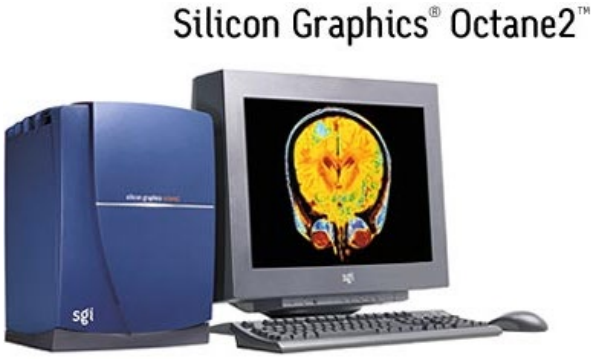
基于GPU的实时渲染技术

目录

- 图形硬件发展历史
- 图形绘制流水线
- 顶点着色器
- 几何着色器
- 片段着色器

图形硬件发展历史

- … - 90年代中期之前
  - SGI图形工作站
  - PC: 只支持2D图形
- 90年代中期
  - 消费级3D图形硬件 (PC)
    - 3dfx, nVidia, Matrox, ATI, …
  - 只支持三角形光栅化
  - 便宜：由游戏工业推动
- 1999年
  - 具有TnL（变换和光照）功能的PC显卡
    - nVIDIA GeForce: Graphics Processing Unit (GPU)
  - PC显卡比专用工作站更加强大



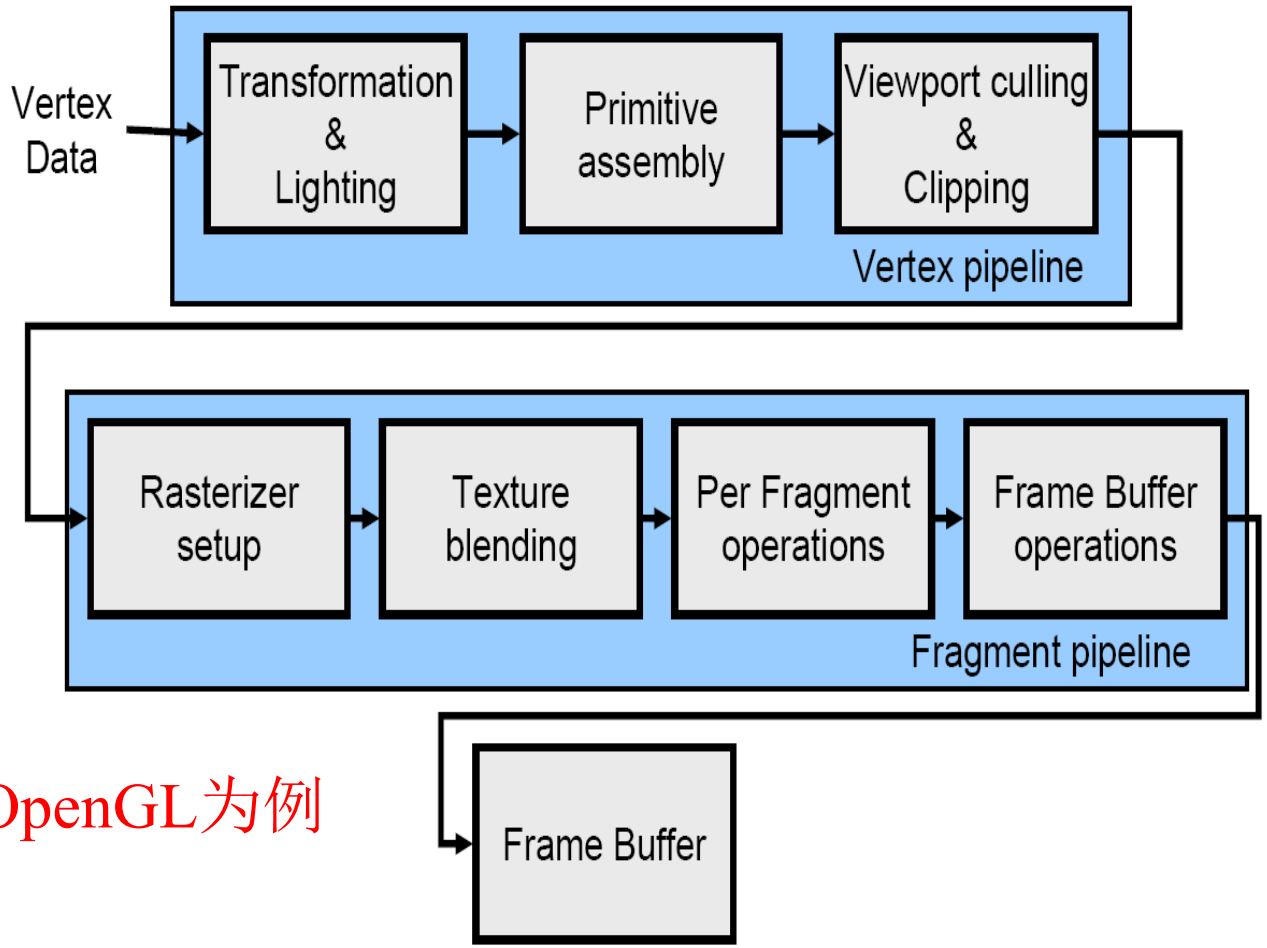
图形硬件发展历史

- 现代图形硬件
  - 图形绘制流水线部分可编程
  - 主要显卡厂商: ATI and nVidia
    - “ATI Radeon HD 5870” and “nVidia GeForce GTX 480”
    - “ATI Radeon HD 5970” and “nVidia GeForce GTX 580”
  - 游戏主机与GPU类似 (Xbox)

