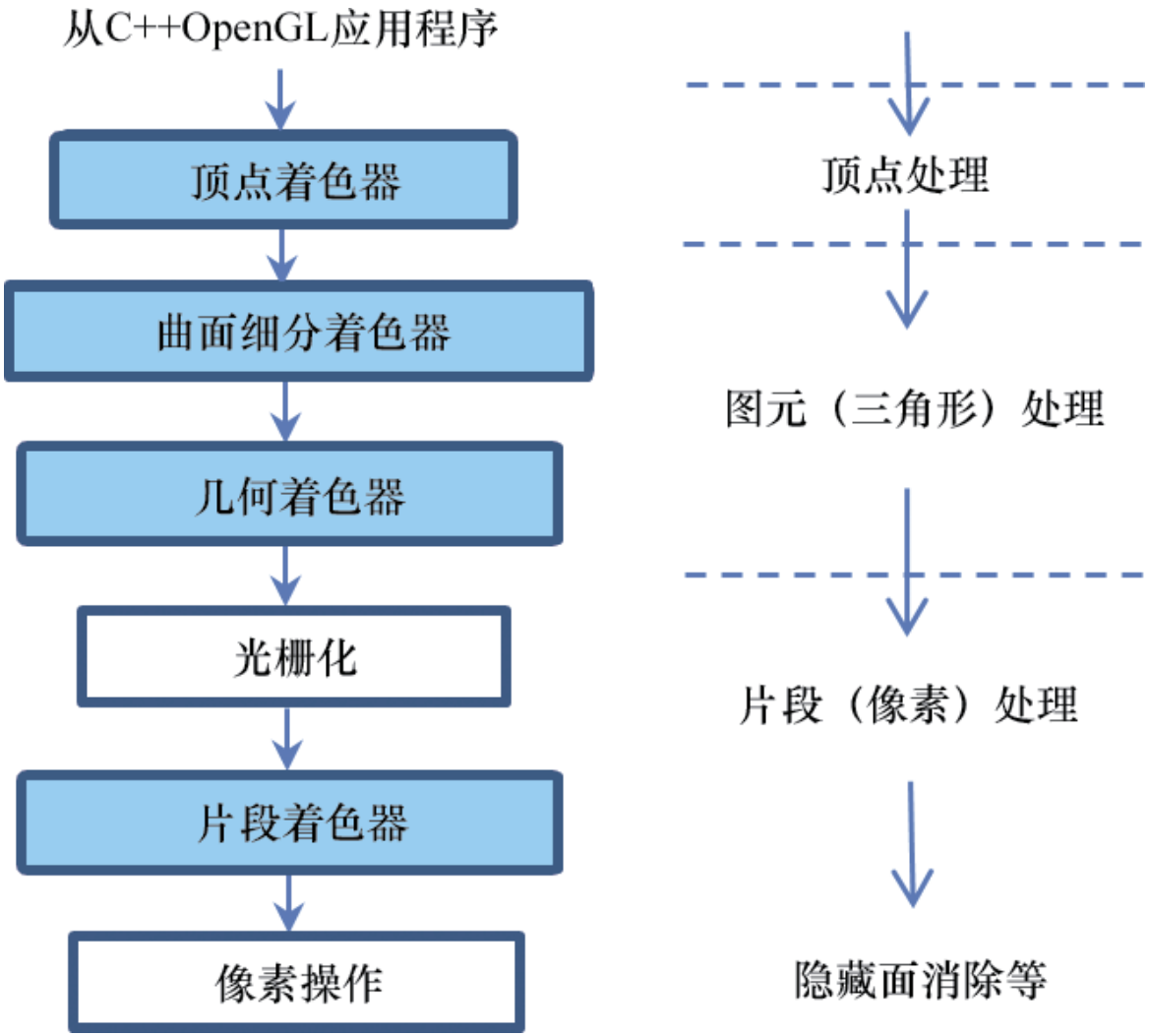
XIAMEN UNIVERSITY

COMPUTER GRAPHICS

第12、13章 曲面细分和几何着色器

陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

OpenGL 图形管线概览



曲面细分

- 曲面细分指的是生成并且操控大量三角形以渲染复杂的形状和表面
- 通过 3 个管线阶段提供：
  （1）曲面细分控制着色器（Tessellation Control Shader，TCS）
  （2）曲面细分器
  （3）曲面细分评估着色器（Tessellation Evaluation Shader，TES）

