **`GL_QUADS`** to **`GL_PATCHES`**.

Also, specify the number of vertices in each patch, inside the initialize() function:

```
glPatchParameteri(GL_PATCH_VERTICES, 4);
```

A patch is not a renderable primitive. Therefore, we should not convert the patch coordinates to clip space. Edit the vertex shader and remove the multiplication of coordinates by the model view projection matrix. The shader is now a simple pass-through shader.

A pass-thru tessellation control shader (`Terrain.cont`) that sets all tessellation levels to 2 is provided. Also provided is an evaluation shader (`Terrain.eval`)

The evaluation shader receives the (*u*, *v*) coordinates of the tessellated mesh in the built-in variable `gl_TessCoord`. It also receives the patch vertices in the

array `gl_in[i].gl_Position`, i = 0..3. Use these vertices and the (u, v) coordinates to generate a bi-linear mapping shown on Slide [1]-26. (Complete the expression for the variable `posn`). Note that the output is a vertex of a triangle on the patch, which is multiplied by `mvpMatrix` to transform it to the clip coordinate space.

In the initialize() function, add statements to load both the above shader codes, and attach them (using glAttachShader()) to the shader program.

The output of the program with tessellation levels set to 2, is given in Fig. 2a. Change the tessellation levels to 6; the output should now change as shown in Fig. 2b.

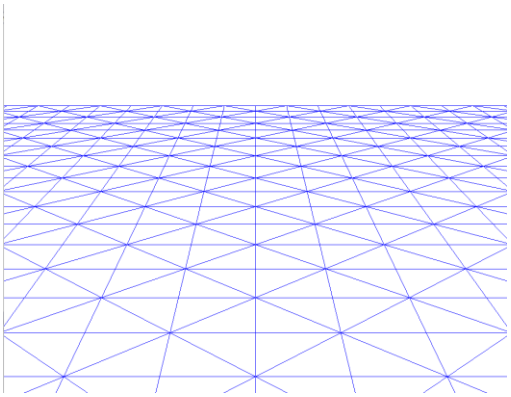

Fig. 2a

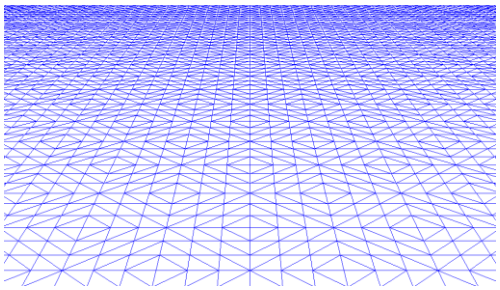

Fig. 2b

(3). The program includes a function to load a texture image "HeightMap1.tga". The texture represents a gray-level height- map of a terrain (Fig. 3a)

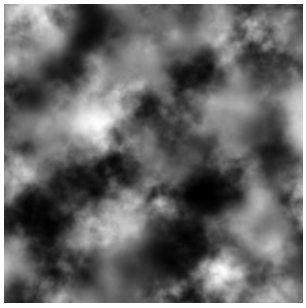

Fig. 3a. Terrain height map.

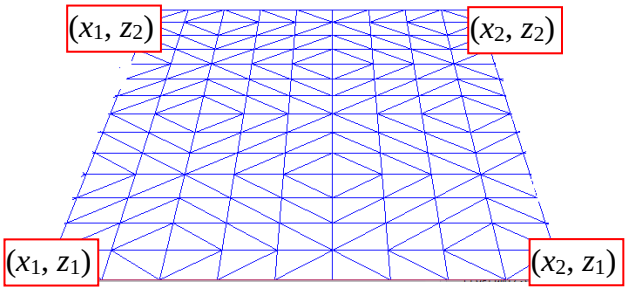

Fig. 3b.

Declare a Sampler2D object to access the height map texture in the evaluation shader. The corresponding statements to map the value to the uniform location have already been included in the application.

```
uniform sampler2D heightMap;
```

In order to map the texture to the tessellated mesh, we require the computation of texture coordinates at each vertex. For this, we compare the x, z components of the vertex (variable `posn` computed in (2) above), with the boundary values of the terrain floor. Refer to the function generateData() in Terrain.cpp. The x values on the terrain floor vary from –45 to +45, and the z-values vary from 0

to −100 (Fig. 3b). Using these min, max values, we can compute the texture coordinates at a vertex (x, z) as

$s = (x - x_1)/(x_2 - x_1)$; $t = (z - z_1)/(z_2 - z_1)$;

Use the above texture coordinates with the Sampler2D object to get a colour value from the height map. Since the height map is a gray-level image, any of its colour components will give the height of the terrain in the range [0, 1]. Scale this value by 10, and assign it to the y-coordinate of the mesh vertex (`posn`). The modified position will then be multiplied by the model-view-projection matrix, and output by the evaluation shader.
The program will output a terrain model shown in Fig. 4.