最新图形硬件

- NVIDIA GeForce RTX3090 (2020)
  - 10496 CUDA Cores
  - 24GB memory
  - ~3200$

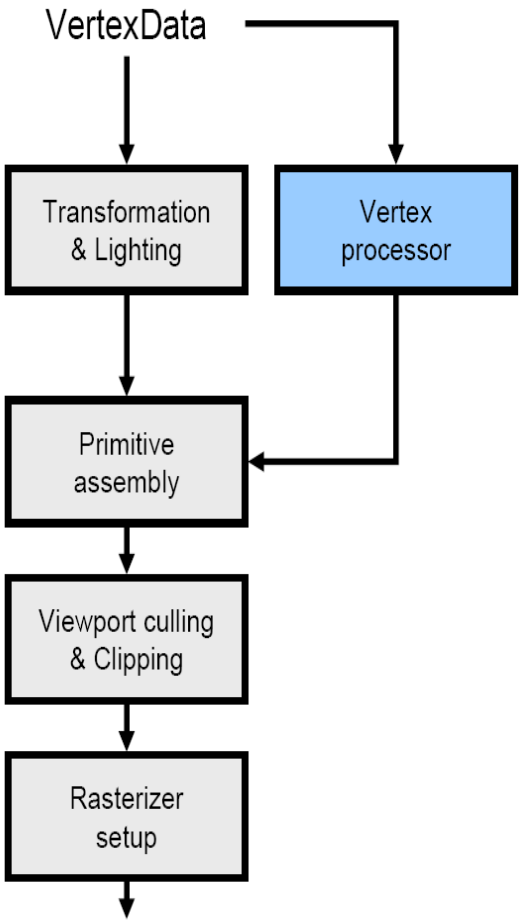
图形绘制流水线



以OpenGL为例

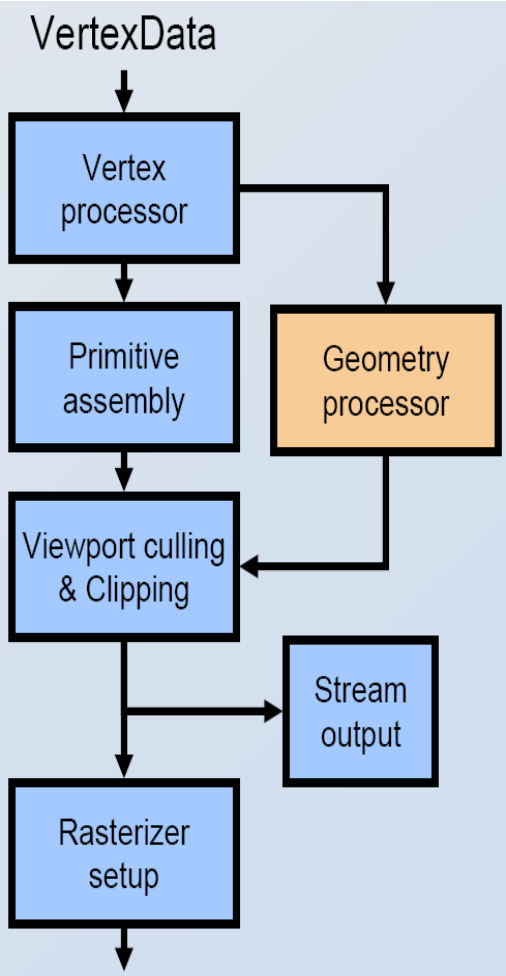
顶点着色器

- 顶点变换
- 法向量变换与单位化
- 纹理坐标生成与变换
- 逐顶点光照



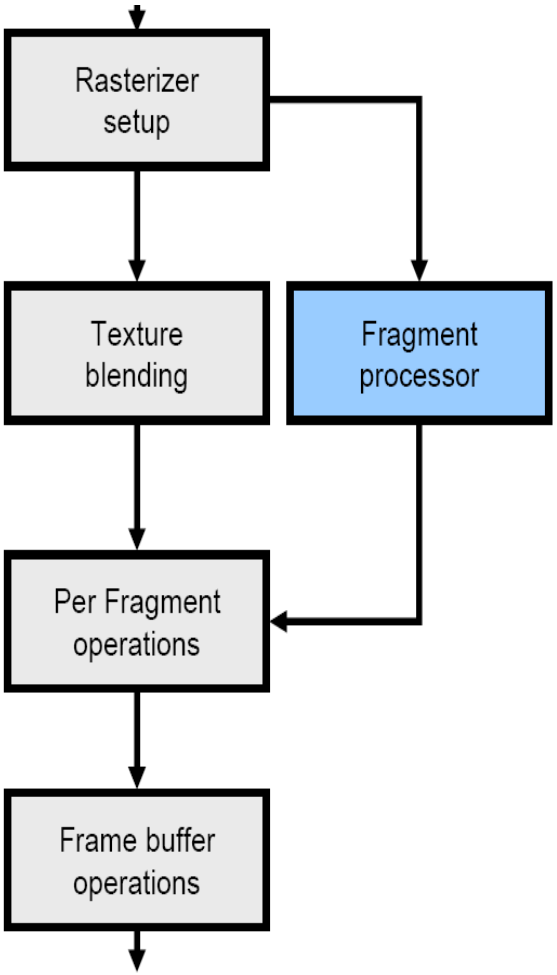
几何着色器

- 增加/删除图形元素
- 增加/删除顶点
- 编辑顶点位置
- 需要OpenGL Extension(Glew 1.4+) or DX10

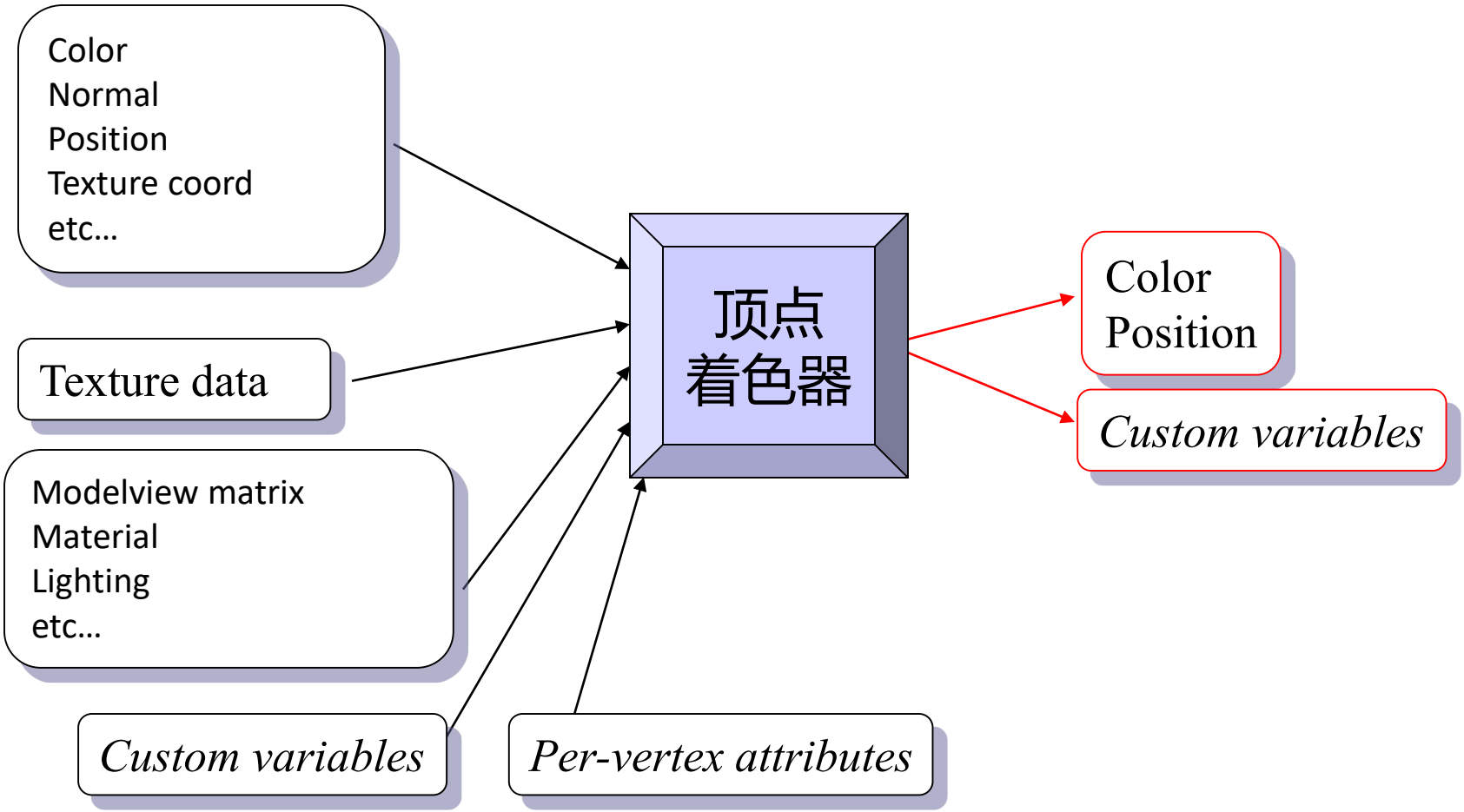


片段（像素）着色器

- 在插值后的属性值上进行计算
- 访问纹理
- 应用纹理
- 雾
- 颜色求和



顶点着色器的输入/输出


Color
Normal
Position
Texture coord
etc…
Texture data
Modelview matrix
Material
Lighting
etc…
Custom variables
Per-vertex attributes
顶点
着色器
Color
Position
Custom variables

关键帧动画

