COSC422 Advanced Computer Graphics

Programming Exercise 04

Bezier Approximation

The tessellation control shader allows us to modify the structure of the input patch. For example, if the input patch has just 4 vertices forming a quad, we can add additional points inside the tessellation control shader so that the patch becomes a set of 3x3 vertices or 4x4 vertices suitable for generating Bezier approximations inside the evaluation shader.

The file `CubePatches.h` contains the coordinates of 8 vertices of a cube, and the definitions of 6 faces using vertex indices (a total of 24 elements). The program `CubePatches.cpp` generates the display of the wireframe model of the cube (Fig. 1).

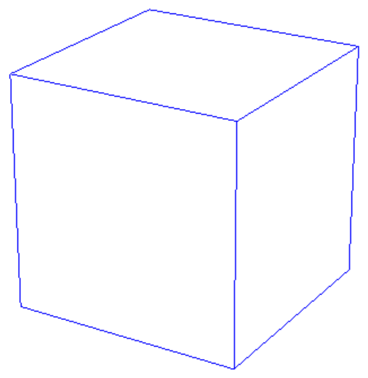


Fig. 1

Modify the program to output patches instead of quads.

```
 glDrawElements(GL_PATHCES, ...)
```

Also, specify the number of vertices in each patch:

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

A patch is not a renderable primitive. We need to create tessellation shaders to generate triangle meshes on each side of the cube. Create a pass-thru tessellation control shader and set all tessellation levels to 2. Create a tessellation evaluation shader to tessellate each quad patch using a bi-linear mapping (Slide [1]-36). You may have to move the transformation of vertices to clip coordinates (multiplication by the `mvpMatrix`) from the vertex shader to the tessellation evaluation shader.

The output should now look similar to that given in Fig. 2. We use this output only to verify that the tessellation shaders are active and are receiving proper inputs.

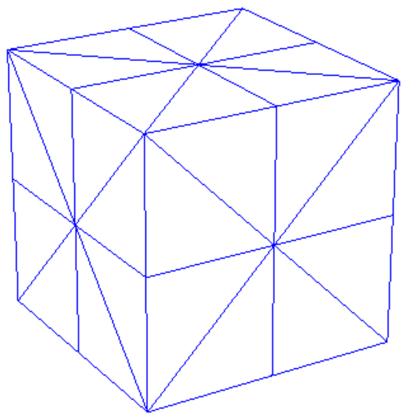


Fig. 2.

We will now modify the structure of the input patch from a quad to a 3x3 grid of points by creating additional patch vertices inside the control shader. The control shader receives all 4 vertices of each patch in the array `gl_in[].gl_Position`. The vertices will have the same order as generated by the application. Inside the control shader, specify the number of output patch vertices as 9:

```
layout(vertices = 9) out;
```

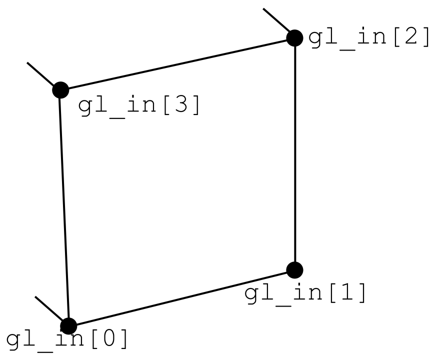


Fig. 3(a)

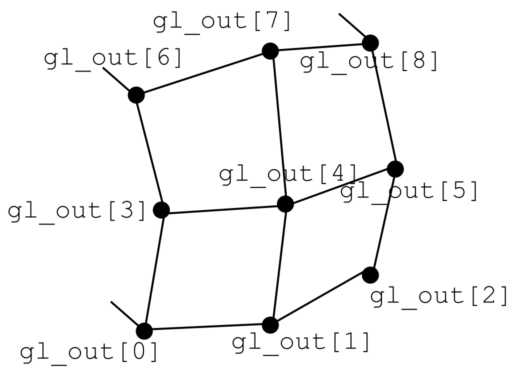


Fig. 3(b)

Fig. 3(a) shows the configuration of input vertices for one of the patches. With the number of output vertices specified as 9, the control shader will execute 9 times, with `gl_InvocationID` varying from 0 to 8, producing an output vertex position in `gl_out[gl_InvocationID].glPosition` on each execution of the shader. These output positions are defined as follows.