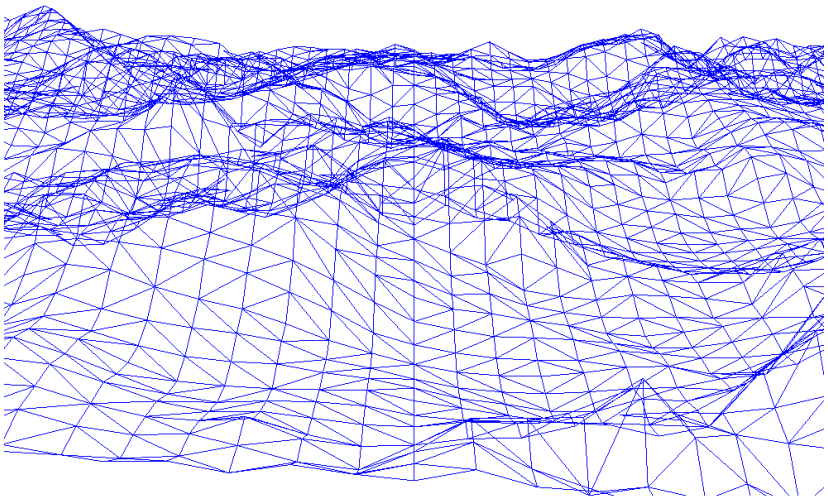

Fig. 4.

(4). We can adjust the level of detail on the terrain based on the distance of the terrain segment from the camera. This is done by modifying the tessellation levels using the average z distance of the patch, inside the tessellation control shader.

Let z_{avg} be the average of the z-coordinates of the 4 patch vertices. The terrain's z values range from 0 to −100. We map these values to tessellation levels 20 to 2 (highest level of detail at z = 0) using the formula

$$\text{level} = \frac{(z_{avg} + 100) * 18}{100} + 2$$

The output should now show a reduction in the tessellation levels with the distance of the patch from the origin. (Fig. 5).

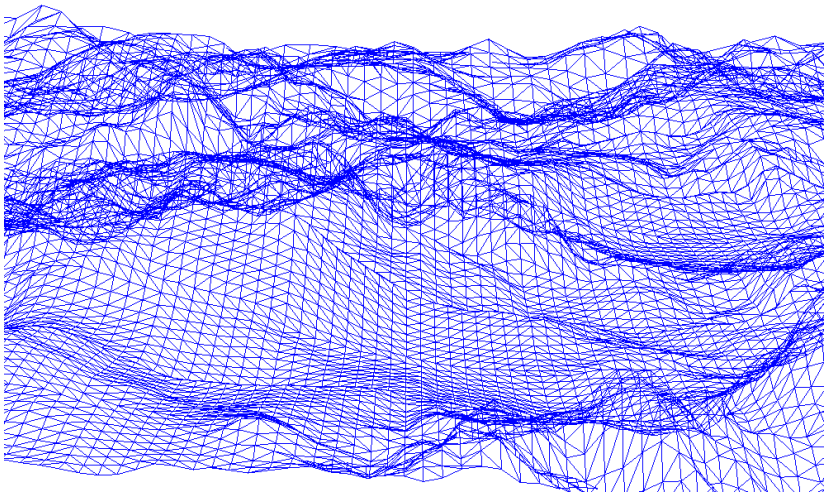

Fig. 5.

(5) The above equation uses terrain's z- values in world coordinates. They must be replaced with z values in eye-coordinates so that they represent the distance from the camera. Also implement a keyboard interface to move the camera through the terrain so that you can see dynamically changing tessellation levels!

(6) In order to render a texture mapped model of the terrain, you will require a geometry shader where you will have access to all three vertices of each triangle to perform lighting calculations and to assign texture coordinates to vertices. Please refer to [2] for more information. When you include a geometry shader, please remember to remove the operation of conversion of points to clip coordinates from the tessellation evaluation shader and perform this operation in the geometry shader.

[1] COSC422 Lecture slides, “1 Introduction:OpenGL4”.
[2] COSC422 Lecture slides, “2 Shader Applications”.