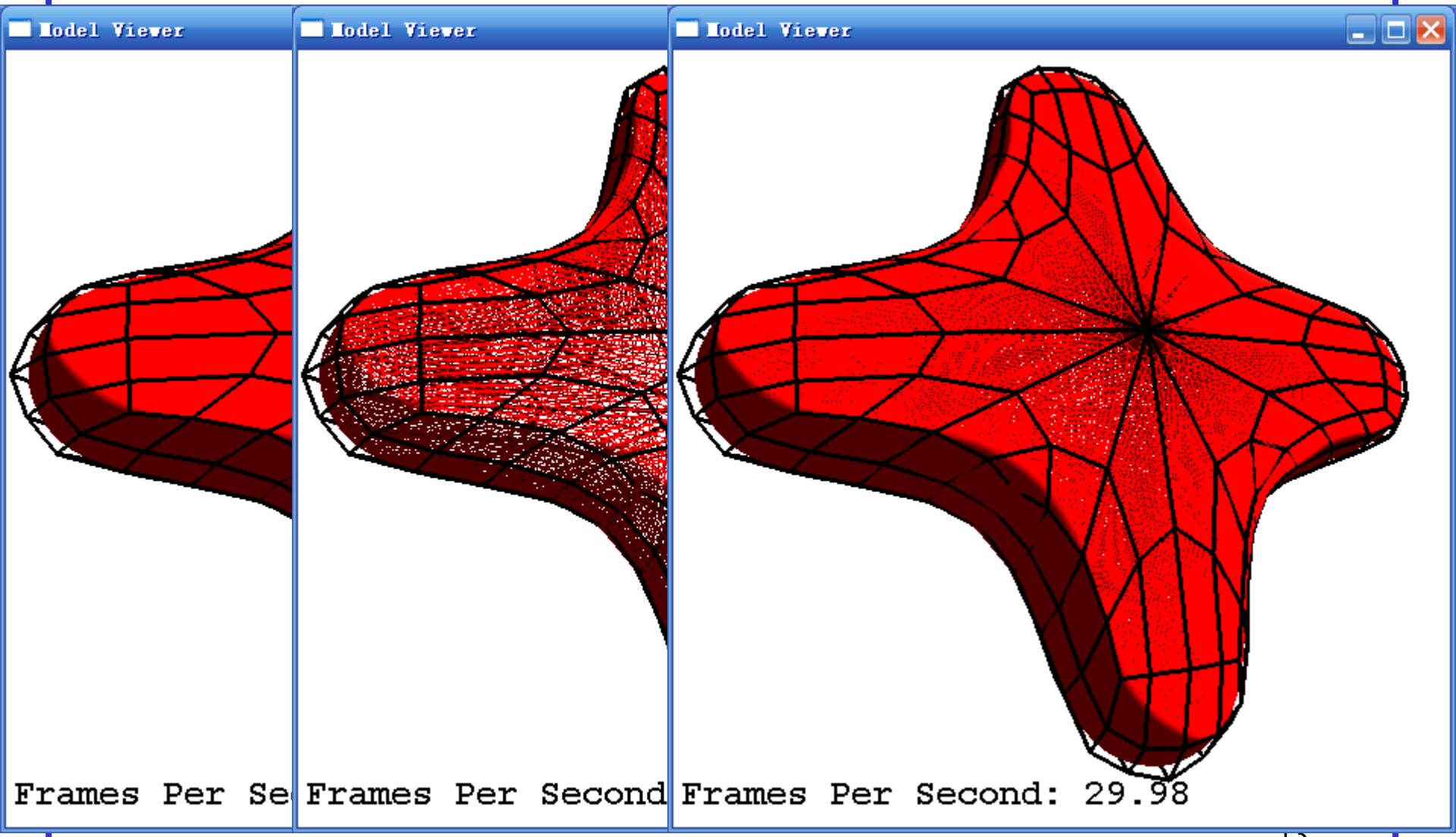

keyfra
keyframe_inter
keyframe_in
keyframe_inter
keyframe
keyframe_interpolation

关键帧动画

```
// vertex shader

uniform float    keyFrameBlend;

void main()
{
     vec4 position = mix(gl_Vertex, gl_MultiTexCoord1,
     keyFrameBlend);

     gl_Position = gl_ModelViewProjectionMatrix*position;
     gl_TexCoord[0] = gl_MultiTexCoord0;
     gl_FrontColor = gl_Color;
}
```

```
// fragment shader

uniform sampler2D data;
uniform vec2 scaleFactor;

void main()
{
     gl_FragColor = gl_Color*texture2D(data,
     gl_TexCoord[0].st*scaleFactor);
}
```

细分曲面


Model Viewer
Model Viewer
Model Viewer
Frames Per Se
Frames Per Second
Frames Per Second: 29.98