The corner vertices are simply copied to the corresponding location in the `gl_out` array: Example:

```
if(gl_InvocationID == 6)
   gl_out[gl_InvocationID].gl_Position = gl_in[3].gl_Position;
```

The midpoints on the edges are obtained by taking the average of the corner vertices and moving that point out of the plane by scaling its coordinates by 1.4. Since averaging requires a division by 2, we minimize the arithmetic operations by adding the end vertices and scaling by 0.7. Example:

```
if(gl_InvocationID == 5) gl_out[gl_InvocationID].gl_Position =
  vec4((gl_in[1].gl_Position.xyz+gl_in[2].gl_Position.xyz)*0.7, 1) ;
```

[Why wouldn't the following expression work?

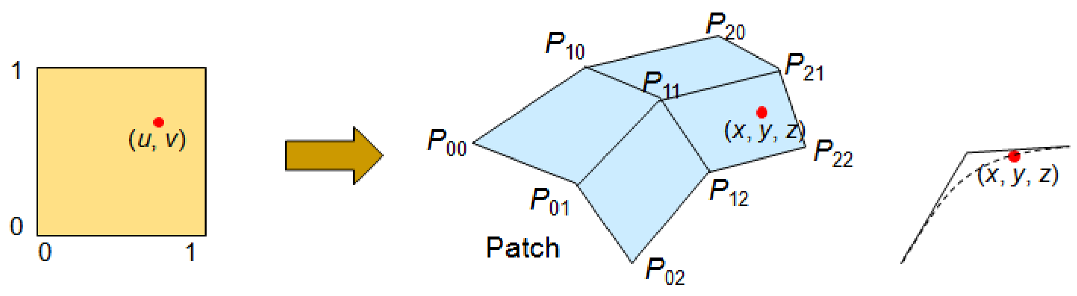
```
  (gl_in[1].gl_Position + gl_in[2].gl_Position)*0.7
```

]

Finally, the midpoint of the patch is got by averaging the four corner points and scaling it by a factor of 2 (i.e., scaling of the sum by 0.5).

Also in the control shader, change all tessellation levels to 8.

The tessellation evaluation shader now receives 9 vertices per patch. We can combine these 9 vertices using the tessellation coordinates *u*, *v* to form a bi-quadratic Bezier patch as shown below:



How can we combine points $P_{11}..P_{33}$ to create the mapping $(u, v) \rightarrow (x, y, z)$?

$$
\begin{aligned}
C = \quad & (1-u)^2 \quad \{ (1-v)^2 P_{00} + 2v(1-v)\, P_{10} + v^2 P_{20} \} \\
& + 2(1-u)u \ \{ (1-v)^2 P_{01} + 2v(1-v)\, P_{11} + v^2 P_{21} \} \\
& + u^2 \ \{ (1-v)^2 P_{02} + 2v(1-v)\, P_{12} + v^2 P_{22} \}
\end{aligned}
$$

If correctly implemented, each patch of the cube will get replaced with a triangular mesh on a Bezier patch as shown in Fig. 4.

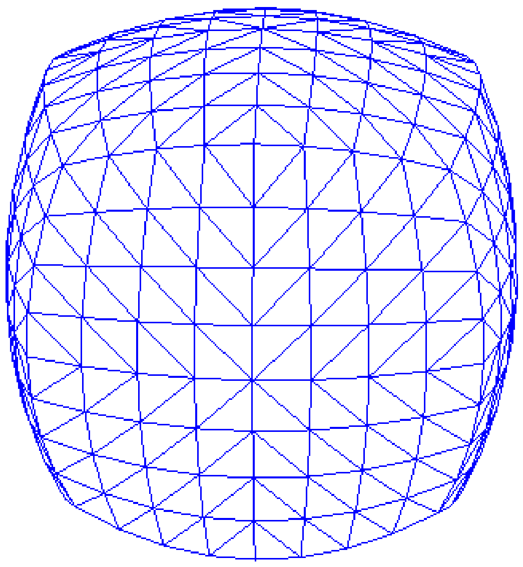


Fig. 4.

[1]: COSC422 Lecture slides, “1 Introduction: OpenGL4 Pipeline”.