- **曲面细分控制着色器**配置曲面细分器要构建什么样的三角形网格
- **曲面细分评估着色器**允许我们以各种方式操控网格

曲面细分

- 让我们从一个简单的应用程序开始，该应用程序只使用曲面细分器创建顶点的三角形网格

  （1）C++/OpenGL 应用程序：创建一个相机和相关的 MVP 矩阵

  （2）顶点着色器：在这个例子中基本上什么都不做，顶点将在曲面细分器中生成。

  （3）曲面细分控制着色器：指定曲面细分器要构建的网格。

  （4）曲面细分评估着色器：将 MVP 矩阵应用于网格中的顶点。

  （5）片段着色器：只需为每个像素输出固定颜色

C++ / OpenGL application

```
GLuint createShaderProgram(
                        const char *vp, const char *tCS, const char *tES, const char *fp )  {
    . . .
    string tcShaderStr = readShaderSource(tCS);
    string teShaderStr = readShaderSource(tES);
    const char *tcShaderSrc = tcShaderStr.c_str();
    const char *teShaderSrc = teShaderStr.c_str();
    GLuint tcShader = glCreateShader(GL_TESS_CONTROL_SHADER);
    GLuint teShader = glCreateShader(GL_TESS_EVALUATION_SHADER);
    glShaderSource(tcShader, 1, &tcShaderSource, NULL);
    glShaderSource(teShader, 1, &teShaderSource, NULL);
    glCompileShader(tcShader);
    glCompileShader(teShader);
    . . .
    glAttachShader(renderingProgram, tcShader);
    glAttachShader(renderingProgram, teShader);
    . . .
}
void display(GLFWwindow* window)  {
    . . .
    glPatchParameteri(GL_PATCH_VERTICES, 1);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glDrawArrays(GL_PATCHES, 0, 1);
}
```

Vertex Shader

```
#version 430
uniform mat4 mvp;
void main(void)  {  }
```

Tessellation Control Shader

```
#version 430
uniform mat4 mvp;
layout (vertices = 1) out;
void main(void)
{   gl_TessLevelOuter[0] = 6;
    gl_TessLevelOuter[1] = 6;
    gl_TessLevelOuter[2] = 6;
    gl_TessLevelOuter[3] = 6;
    gl_TessLevelInner[0] = 12;
    gl_TessLevelInner[1] = 12;
}
```

Tessellation Evaluation Shader

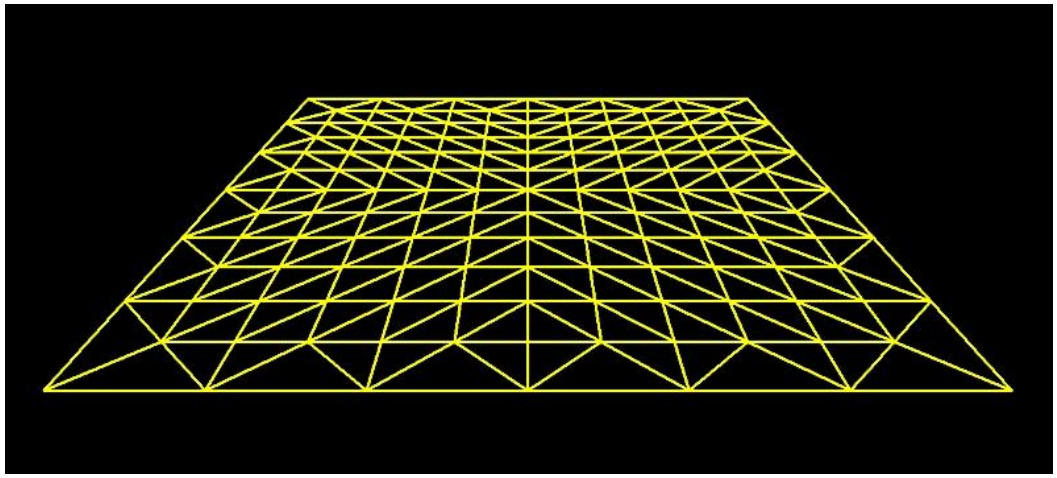
```
#version 430
uniform mat4 mvp;
layout (quads, equal_spacing, ccw)  in;
void main (void)
{   float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    gl_Position = mvp * vec4(u,0,v,1);
}
```

Fragment Shader

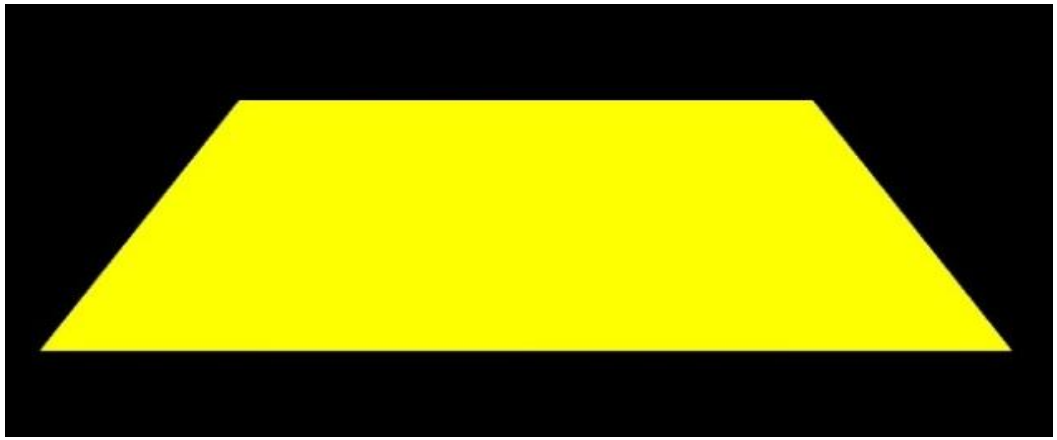
```
#version 430
out vec4 color;
uniform mat4 mvp;
void main(void)
{   color = vec4(1.0, 1.0, 0.0, 1.0);  // yellow
}
```