细分曲面

```
// vertex shader

struct  VertexInput
{
    float4 vPosition    : POSITION;
     float4 params0           : TEXCOORD0;
     float4 params1           : TEXCOORD1;
     float4 params2           : TEXCOORD2;
     float4 params3           : TEXCOORD3;
     float4 params4           : TEXCOORD4;
     float4 params5           : TEXCOORD5;
     float4 params6           : TEXCOORD6;
     float4 params7           : TEXCOORD7;
};

struct VertexToFragment
{
    float4 vPosition   : POSITION;
    float3 vTopColor   : COLOR;
     float3 vObjNorm    : TEXCOORD0;
     float3 vObjPos     : TEXCOORD1;
     float3 vObjParam   : TEXCOORD2;
     float4 vEyeDir     : TEXCOORD3;
};
```

细分曲面

```
void
main(in  VertexInput      oInput,
     out VertexToFragment oOutput,


     uniform float3       ctrls[32],
     uniform float3            norms[32],
     uniform int          valence,
     uniform float4       params,
     uniform float4x4     mModelView,
     uniform float4x4     mModelViewInverse)
{

    float3 dst =
           oInput.params0.x*ctrls[0] +
           oInput.params0.y*ctrls[1] +
           oInput.params0.z*ctrls[2] +
           oInput.params0.w*ctrls[3] +
           oInput.params1.x*ctrls[4] +
           oInput.params1.y*ctrls[5] +
           oInput.params1.z*ctrls[6] +
           oInput.params1.w*ctrls[7] +
           oInput.params2.x*ctrls[8] +
           oInput.params2.y*ctrls[9] +
           oInput.params2.z*ctrls[10] +
           oInput.params2.w*ctrls[11] +
           oInput.params3.x*ctrls[12] +
           oInput.params3.y*ctrls[13] +
           oInput.params3.z*ctrls[14] +
           oInput.params3.w*ctrls[15];
```

细分曲面

```
if (valence > 4)
 dst += oInput.params4.x*ctrls[16] + oInput.params4.y*ctrls[17];

if (valence > 5)
 dst += oInput.params4.z*ctrls[18] + oInput.params4.w*ctrls[19];

if (valence > 6)
 dst += oInput.params5.x*ctrls[20] + oInput.params5.y*ctrls[21];

if (valence > 7)
 dst += oInput.params5.z*ctrls[22] + oInput.params5.w*ctrls[23];

if (valence > 8)
 dst += oInput.params6.x*ctrls[24] + oInput.params6.y*ctrls[25];

if (valence > 9)
 dst += oInput.params6.z*ctrls[26] + oInput.params6.w*ctrls[27];

if (valence > 10)
 dst += oInput.params7.x*ctrls[28] + oInput.params7.y*ctrls[29];

if (valence > 11)
 dst += oInput.params7.z*ctrls[30] + oInput.params7.w*ctrls[31];
```

细分曲面

```
float3 nrm =
      oInput.params0.x*norms[0] +
      oInput.params0.y*norms[1] +
      oInput.params0.z*norms[2] +
      oInput.params0.w*norms[3] +
      oInput.params1.x*norms[4] +
      oInput.params1.y*norms[5] +
      oInput.params1.z*norms[6] +
      oInput.params1.w*norms[7] +
      oInput.params2.x*norms[8] +
      oInput.params2.y*norms[9] +
      oInput.params2.z*norms[10] +
      oInput.params2.w*norms[11] +
      oInput.params3.x*norms[12] +
      oInput.params3.y*norms[13] +
      oInput.params3.z*norms[14] +
      oInput.params3.w*norms[15];
```

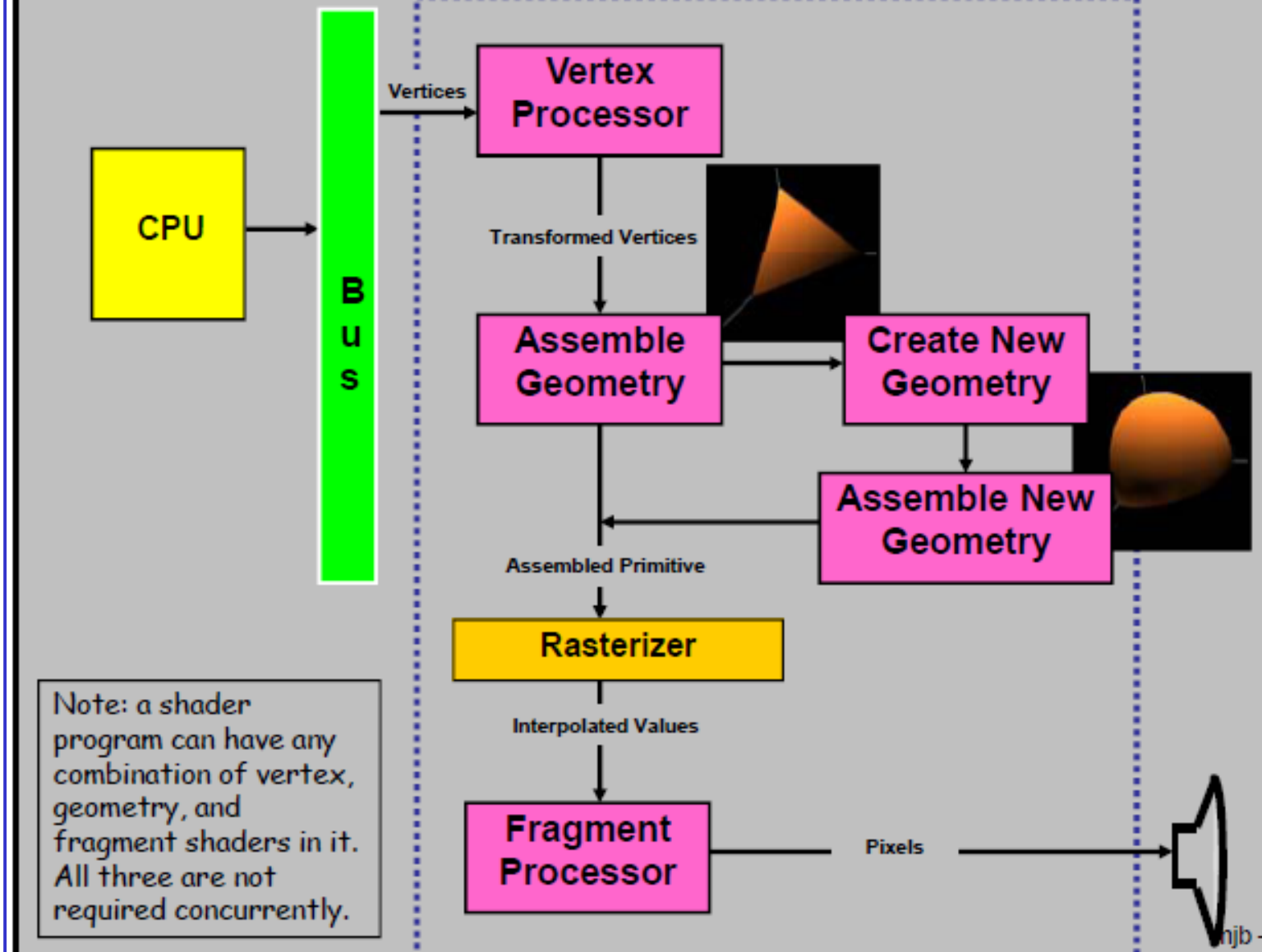
细分曲面

```
    if (valence > 4)
     nrm += oInput.params4.x*norms[16] + oInput.params4.y*norms[17];

   if (valence > 5)
     nrm += oInput.params4.z*norms[18] + oInput.params4.w*norms[19];

   if (valence > 6)
     nrm += oInput.params5.x*norms[20] + oInput.params5.y*norms[21];

   if (valence > 7)
     nrm += oInput.params5.z*norms[22] + oInput.params5.w*norms[23];

   if (valence > 8)
     nrm += oInput.params6.x*norms[24] + oInput.params6.y*norms[25];

   if (valence > 9)
     nrm += oInput.params6.z*norms[26] + oInput.params6.w*norms[27];

   if (valence > 10)
     nrm += oInput.params7.x*norms[28] + oInput.params7.y*norms[29];

   if (valence > 11)
     nrm += oInput.params7.z*norms[30] + oInput.params7.w*norms[31];
```

