COSC422 Advanced Computer Graphics

Programming Exercise 06

Terrain Rendering

Aim:

This programmin exercise introduces tessellation control and evaluation shaders of the OpenGL-4 pipeline, and demonstrates their applications in mesh tessellation and modelling. This lab also 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.

Reading: The main steps used in the construction of a terrain model are detailed in Slides 38-44 in COSC363 Lecture Notes Lec10_Tessellation.pdf (Reading Material Section)

I. TerrainPatches.cpp:

(1) The program TerrainPatches.cpp draws a set of quads arranged in a rectangular 10x10 grid containing 100 vertices (Fig. 1). The quads represent the terrain's initial ground plane.

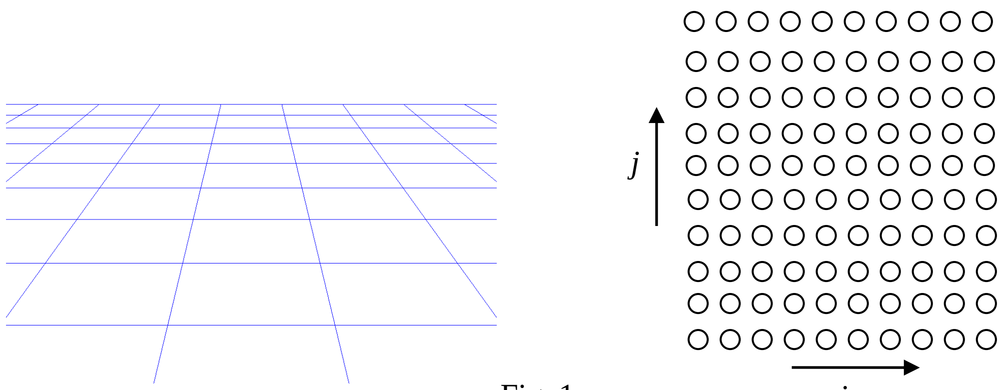

Fig. 1 *i*

(2) Please make the following changes in the program/shaders:

- In the `display()` function, replace the primitive type GL_QUADS in `glDrawElements()` with GL_PATCHES
- In the `initialise()`, load and attach control and evaluation shaders by uncommenting the corresponding statements. Please note that the number of patch vertices is specified as 4 (line 169).
- Convert the vertex shader (TerrainPatches.vert) to a pass-thru shader by removing the multiplication by mvpMatrix. Note that a patch is not a renderable primitive, and therefore the vertices of a patch are not converted to clip coordinates.

- A pass-thru tessellation control shader (TerrainPatches.cont) that sets all tessellation levels to 2 is provided. Please change all tessellation levels to 6.
- The evaluation shader (TerrainPatches.eval) 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 [10]-42. Note that this produces the position of a 3D vertex of a tessellated quad, which is multiplied by "mvpMatrix" to transform it to the clip coordinate space.

The program produces a tessellated floor as shown below (Fig. 2).

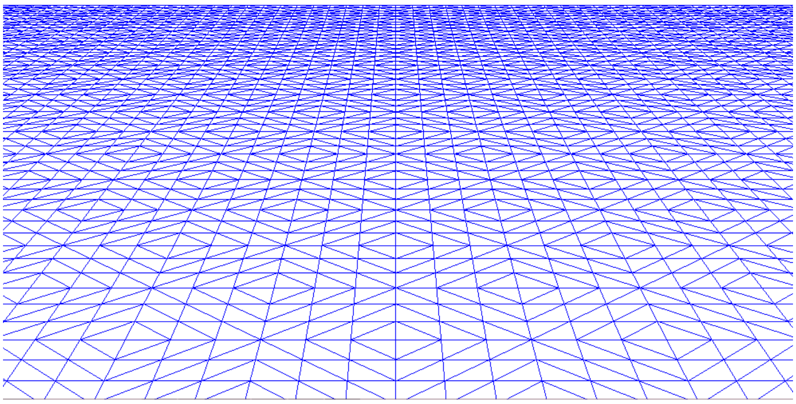

Fig. 2.

(3). The program includes a function to load a texture image "Terrain_hm_01.tga". The texture represents the height-map of a terrain (Fig. 3)

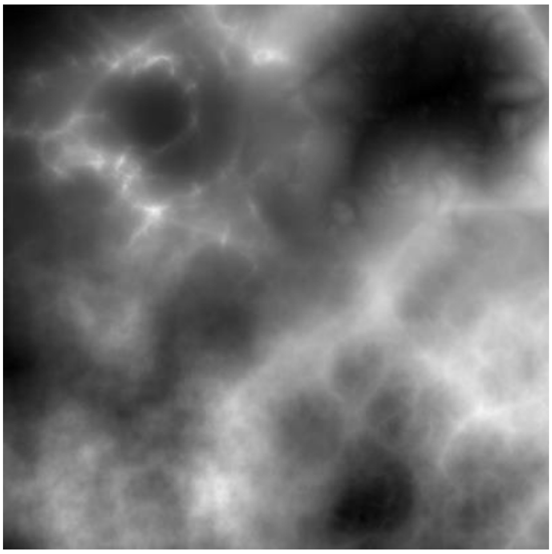

Fig. 3. A terrain height map

(4) In the evaluation shader, the local variable `tcoord` represents the texture coordinates of the current vertex within the terrain's base. The height map texture needs to be mapped to the whole base. Compute the texture coordinates (tcoord.s, tcoord.t) using the equations given on slide Lec[10]-43.

