

# COSC363 Computer Graphics

## Lab11: OpenGL-4 Tessellation

### Aim:

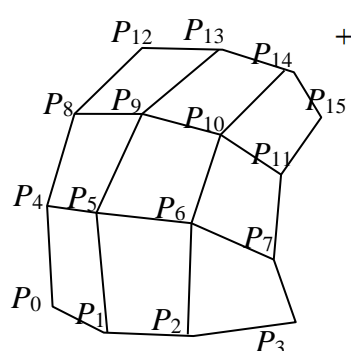
This lab introduces tessellation control and evaluation shaders of the OpenGL-4 pipeline, and demonstrates their applications in mesh tessellation and modelling. This lab also introduces you to the fascinating field of terrain rendering through an example that generates a terrain model using tessellation 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.

### I TeapotPatches.cpp:

In Lab-10, you used bi-quadratic blending functions (Lec10-Slides 16, 17) to construct a Bezier surface patch using 9 patch vertices. Bi-cubic Bezier surfaces are similarly constructed using 4x4 control patches containing 16 patch vertices. The teapot model consists of 32 such control patches, with a total of  $32 \times 16 = 512$  patch vertices. The program `TeapotPatches.cpp` reads the data file "PatchVerts\_Teapot.txt" that contains the coordinates of all patch vertices.

The vertex shader `TeapotPatch.vert` is a simple pass-thru shader. The evaluation shader `TeapotPatch.eval` receives 16 patch vertices in the built-in array `gl_in[]`. These vertices can be combined with the tessellation coordinates  $u, v$  using bi-cubic polynomials (Fig. 1) to obtain vertices of the Bezier patch. The structure of `TeapotPatches.eval` is very similar to that of `QuadPatches.eval` used in Lab-10.



$$\begin{aligned}
 Q = & (1-u)^3 \{ (1-v)^3 P_0 + 3(1-v)^2 v P_1 + 3(1-v) v^2 P_2 + v^3 P_3 \} \\
 & + 3(1-u)^2 u \{ (1-v)^3 P_4 + 3(1-v)^2 v P_5 + 3(1-v) v^2 P_6 + v^3 P_7 \} \\
 & + 3(1-u) u^2 \{ (1-v)^3 P_8 + 3(1-v)^2 v P_9 + 3(1-v) v^2 P_{10} + v^3 P_{11} \} \\
 & + u^3 \{ (1-v)^3 P_{12} + 3(1-v)^2 v P_{13} + 3(1-v) v^2 P_{14} + v^3 P_{15} \}
 \end{aligned}$$

Fig. 1

Please use the above equation to compute the position of the current vertex (`posn`) in the evaluation shader. The output of the program is shown in Fig. 2.

Note that the program does not use a tessellation control shader. A control shader is needed only if either patch vertices or tessellation levels are required to be computed or modified while rendering an object. All patches of the teapot are assigned a constant tessellation level in the application itself. The tessellation levels

are specified inside the `display()` function using the `glPatchParameterfv()` function.

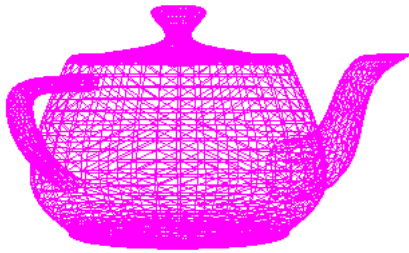


Fig. 2.

The program allows the movement of the camera towards and away from the teapot using up/down arrow keys. Inside the callback function `"special()"`, the minimum camera distance is clamped at 10 units and the maximum distance at 50 units.

Take-home Exercise: Include a tessellation control shader in the shader set. Implement a simple level of detail (LOD) method in the `display()` function to decrease the tessellation levels of the teapot (variable `level`) as the distance from the camera (`camDist`) is increased. Set the minimum value of the tessellation level at 3 (at max camera distance 50), and the maximum tessellation level at 13 (at minimum camera distance 10). The corresponding outputs are shown below (Fig. 3).