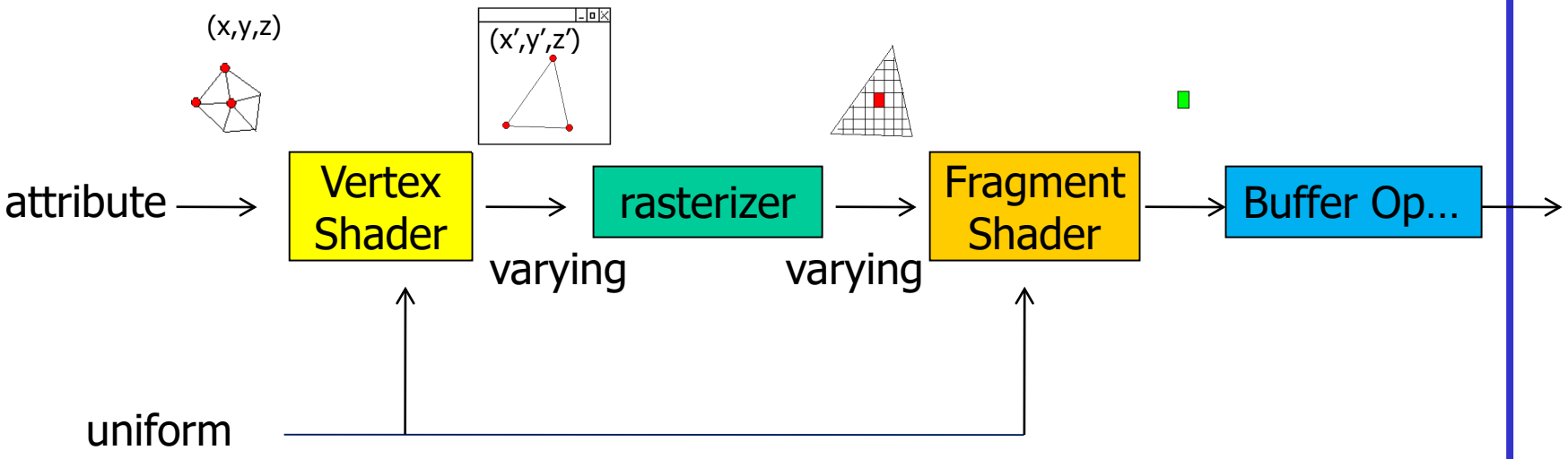
细分曲面

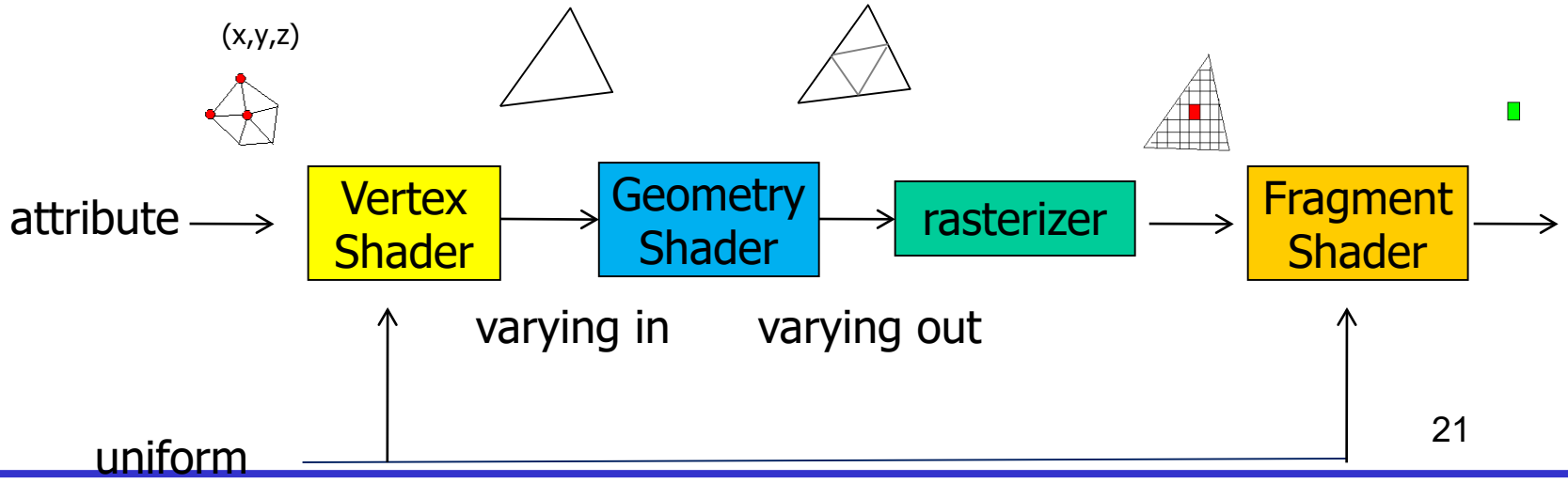
```
    oOutput.vTopColor = normalize(nrm);

    oOutput.vObjNorm = oOutput.vTopColor;
    oOutput.vObjPos = dst;
    oOutput.vObjParam = oInput.vPosition.xyz;

    oOutput.vPosition = mul(mModelView, float4(dst, 1));

    float4 eyePos_obj = mul(mModelViewInverse, float4(0.0, 0.0, 0.0, 1.0));
   float3 V = dst - eyePos_obj.xyz;

   oOutput.vEyeDir = float4(V, 0.0);
}
```

The Geometry Shader: Where Does it Fit in the Pipeline?