曲面细分结果

Result:



Result if changing from GL_LINE to GL_FILL:



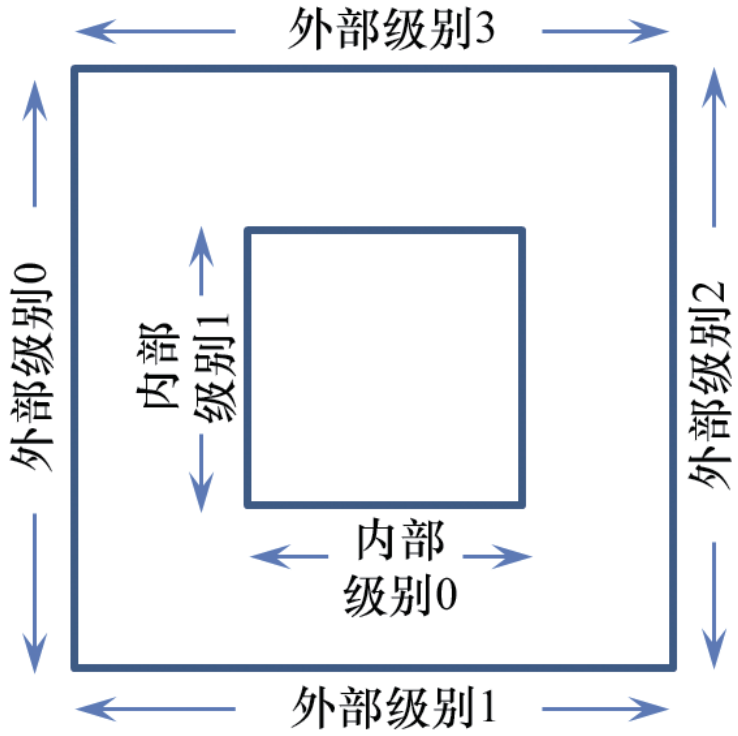
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

细分级别

- 曲面细分器生成由两个参数定义的顶点网格： 内层级别和外层级别

Tessellation Control Shader

```
#version 430
uniform mat4 mvp;
layout (vertices = 1) out;
void main(void)
{   gl_TessLevelOuter[0] = 6;
    gl_TessLevelOuter[1] = 6;
    gl_TessLevelOuter[2] = 6;
    gl_TessLevelOuter[3] = 6;
    gl_TessLevelInner[0] = 12;
    gl_TessLevelInner[1] = 12;
}
```



细分评估着色器

- 曲面细分器生成的顶点将被发送到评估着色器
- 曲面细分评估着色器对曲面细分器生成的每个顶点执行一次
- 使用内置变量 gl_TessCoord
- 曲面细分网格被指定位于 *xz* 平面
- gl_TessCoord 的值，范围为 0.0～1.0

Tessellation Evaluation Shader

```
#version 430
uniform mat4 mvp;
layout (quads, equal_spacing, ccw)  in;
void main (void)
{   float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    gl_Position = mvp * vec4(u,0,v,1);
}
```

几何着色器

- 顶点着色器允许一次操作一个顶点，片段着色器一次可以操作一个片段，几何着色器却可以一次操作一个图元
- 几何着色器允许一次性访问图元中的所有顶点，然后：
  - 输出相同的图元保持不变
  - 输出修改了顶点位置的相同类型图元
  - 输出不同类型的图元
  - 输出更多的其他图元
  - 删除图元（根本不输出）