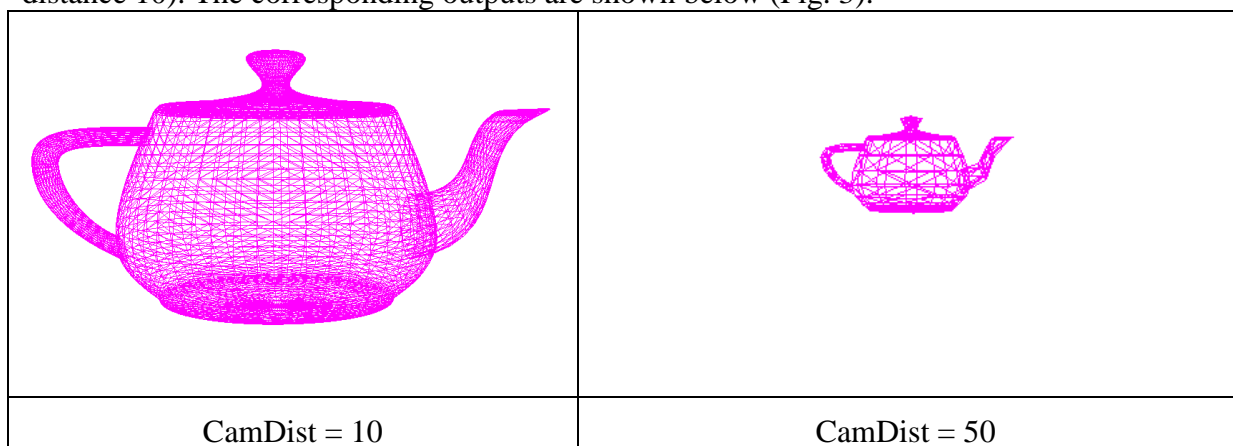


Fig. 3.

## II. TerrainPatches.cpp:

- (1) The program `TerrainPatches.cpp` draws a set of quads arranged in a rectangular 10x10 grid containing 100 vertices (Fig. 4). The quads represent the terrain's initial ground plane.

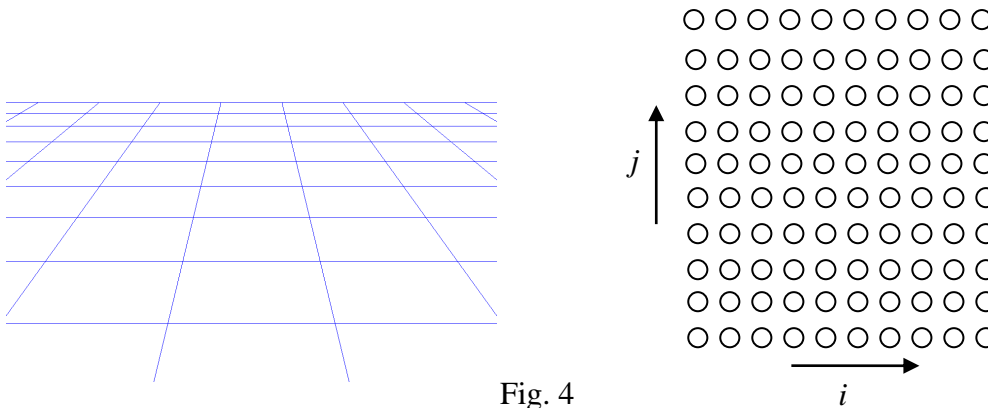


Fig. 4

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

- In `display()`, replace the primitive type in `glDrawElements()` with `GL_PATCHES`
- In `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.
- Convert the vertex shader to a pass-thru shader by removing the multiplication by `mvpMatrix`.
- In the control shader, set all tessellation levels to 6.
- In the evaluation shader, define a bi-linear mapping of mesh vertices ( $u, v$ ) (see Slide [10]-42).

The program produces a tessellated floor as shown below (Fig. 5).