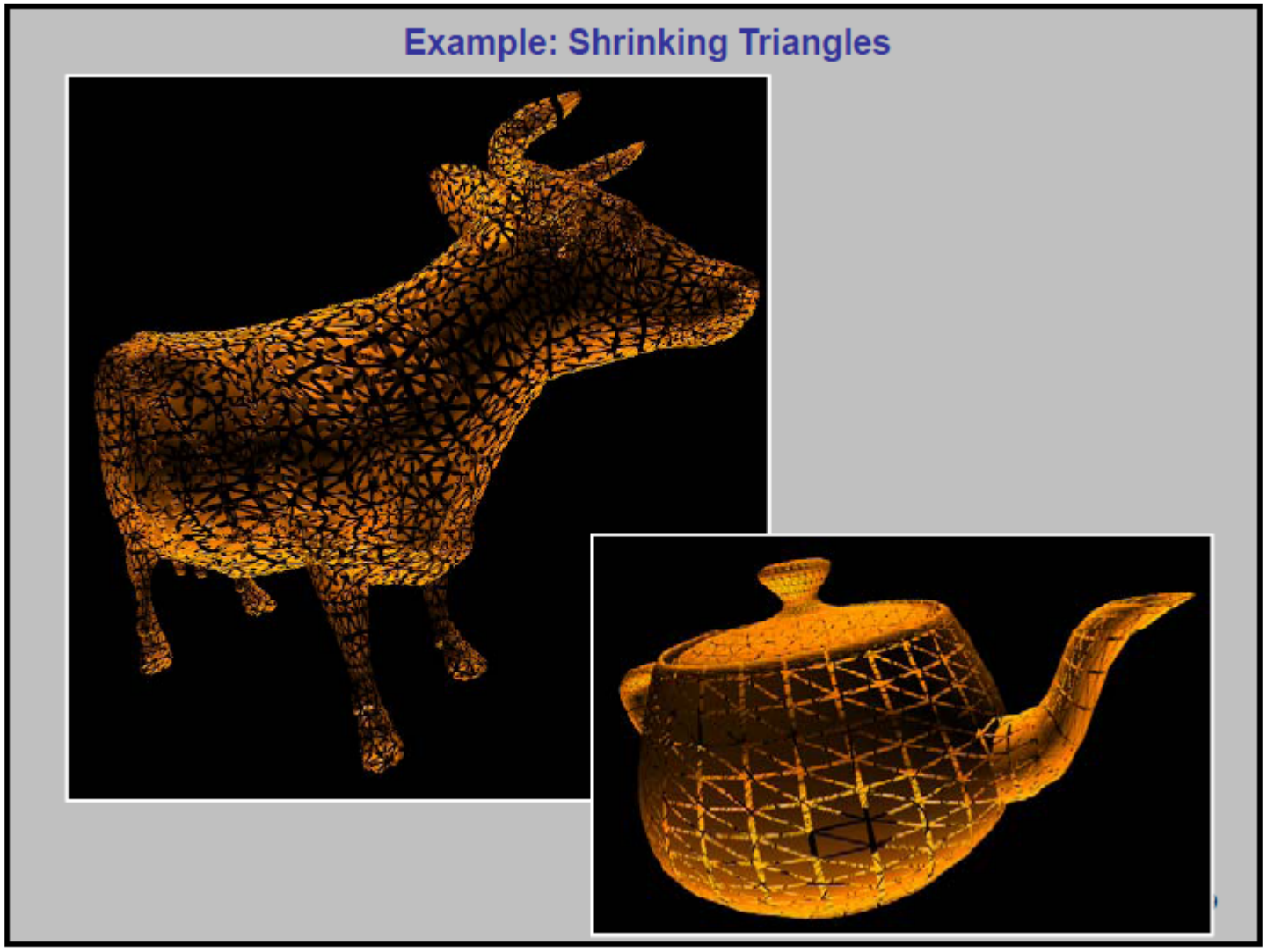


几何着色器


(x,y,z)
(x′,y′,z′)
attribute
Vertex Shader
rasterizer
Fragment Shader
Buffer Op…
varying
varying
uniform


(x,y,z)
attribute
Vertex Shader
Geometry Shader
rasterizer
Fragment Shader
varying in
varying out
uniform

