THE GEOMETRY SHADER

OUTLINE

- Per-Primitive Processing in OpenGL
- Altering Primitives
- Deleting Primitives
- Adding Primitives

PER-PRIMITIVE PROCESSING

- Vertex shaders operate on one vertex at a time
- Tessellation shaders work on control points, not vertices
- Fragment shaders work on one potential pixel at a time

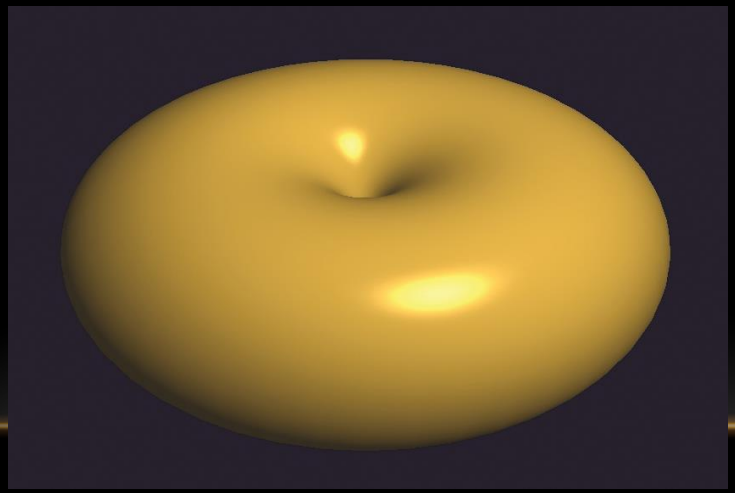
- But, geometry shaders work on one primitive at a time
  - Primitives in OpenGL are points, lines and triangles

PER-PRIMITIVE PROCESSING

- With a geometry shader you can:
  - Access all vertices of a primitive at once
  - Output the same primitive unchanged
  - Output the same primitive changed
  - Output additional primitives
  - Delete the primitive

ALTERING PRIMITIVES

- Can alter primitives to change the shape of an object
  - (When that change can be isolated to the primitives that make up the object)
  - For example, inflating our torus
    - Just scaling it will only increase the size, not change the proportions
    - You could add the surface normal to each vertex in the vertex shader, but… where's the fun in that? We'll use the geometry shader instead



GEOMETRY SHADER CODE (1/2)

```
layout (triangles) in;

in vec3 varyingNormal[];
in vec3 varyingLightDir[];
in vec3 varyingHalfVector[];

out vec3 varyingNormalG;
out vec3 varyingLightDirG;
out vec3 varyingHalfVectorG;

layout (triangle_strip, max_vertices=3) out;

struct PositionalLight
{         vec4 ambient;
          vec4 diffuse;
          vec4 specular;
          vec3 position;
};
struct Material
{         vec4 ambient;
          vec4 diffuse;
          vec4 specular;
          float shininess;
};
```