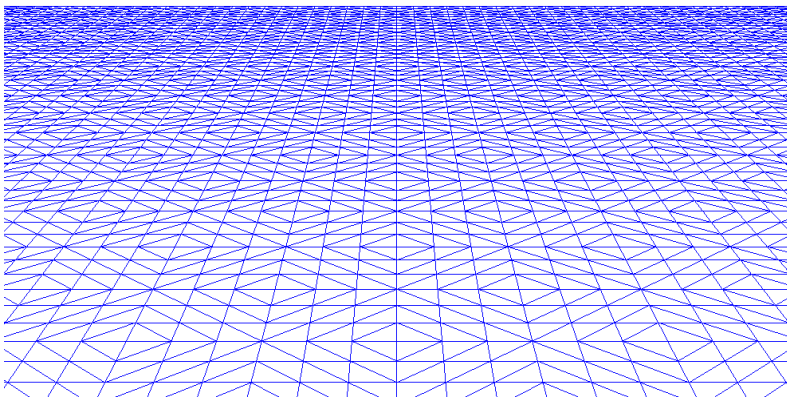


Fig. 5.

- (3). The program includes a function to load a texture image "Terrain\_hm\_01.tga". The texture represents a gray-level height- map of a terrain (Fig. 6)

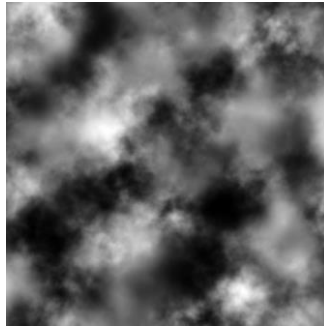


Fig. 6. Terrain height map

- (4) In the evaluation shader, the local variable `tcoord` represents the texture coordinates of the current vertex. 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. 7.

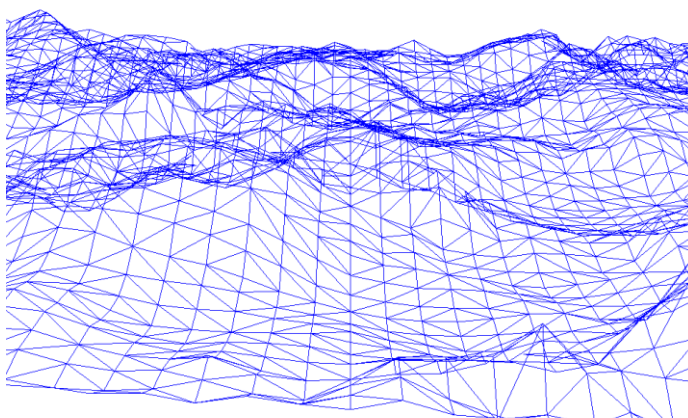


Fig. 7.

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

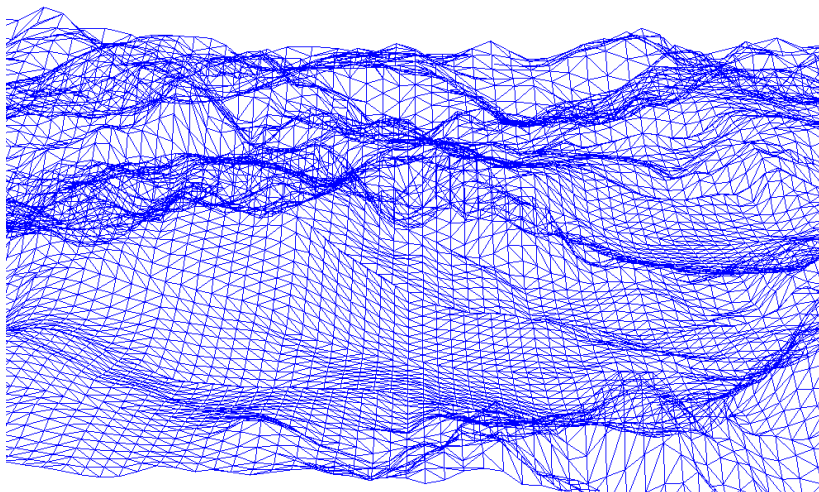
- (6) Further work: We can adjust the level of detail on the terrain based on the distance of the terrain segment from the camera. This is done by modifying the tessellation levels inside the tessellation control shader, based on the average z distance of the patch from the origin:

```
float dist = ( 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 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

```
int level = int((dist+100.0)*0.18)+2;
```

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 origin. (Fig. 8).



(Fig. 8)