模型镂空

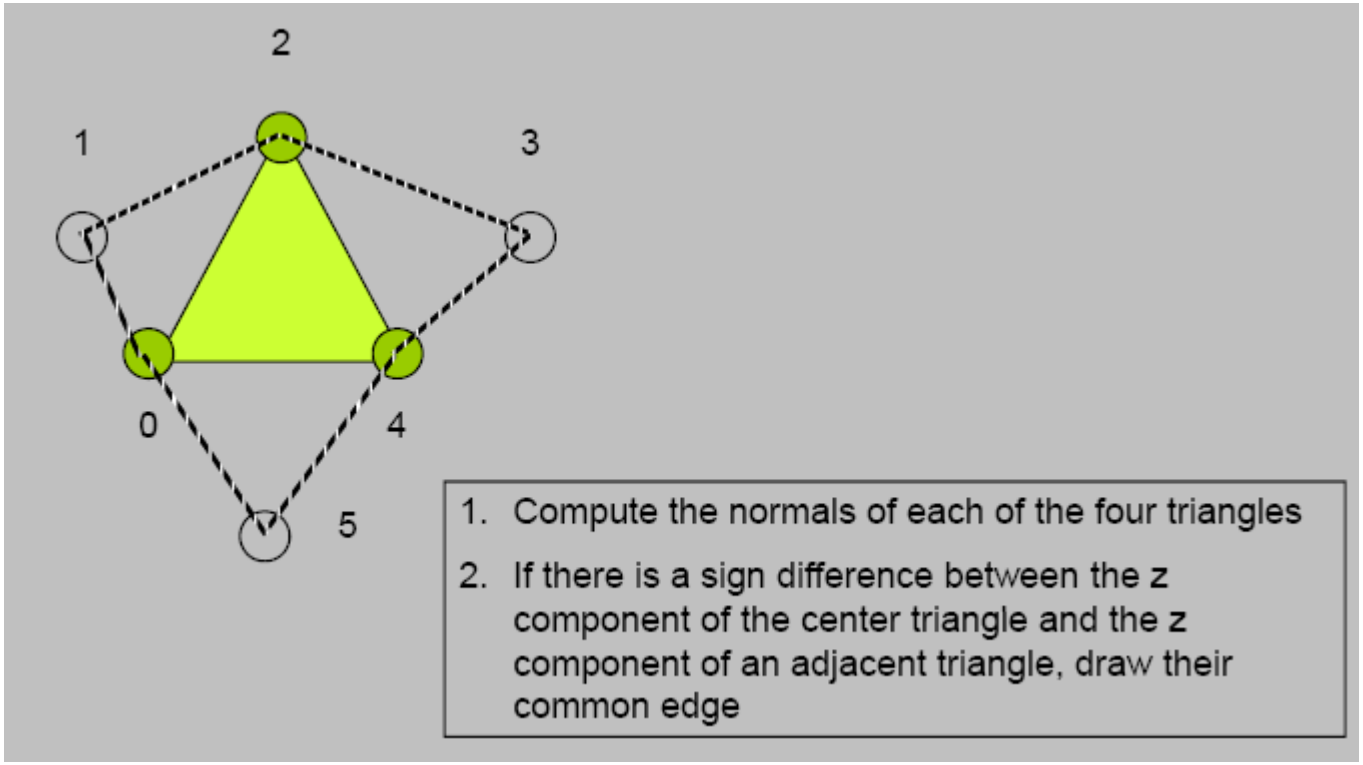


模型镂空

```
#version 120
#extension GL_EXT_geometry_shader4: enable

uniform float Shrink;
varying in vec3 Normal[3];
varying out float LightIntensity;
const vec3 LightPos = vec3( 0., 10., 0. );
vec3 V[3];
vec3 CG;

void
ProduceVertex( int v )
{
	LightIntensity  = dot( normalize(LightPos - V[v]), Normal[v] );
	LightIntensity = abs( LightIntensity );
	LightIntensity *= 1.5;

	gl_Position = gl_ModelViewProjectionMatrix * vec4( CG + Shrink * ( V[v] - CG ), 1. );
	EmitVertex();
}

void
main()
{
	V[0]  =  gl_PositionIn[0].xyz;
	V[1]  =  gl_PositionIn[1].xyz;
	V[2]  =  gl_PositionIn[2].xyz;
	CG = (  V[0] + V[1] + V[2]  ) / 3.;
	ProduceVertex( 0 );
	ProduceVertex( 1 );
	ProduceVertex( 2 );
}
```

轮廓线提取

轮廓线



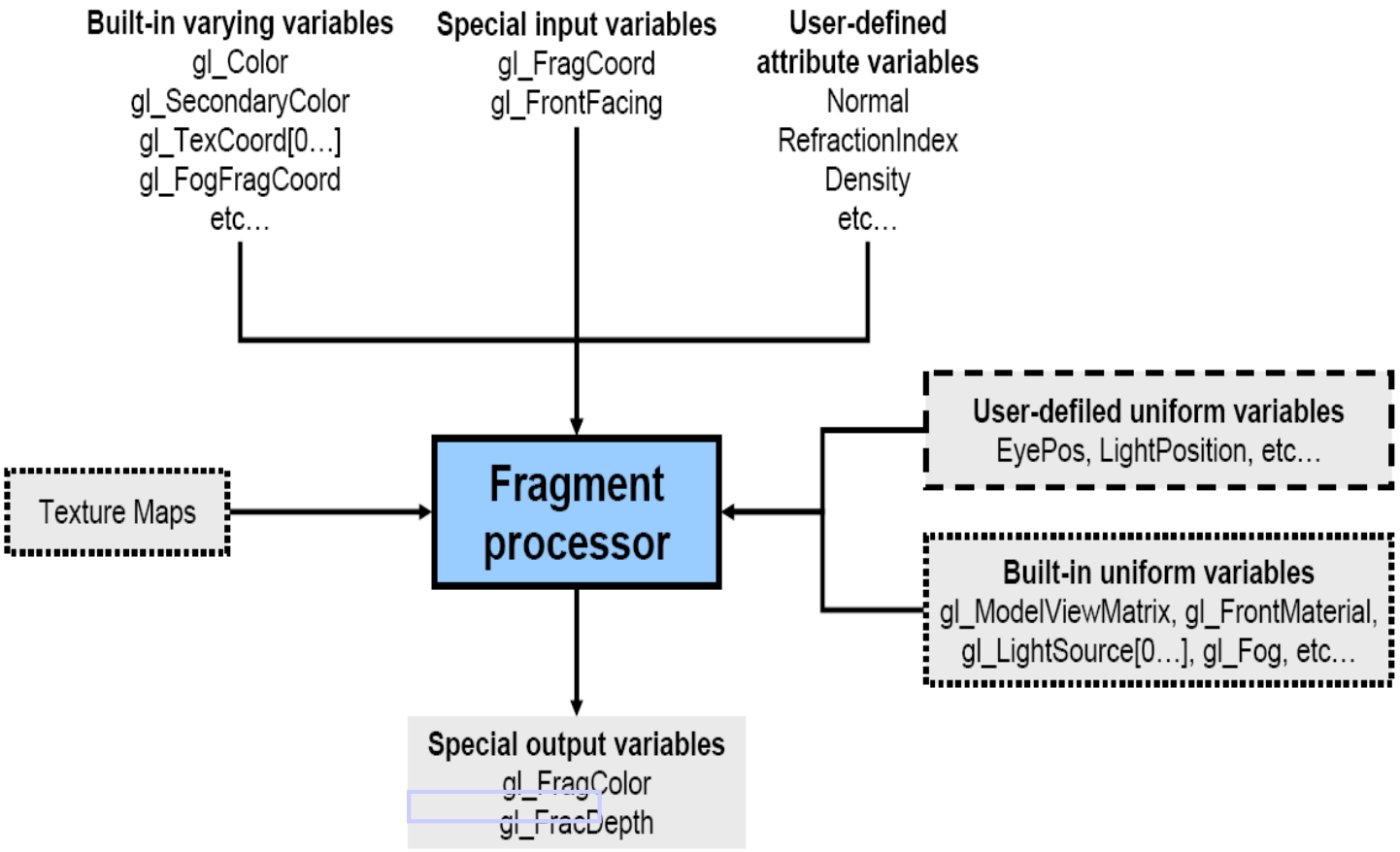
Input: gl_triangle_adjacency
Output: gl_line_strip