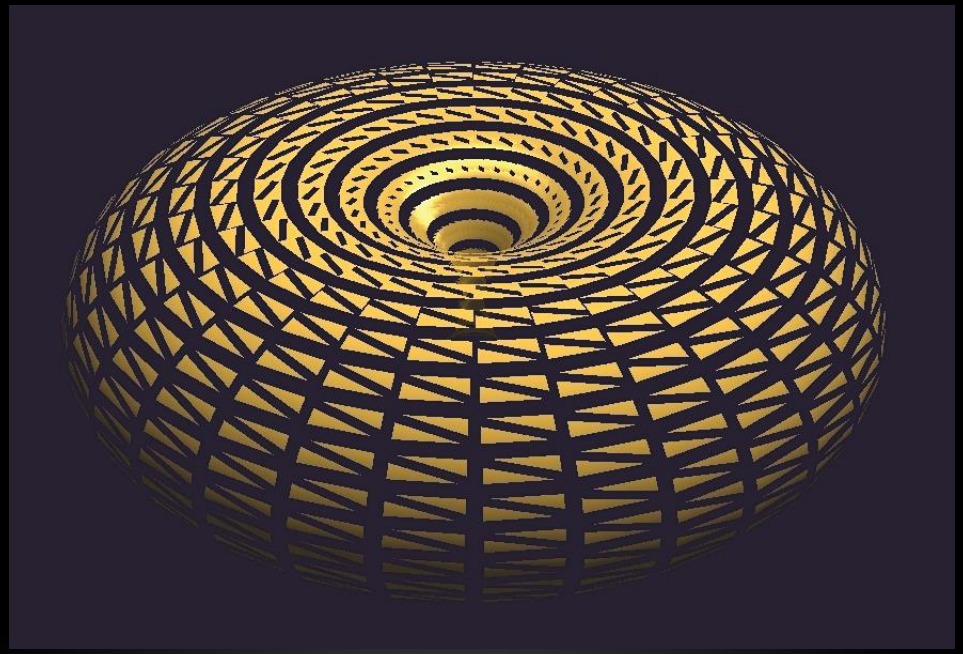
GEOMETRY SHADER CODE (2/2)

```
uniform vec4 globalAmbient;
uniform PositionalLight light;
uniform Material material;
uniform mat4 mv_matrix;
uniform mat4 proj_matrix;
uniform mat4 normalMat;

void main (void)
{         for (int i=0; i<3; i++)
          {         gl_Position = proj_matrix *
                                  (gl_in[i].gl_Position +
                                   normalize(vec4(varyingNormal[i],1.0))*0.4);
                    varyingNormalG = varyingNormal[i];
                    varyingLightDirG = varyingLightDir[i];
                    varyingHalfVectorG = varyingHalfVector[i];
                    EmitVertex();
          }
          EndPrimitive();
}
```

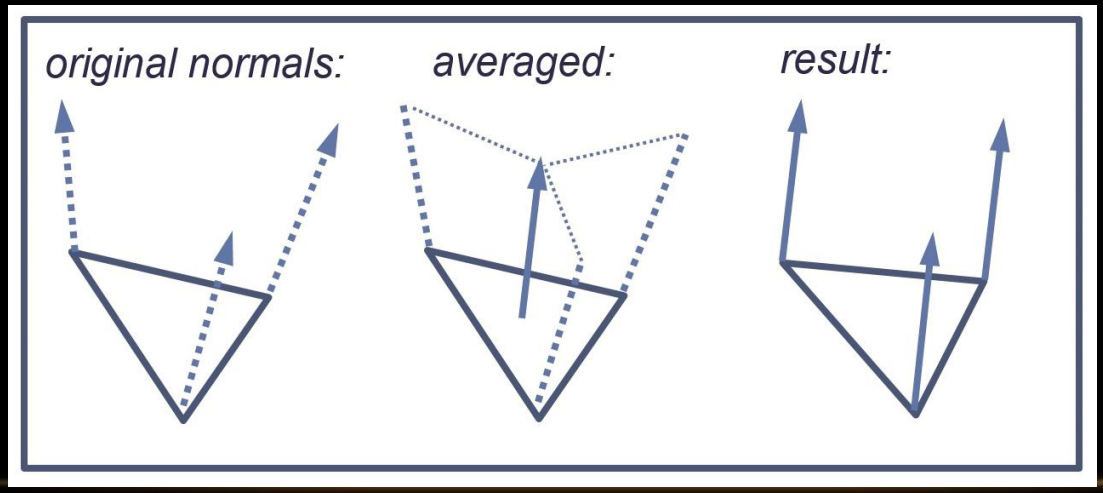
I DO NOT LIKE PUFFY TOROIDS – I WANT TO SEE ONE EXPLODE…

- Instead of just modifying vertices by normals, let's move each individual primitive (triangle) outward along its surface normal



I DO NOT LIKE PUFFY TOROIDS – I WANT TO SEE ONE EXPLODE…

- Sounds good, right?
- But we don't have surface normals, we only have normals defined at the vertices
  - No Problem! We can get around that!
  - We can get an average normal for a surface from the vertex normals and then use those for the direction of movement of the primitive

