Geometry Shaders

Ludovic Favre & Wadeck Follonier

Overview

1. Purpose and background
2. What's new ?
3. Usage
4. Demos
5. Conclusion: pros and cons

Geometry shader purpose

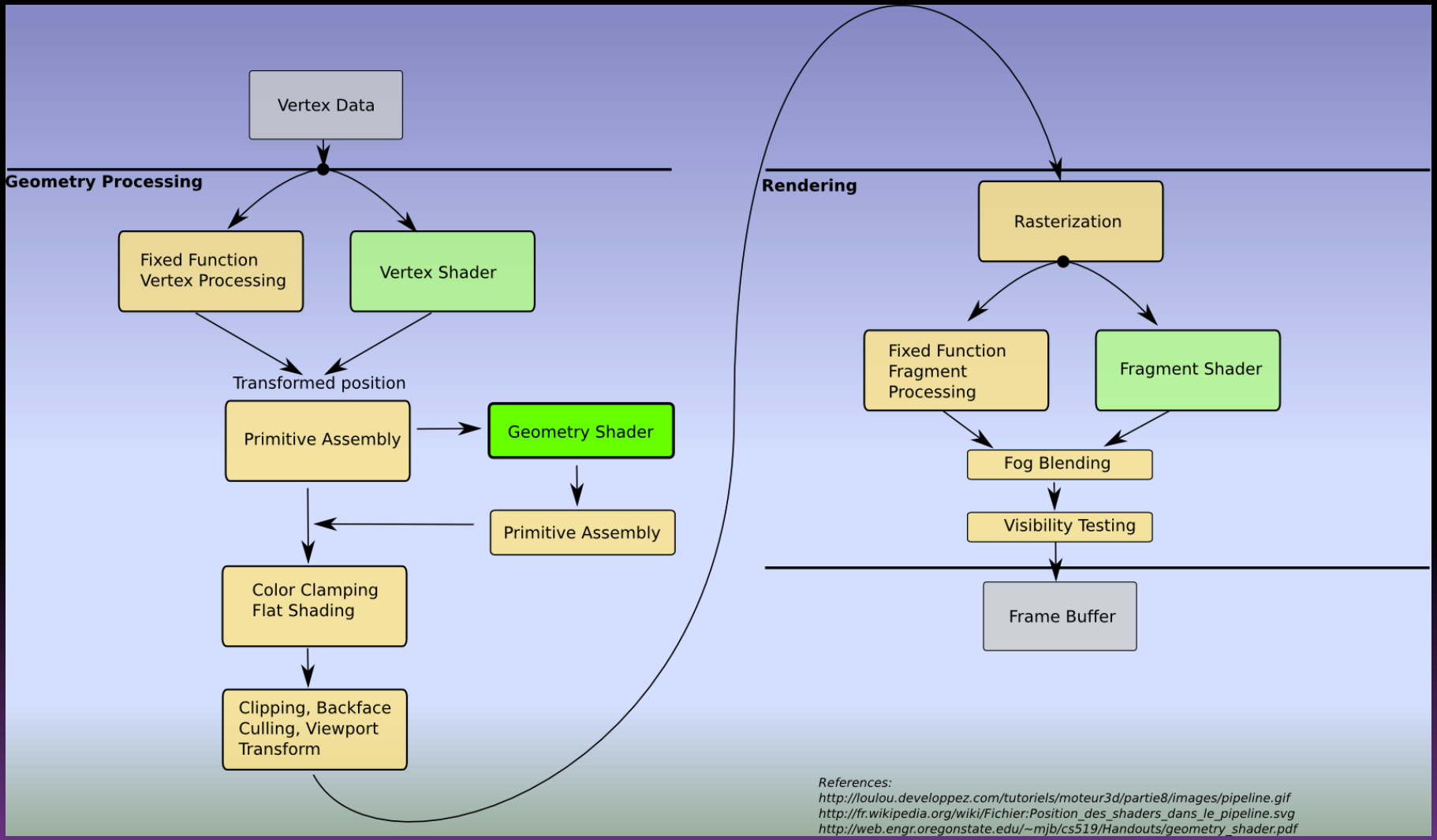
- In a few words :
  - you can add / remove / modify geometry primitives

Some history

- Requires OpenGL 3.2, *August 2009*
  - (or OpenGL 1.1 using the EXT_geometry_shader4 extension, since 2006)
- In DirectX world : since Shader Model 4, DirectX 10, *November, 2006*
  - For NVIDIA : Geforce 8
  - For ATI : Radeon HD series

http://en.wikipedia.org/wiki/DirectX
http://en.wikipedia.org/wiki/OpenGL

In the OpenGL Pipeline



References:
http://loulou.developpez.com/tutoriels/moteur3d/partie8/images/pipeline.gif
http://fr.wikipedia.org/wiki/Fichier:Position_des_shaders_dans_le_pipeline.svg
http://web.engr.oregonstate.edu/~mjb/cs519/Handouts/geometry_shader.pdf