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]:

```
float height = texture(heightMap, tcoord).r;
```

Scale this value by 10, and assign it to the *y*-coordinate of the mesh vertex:

```
posn.y = height * 10.0;
```

As usual, the final position (`posn`) is multiplied by the model-view-projection matrix, and output by the evaluation shader.
The program will output a terrain model as shown in Fig. 4.

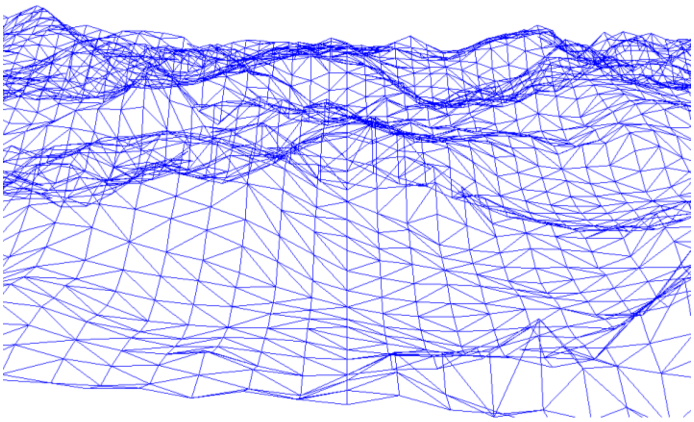

Fig. 4.

Congratulations! You have created a complex terrain model using the tessellation shader stage of the OpenGL-4 pipeline.

(6) Further work: We can adjust the level of detail on the terrain based on the distance of the terrain segment (patch) from the camera. This is done by modifying the tessellation levels inside the tessellation control shader. Pass the camera's position coordinates to the control shader using a uniform variable `eyePos`. Compute the average z coordinate of the current patch as follows:

```
float avg_z = ( gl_in[0].gl_Position.z
              + gl_in[1].gl_Position.z
              + gl_in[2].gl_Position.z
              + gl_in[3].gl_Position.z ) * 0.25;
```

The z-distance of the centre of the patch from the camera is given by

```
dist = distance(eyePos, avg_z);
```

We assign a tessellation level 20 (highest level of detail) to the patch closest to the camera (dist = z_{min}), and a tessellation level 2 to a patch farthest from the camera (dist >= z_{max}). The tessellation level for the current patch can be computed as follows:

```
int level = 20 - int((dist-zmin)*18.0/(zmax-zmin));
```

Set the outer and inner tessellation levels to the value computed above. The output should now show a reduction in the tessellation levels with the distance of the patch from the camera. (Fig. 5).

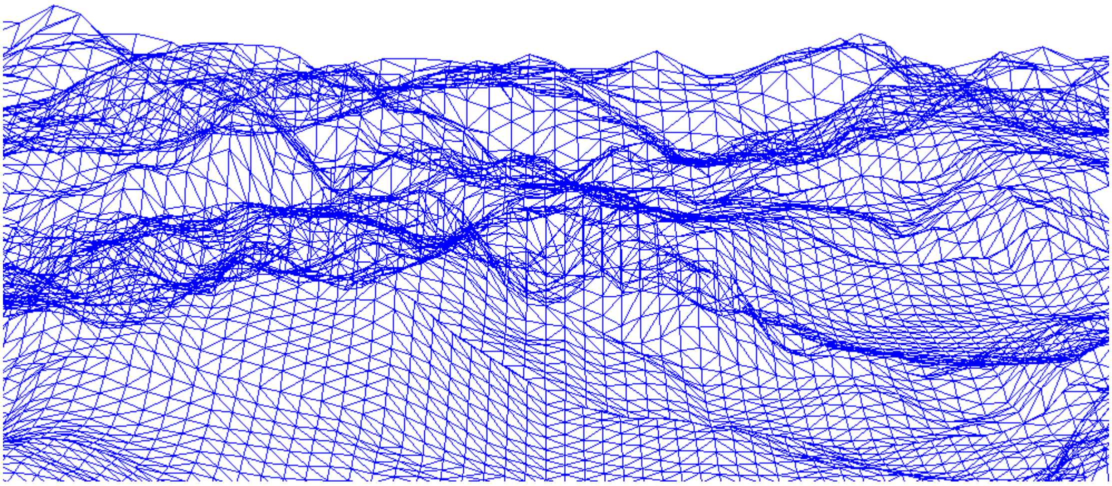

Fig. 5

Ref: [10]: COSC363 Lecture Slides Lec10_Tessellation.pdf