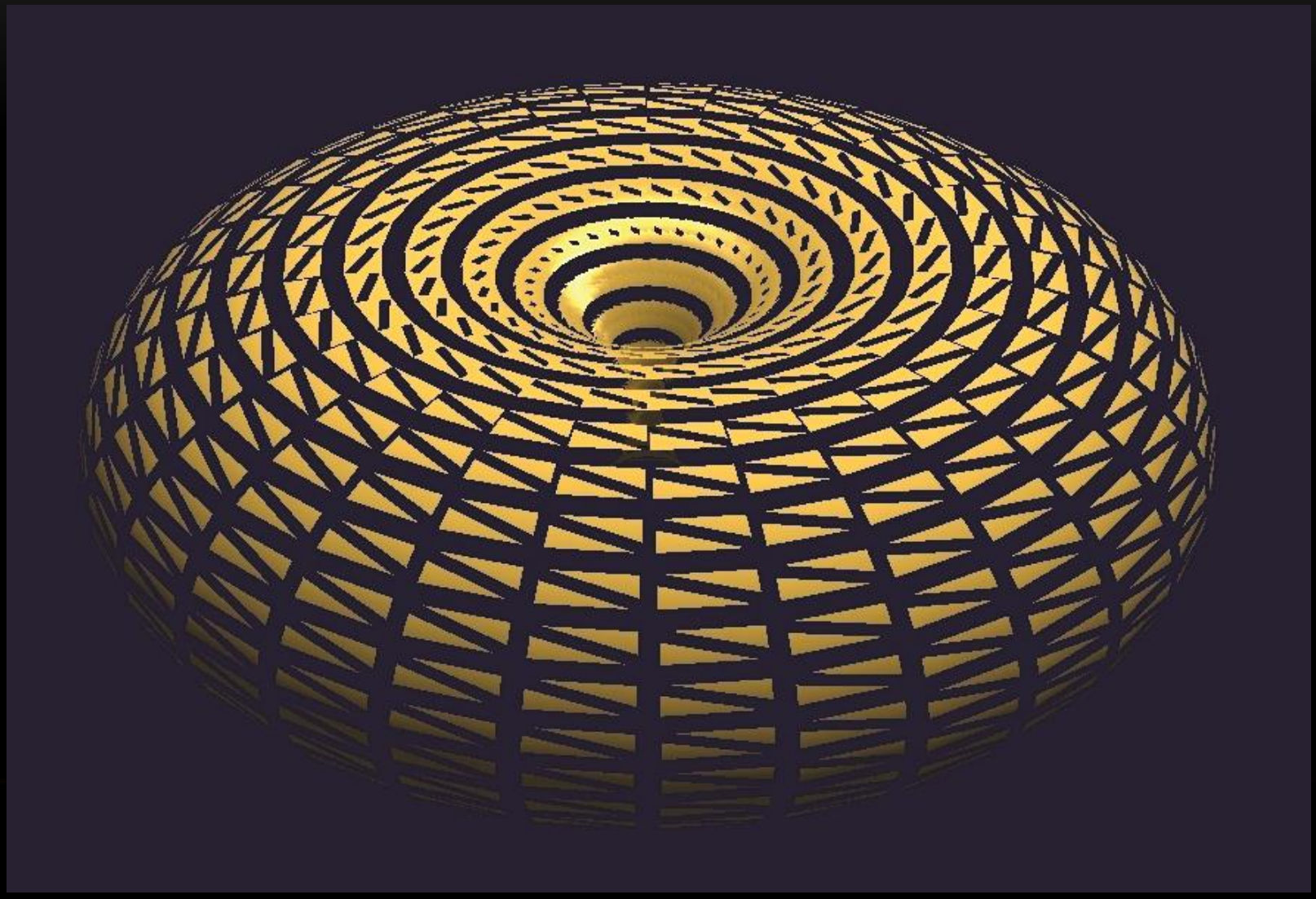


MODIFIED GEOMETRY SHADER MAIN()

```
void main (void)
{       vec4 triangleNormal =
           vec4(((varyingNormal[0]+varyingNormal[1]+varyingNormal[2])/3.0),1.0);

        for (int i=0; i<3; i++)
        {       gl_Position = proj_matrix *
                        (gl_in[i].gl_Position +
                         normalize(triangleNormal)*0.4);
                varyingNormalG = varyingNormal[i];
                varyingLightDirG = varyingLightDir[i];
                varyingHalfVectorG = varyingHalfVector[i];
                EmitVertex();
        }
        EndPrimitive();
}
```

BUT… WHAT'S WRONG WITH THIS PICTURE?