轮廓线

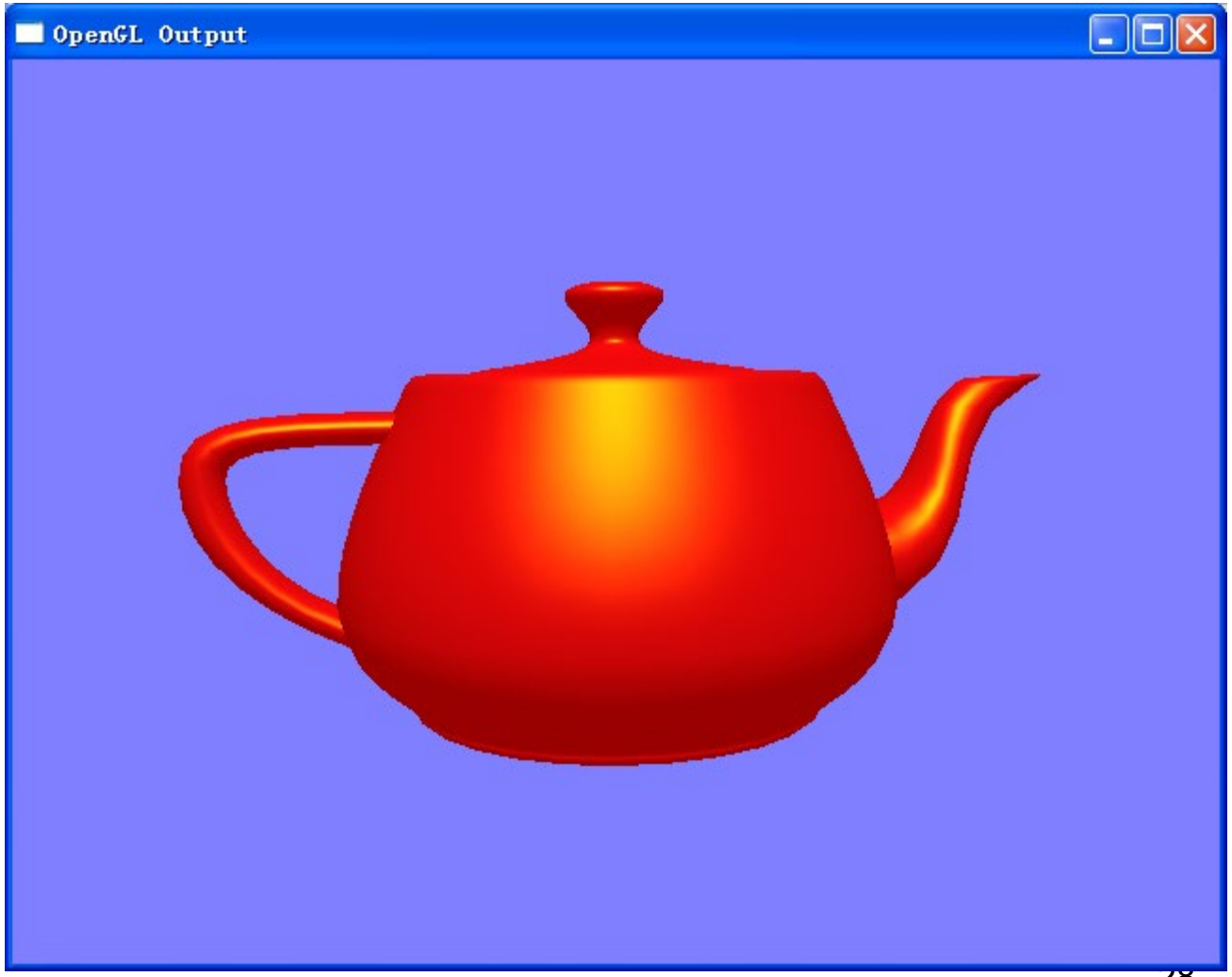
```
#version 120
#extension GL_EXT_geometry_shader4: enable

void
main()
{
        vec3 V0 = gl_PositionIn[0].xyz;
        vec3 V1 = gl_PositionIn[1].xyz;
        vec3 V2 = gl_PositionIn[2].xyz;
        vec3 V3 = gl_PositionIn[3].xyz;
        vec3 V4 = gl_PositionIn[4].xyz;
        vec3 V5 = gl_PositionIn[5].xyz;

        vec3 N042 = cross( V4-V0, V2-V0 );
        vec3 N021 = cross( V2-V0, V1-V0 );
        vec3 N243 = cross( V4-V2, V3-V2 );
        vec3 N405 = cross( V0-V4, V5-V4 );

        if( dot( N042, N021 ) < 0. )
                N021 = vec3(0.,0.,0.) - N021;

        if( dot( N042, N243 ) < 0. )
                N243 = vec3(0.,0.,0.) - N243;

        if( dot( N042, N405 ) < 0. )
                N405 = vec3(0.,0.,0.) - N405;
```

```
if( N042.z * N021.z < 0. )
{
        gl_Position = gl_ProjectionMatrix * vec4( V0, 1. );
        EmitVertex();
        gl_Position = gl_ProjectionMatrix * vec4( V2, 1. );
        EmitVertex();
        EndPrimitive();
}

if( N042.z * N243.z < 0. )
{
        gl_Position = gl_ProjectionMatrix * vec4( V2, 1. );
        EmitVertex();
        gl_Position = gl_ProjectionMatrix * vec4( V4, 1. );
        EmitVertex();
        EndPrimitive();
}

if( N042.z * N405.z < 0. )
{
        gl_Position = gl_ProjectionMatrix * vec4( V4, 1. );
        EmitVertex();
        gl_Position = gl_ProjectionMatrix * vec4( V0, 1. );
        EmitVertex();
        EndPrimitive();
}
```

片段着色器输入/输出


Built-in varying variables
gl_Color
gl_SecondaryColor
gl_TexCoord[0…]
gl_FogFragCoord
etc…
Special input variables
gl_FragCoord
gl_FrontFacing
User-defined attribute variables
Normal
RefractionIndex
Density
etc…
User-defiled uniform variables
EyePos, LightPosition, etc…
Texture Maps
Fragment processor
Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial,
gl_LightSource[0…], gl_Fog, etc…
Special output variables
gl_FragColor
gl_FragDepth

逐片段光照



逐片段光照

- 顶点着色器

```
varying vec3 N;
varying vec3 v;

void main(void)
{

   v = vec3(gl_ModelViewMatrix * gl_Vertex);
   N = normalize(gl_NormalMatrix * gl_Normal);

   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;


}
```

逐片段光照

- 片段着色器

```
varying vec3 N;
varying vec3 v;

void main (void)
{
   vec3 L = normalize(gl_LightSource[0].position.xyz - v);
   vec3 E = normalize(-v); // we are in Eye Coordinates, so EyePos is (0,0,0)
   vec3 R = normalize(-reflect(L,N));

   //calculate Ambient Term:
   vec4 Iamb = gl_FrontLightProduct[0].ambient;

   //calculate Diffuse Term:
   vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

   // calculate Specular Term:
   vec4 Ispec = gl_FrontLightProduct[0].specular
                * pow(max(dot(R,E),0.0),0.3*gl_FrontMaterial.shininess);

   // write Total Color:
   gl_FragColor = gl_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;
}
```

THANK YOU