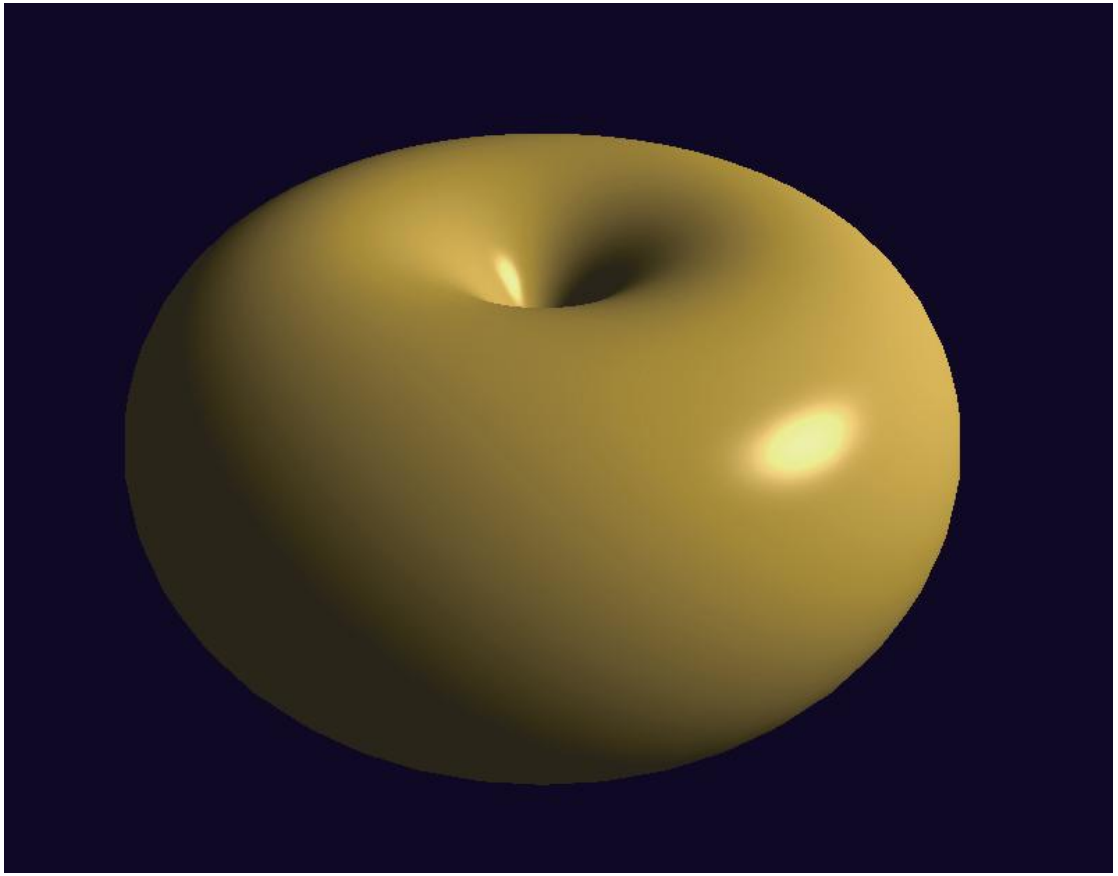
修改图元

Geometry Shader

```
#version 430
layout (triangles) in;
in vec3 varyingNormal[ ];    // inputs from the vertex shader (note they are arrays)
in vec3 varyingLightDir[ ];
in vec3 varyingHalfVector[ ];

out vec3 varyingNormalG;  // outputs through the rasterizer to the fragment shader
out vec3 varyingLightDirG;                          (note they are scalars)
out vec3 varyingHalfVectorG;
layout (triangle_strip, max_vertices=3) out;
. . .
void main (void)          // move vertices outward along the surface normal
{   for (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();
}
```