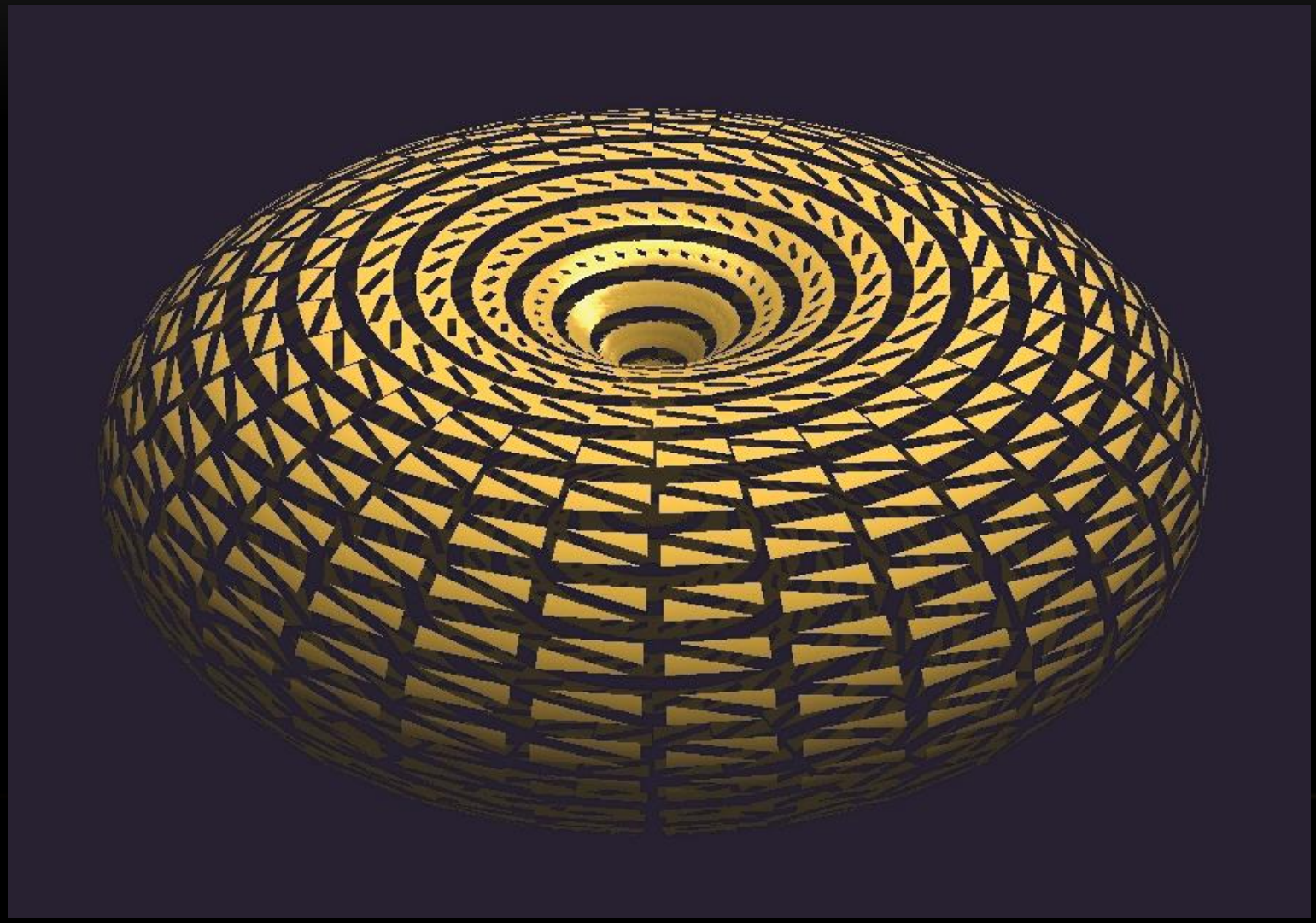
BUT… WHAT'S WRONG WITH THIS PICTURE?

- Yep – those are empty spaces between the primitives, but we can't see the back faces of inside primitives
- This is because back faces are culled
  - One solution: draw the torus twice in display()
    - First time winding order is CCW
    - Second time, winding order is CW

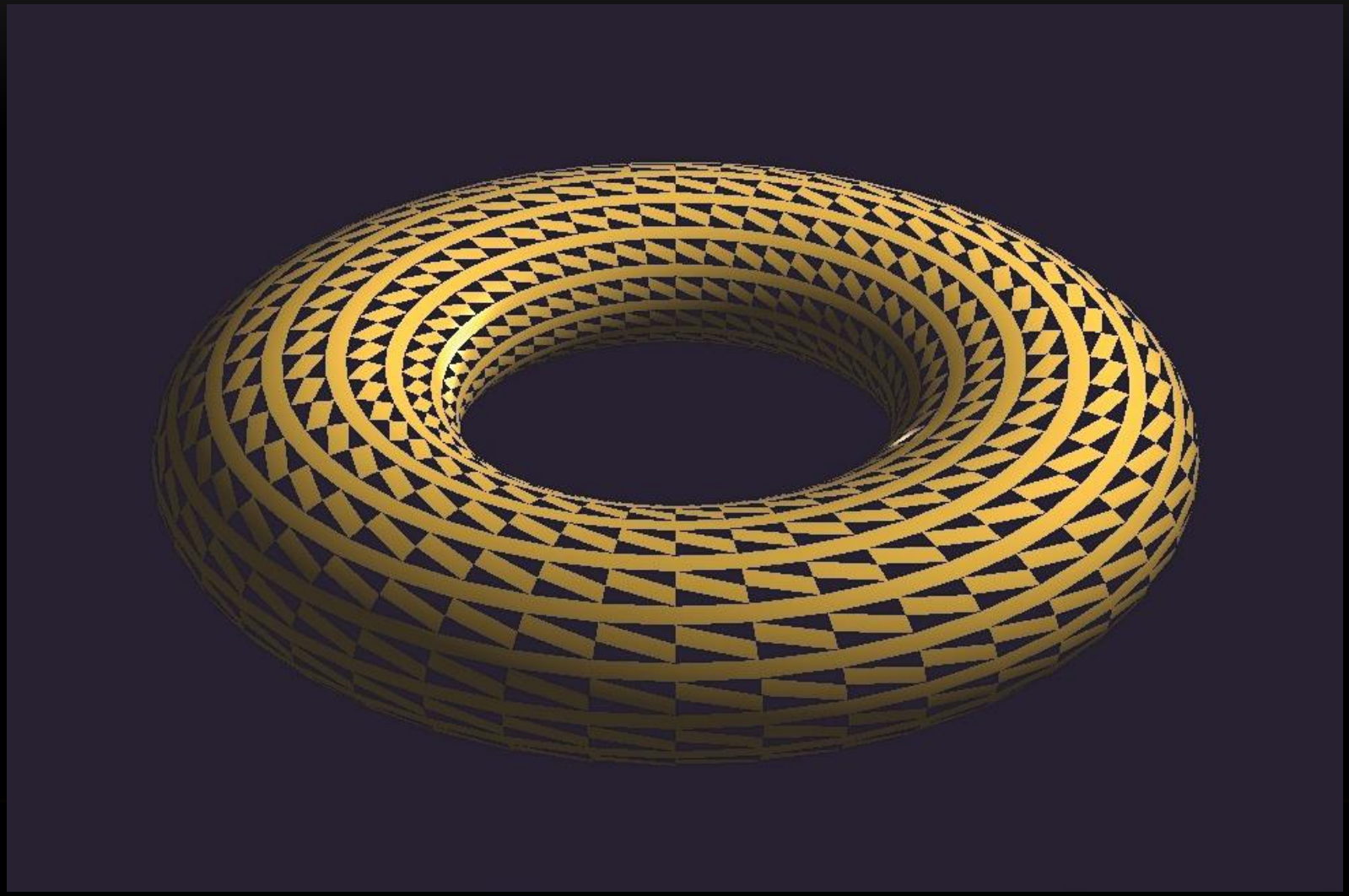
```
gl.glUniform1i(l_location,1);
gl.glFrontFace(GL_CCW);
gl.glDrawArrays(GL_TRIANGLES, 0, numTorusVertices);

gl.glUniform1i(l_location,0);
gl.glFrontFace(GL_CW);
gl.glDrawArrays(GL_TRIANGLES, 0, numTorusVertices);
```