修改图元

- “充气”的环面，顶点由几何着色器修改



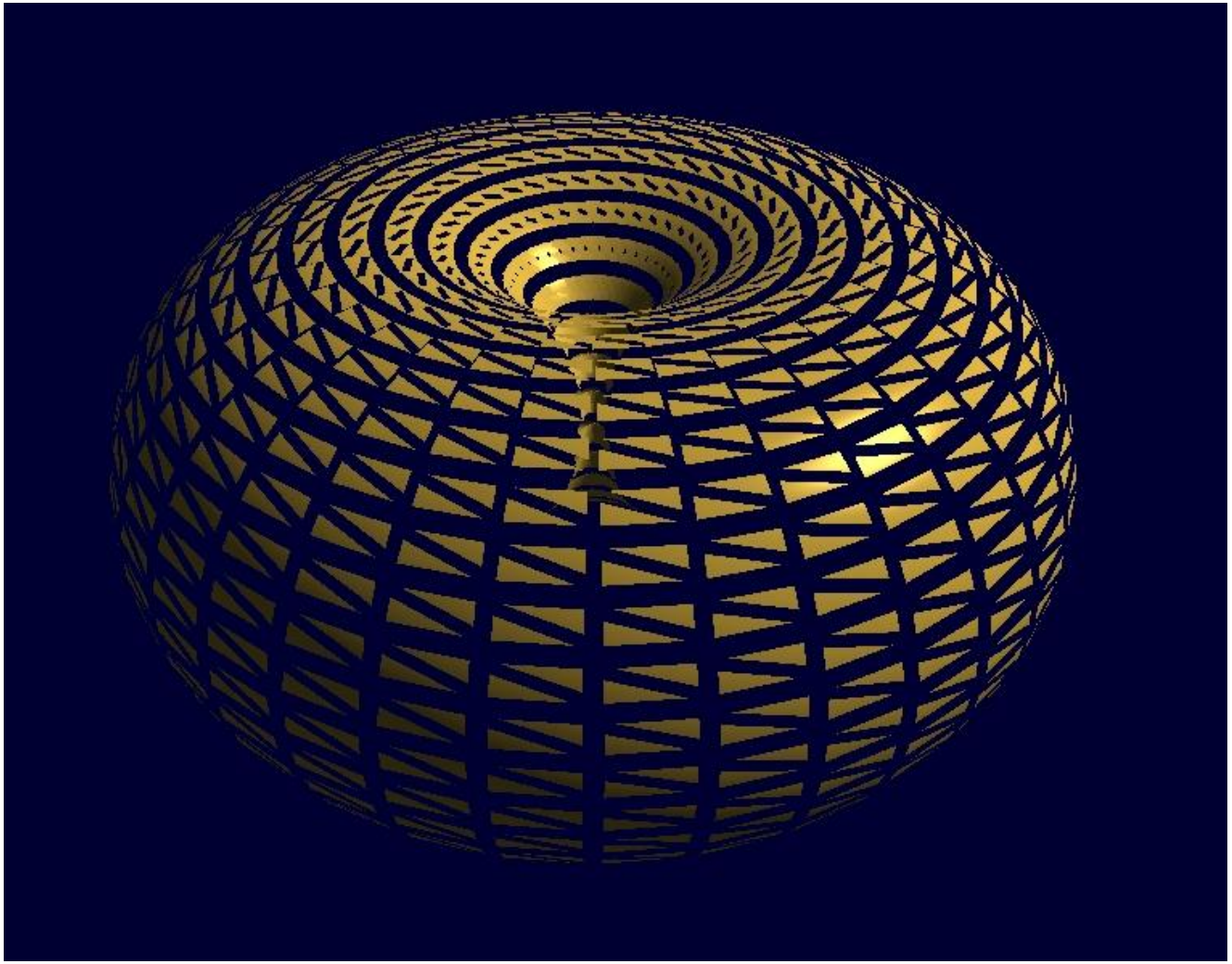
修改图元

Geometry Shader

```
void main (void)
{  // average the three triangle vertex normals, creating a single triangle normal
   vec4 triangleNormal =
      vec4(((varyingNormal[0] + varyingNormal[1] + varyingNormal[2]) / 3.0),1.0);
   // move all three vertices outward along the same normal
   for (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();
}
```

修改图元

- “爆炸”的环面

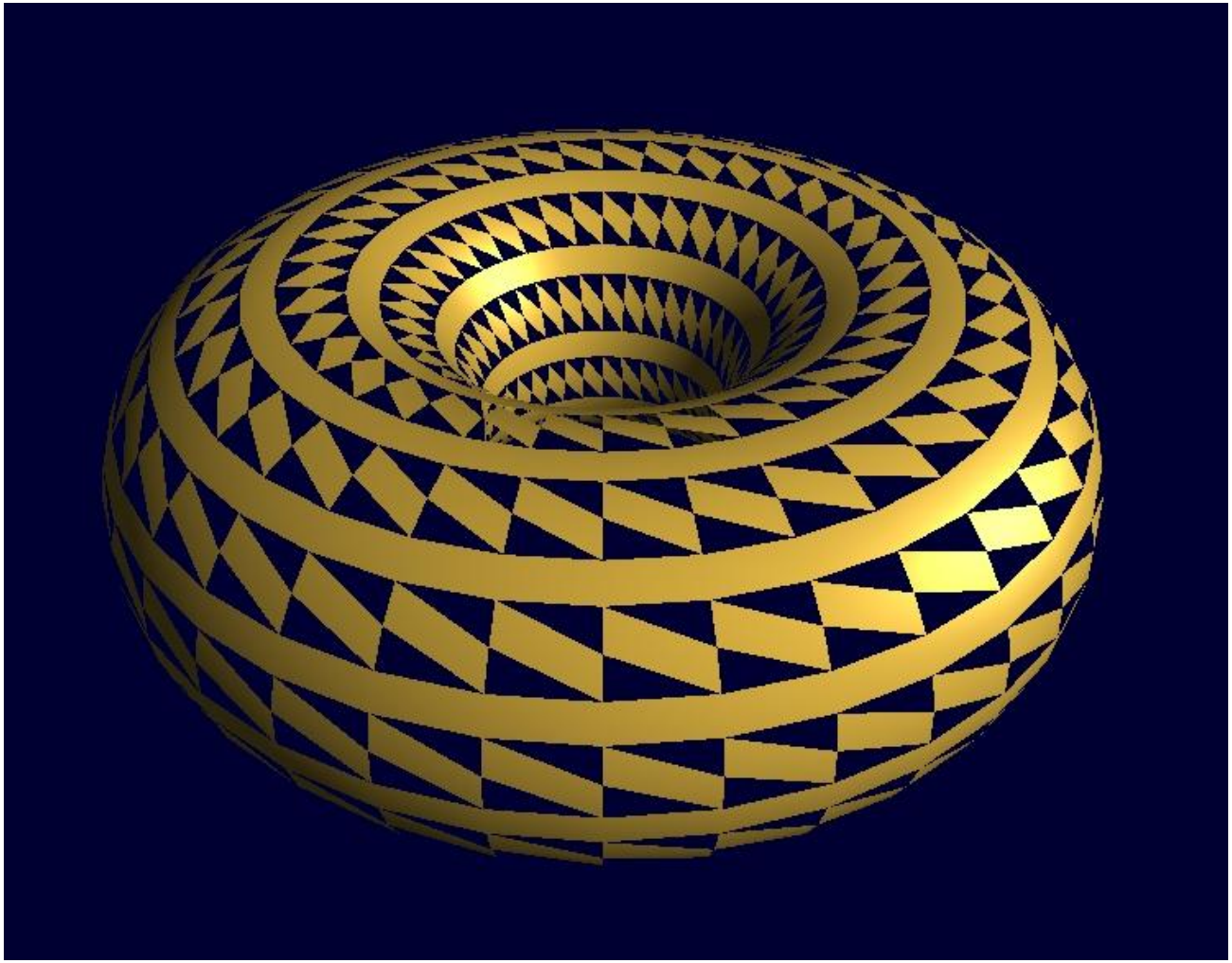


删除图元

Geometry Shader

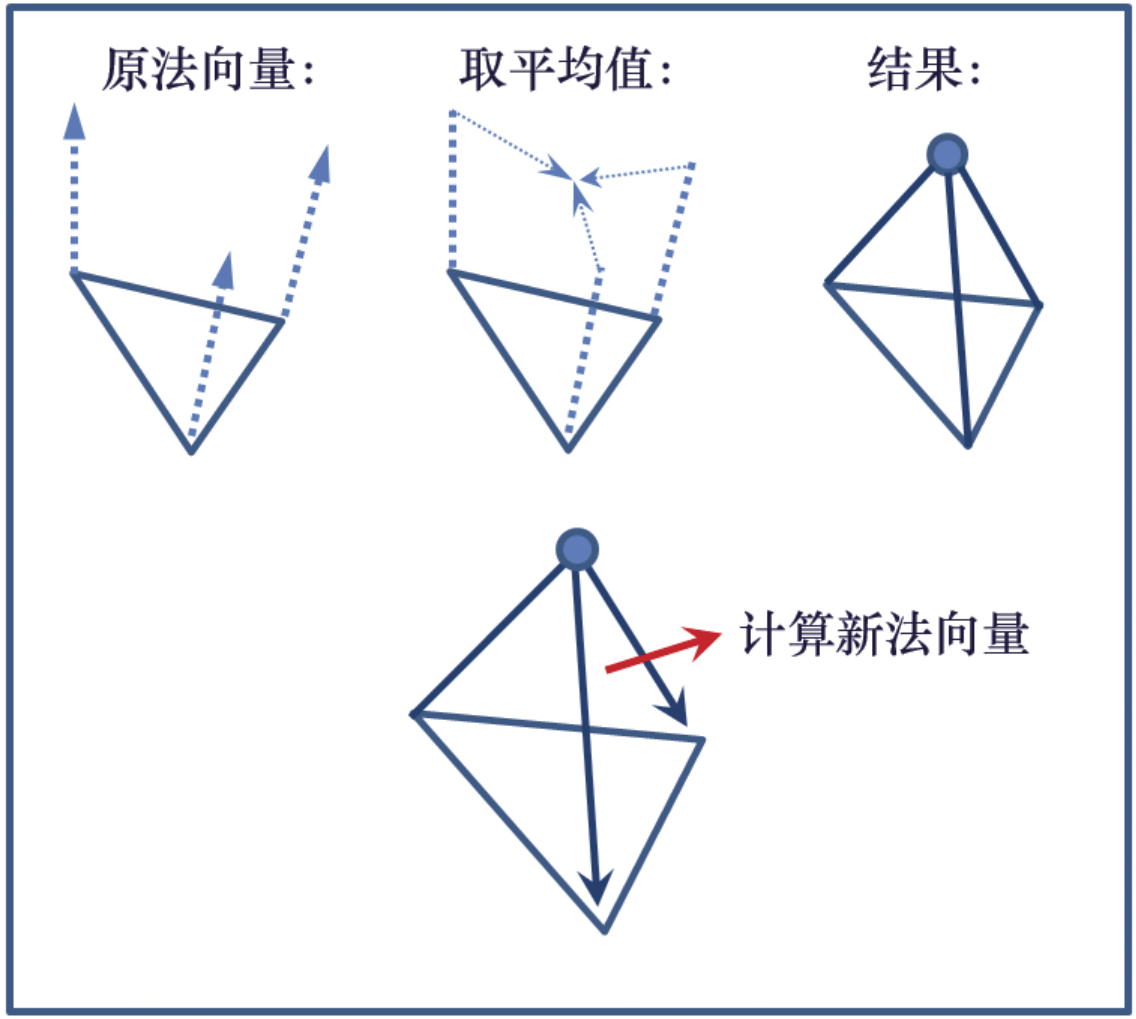
```
//  inputs, outputs, and uniforms as before
. . .
void main (void)
{   if ( mod(gl_PrimitiveIDIn,3) != 0 )
    {   for (int i=0; i<3; i++)
        {   gl_Position = proj_matrix * gl_in[i].gl_Position;
            varyingNormalG = varyingNormal[i];
            varyingLightDirG = varyingLightDir[i];
            varyingHalfVectorG = varyingHalfVector[i];
            EmitVertex();
    }   }
    EndPrimitive();
}
```