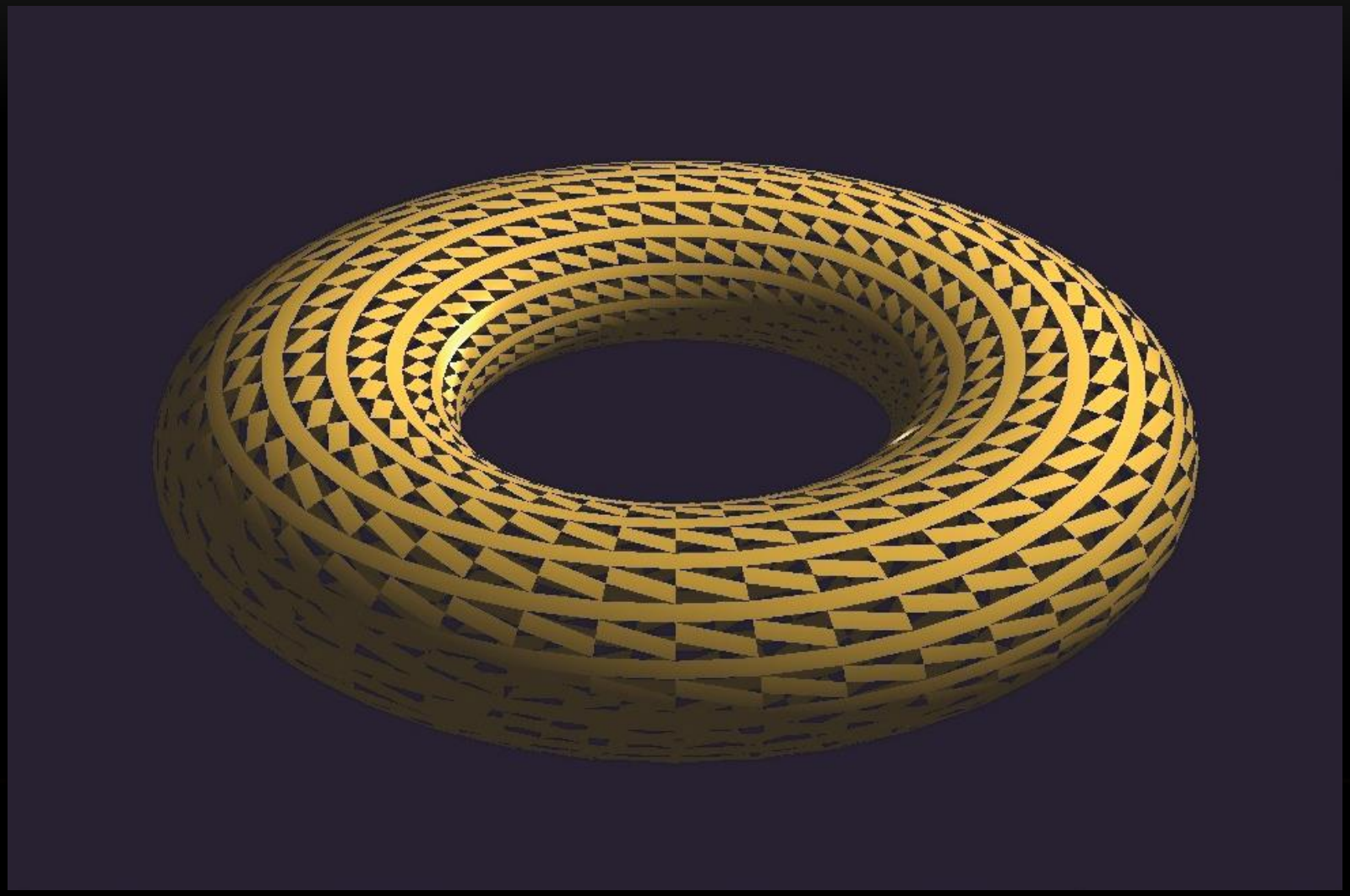
NOW WE SEE THE INTERIOR PRIMITIVES



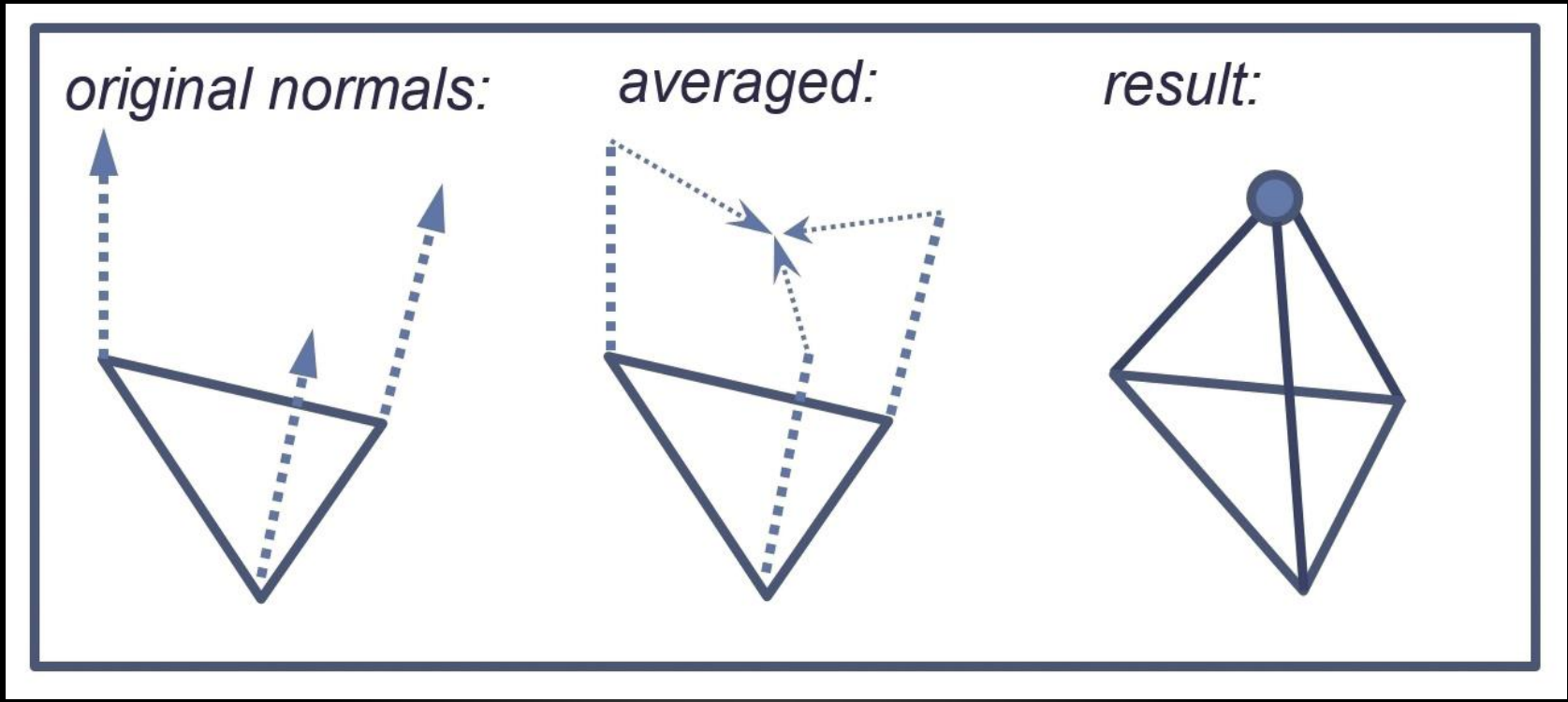