删除图元



添加图元

- 将三角形转换为三棱锥



Geometry Shader

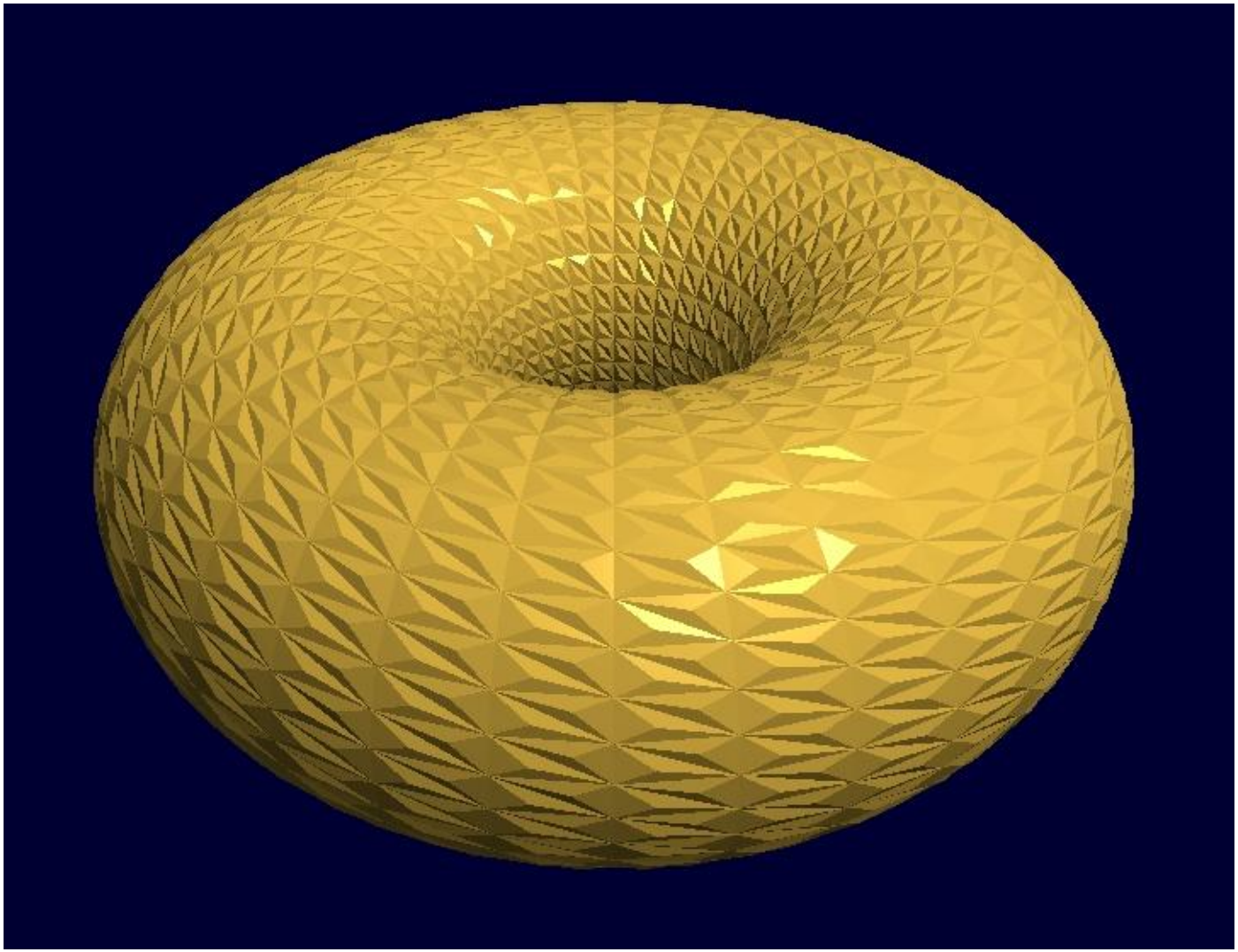
```
. . .
void main (void)
{  // offset the three triangle vertices by the original surface normal
   vec3 sp0 = gl_in[0].gl_Position.xyz + varyingOriginalNormal[0]*sLen;
   vec3 sp1 = gl_in[1].gl_Position.xyz + varyingOriginalNormal[1]*sLen;
   vec3 sp2 = gl_in[2].gl_Position.xyz + varyingOriginalNormal[2]*sLen;

   // compute the new points comprising a small pyramid
   newPoints[0] = gl_in[0].gl_Position.xyz;
   newPoints[1] = gl_in[1].gl_Position.xyz;
   newPoints[2] = gl_in[2].gl_Position.xyz;
   newPoints[3] = (sp0+sp1+sp2) / 3.0;  // spike point

   // compute the directions from the vertices to the light
   lightDir[0] = light.position – newPoints[0];
   lightDir[1] = light.position – newPoints[1];
   lightDir[2] = light.position – newPoints[2];
   lightDir[3] = light.position – newPoints[3];

   //  build three new triangles to form a small pyramid on the surface
   makeNewTriangle(0,1);   // the third point is always the spike point
   makeNewTriangle(1,2);
   makeNewTriangle(2,0);
}
```

continued . . .