DELETING PRIMITIVES



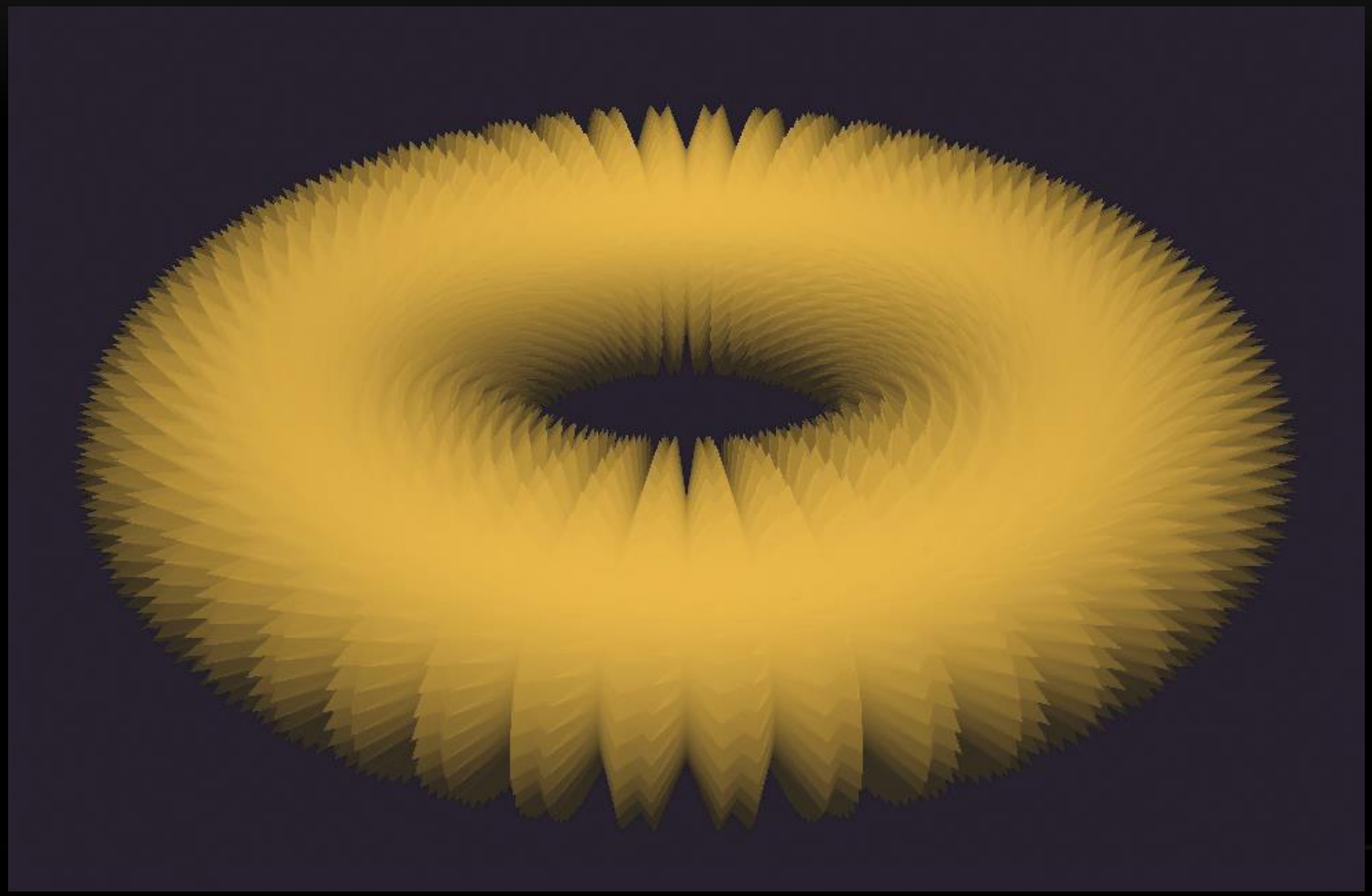
DELETING PRIMITIVES WITH BACK FACES SHOWN



ADDING PRIMITIVES



ADDING PRIMITIVES



SUMMARY

- Per-Primitive Processing in OpenGL
- Altering Primitives
- Deleting Primitives
- Adding Primitives