Geometry Shader (continued)

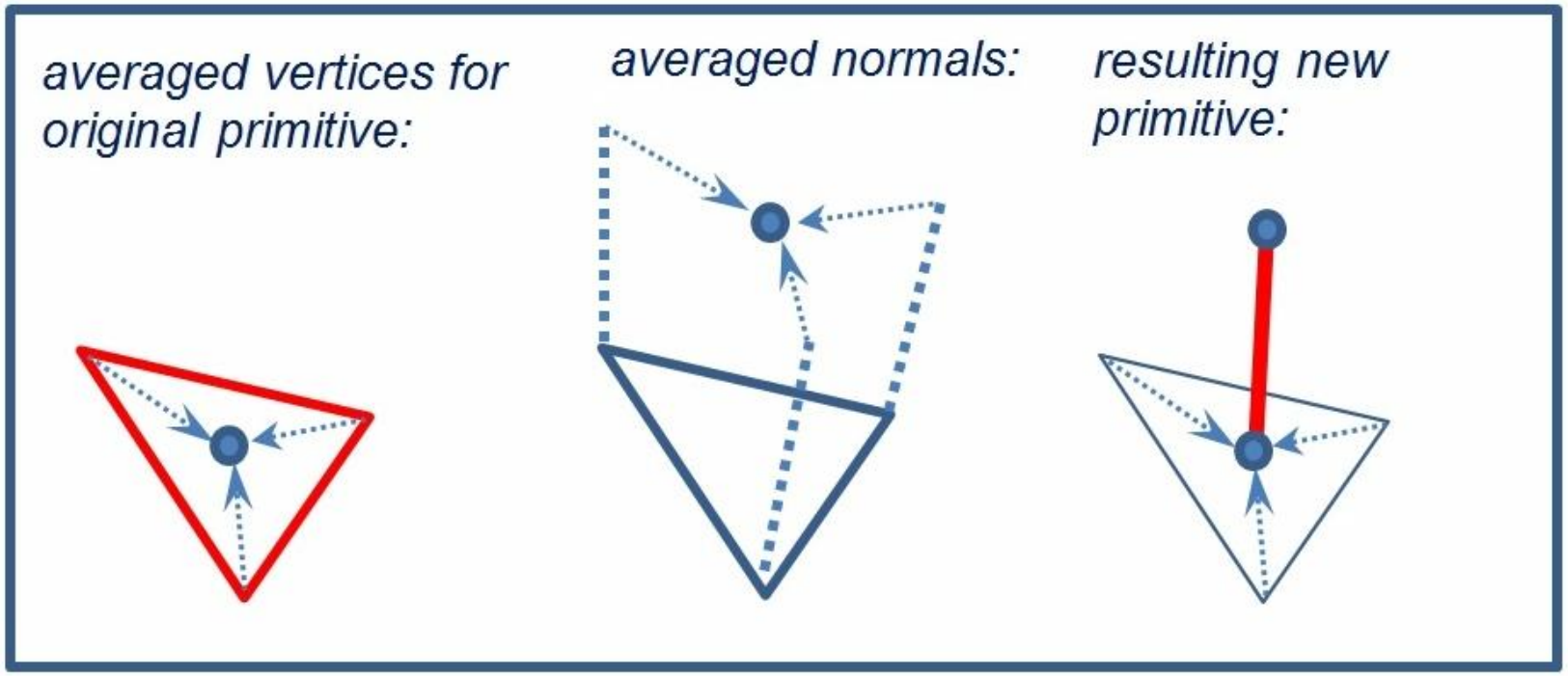
```
. . .
vec3 newPoints[ ], lightDir[ ];
float sLen = 0.01;     // sLen is the "spike length", the height of the small pyramid

void setOutputValues(int p, vec3 norm)
{   varyingNormal = norm;
    varyingLightDir = lightDir[p];
    varyingVertPos = newPoints[p];
    gl_Position = proj_matrix * vec4(newPoints[p], 1.0);
}

void makeNewTriangle(int p1, int p2)
{   // generate surface normal for this triangle
    vec3 c1 = normalize(newPoints[p1] - newPoints[3]);
    vec3 c2 = normalize(newPoints[p2] - newPoints[3]);
    vec3 norm = cross(c1,c2);

    // generate and emit the three vertices
    setOutputValues(p1, norm); EmitVertex();
    setOutputValues(p2, norm); EmitVertex();
    setOutputValues(3, norm); EmitVertex();
    EndPrimitive();
}
```

添加图元



改变图元

- 将三角形图元更改为线图元



Geometry Shader

```
void main(void)
{   vec3 op0 = gl_in[0].gl_Position.xyz;    // original triangle vertices
    vec3 op1 = gl_in[1].gl_Position.xyz;
    vec3 op2 = gl_in[2].gl_Position.xyz;
    vec3 ep0 = gl_in[0].gl_Position.xyz + varyingNormal[0]*sLen;  // offset vertices
    vec3 ep1 = gl_in[1].gl_Position.xyz + varyingNormal[1]*sLen;
    vec3 ep2 = gl_in[2].gl_Position.xyz + varyingNormal[2]*sLen;

    // compute the new points comprising a small line segment
    vec3 newPoint1 = (op0 + op1 + op2)/3.0;          // original (start) point
    vec3 newPoint2 = (ep0 + ep1 + ep2)/3.0;          // end point
    gl_Position = proj_matrix * vec4(newPoint1, 1.0);

    varyingVertPosG = newPoint1;
    varyingLightDirG = light.position - newPoint1;
    varyingNormalG = varyingNormal[0];
    EmitVertex();

    gl_Position = proj_matrix * vec4(newPoint2, 1.0);
    varyingVertPosG = newPoint2;
    varyingLightDirG = light.position - newPoint2;
    varyingNormalG = varyingNormal[1];
    EmitVertex();
    EndPrimitive();
}
```

改变图元