New instructions available

- EmitVertex() : create a new vertex
  - *send the vertex you have been developing on to the second primitive assembly step.*
- EndPrimitive() : we are done with a primitive
  - *take all the vertices that have been sent to primitive assembly and create a geometry primitive to send on for further processing.*

New instructions available

- You need to put these instructions at the beginning of your geometry shader code

```
#version 120
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
```

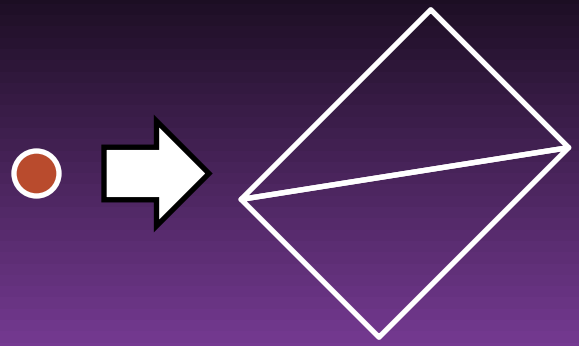
Primitives types

- Input Primitives
  - GL_POINTS
  - GL_LINES
  - GL_TRIANGLES

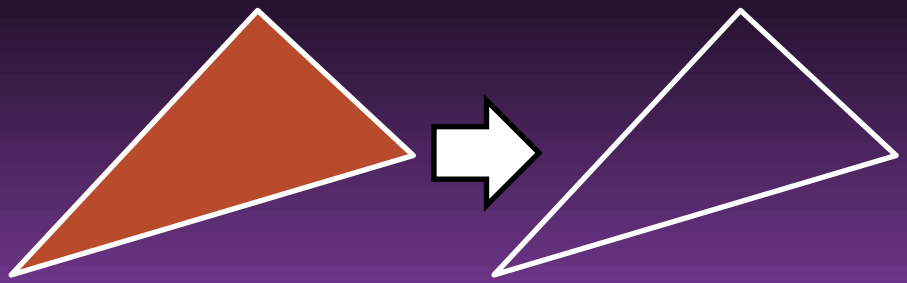
Output Primitives

- GL_POINTS
- GL_LINE_STRIP
- GL_TRIANGLE_STRIP

Points => Triangle Strips



Triangles => Line Strips



How to use it ?

```
void Shader::init(string vertexFile, string fragmentFile, string geometryFile, GLushort geomInput, GLushort geomOutput) {
    m_name = vertexFile;
    m_name.resize(m_name.size()-5);
    m_id = glCreateProgram();
    m_vertexId = glCreateShader(GL_VERTEX_SHADER);
    m_fragmentId = glCreateShader(GL_FRAGMENT_SHADER);
    m_geometryId = glCreateShader(GL_GEOMETRY_SHADER_EXT);

    const char* vertexBuffer = m_readTextFile(vertexFile);
    const char* fragmentBuffer = m_readTextFile(fragmentFile);
    const char* geometryBuffer = m_readTextFile(geometryFile);

    if (vertexBuffer == NULL || fragmentBuffer == NULL || geometryBuffer == NULL) {
        fprintf(stderr,"Critical: Cannot open one of the shaders !\n");
    }

    glShaderSource(m_vertexId, 1, &vertexBuffer, NULL);
    glShaderSource(m_fragmentId, 1, &fragmentBuffer, NULL);
    glShaderSource(m_geometryId, 1, &geometryBuffer,NULL);

    delete vertexBuffer;
    delete fragmentBuffer;
    delete geometryBuffer;

    glCompileShader(m_vertexId);
    glCompileShader(m_fragmentId);
    glCompileShader(m_geometryId);


    glProgramParameteriEXT(m_id,GL_GEOMETRY_INPUT_TYPE_EXT , geomInput );
    glProgramParameteriEXT(m_id,GL_GEOMETRY_OUTPUT_TYPE_EXT , geomOutput);
    int temp;
    glGetIntegerv(GL_MAX_GEOMETRY_OUTPUT_VERTICES_EXT,&temp);
    fprintf(stderr,"Supported vertices output : %d\n",temp);
    glProgramParameteriEXT(m_id,GL_GEOMETRY_VERTICES_OUT_EXT, temp);
    glAttachShader(m_id, m_vertexId);
    glAttachShader(m_id, m_fragmentId);
    glAttachShader(m_id, m_geometryId);

    glLinkProgram(m_id);

    m_printProgramInfoLog();
}
```

Type of shader to be created

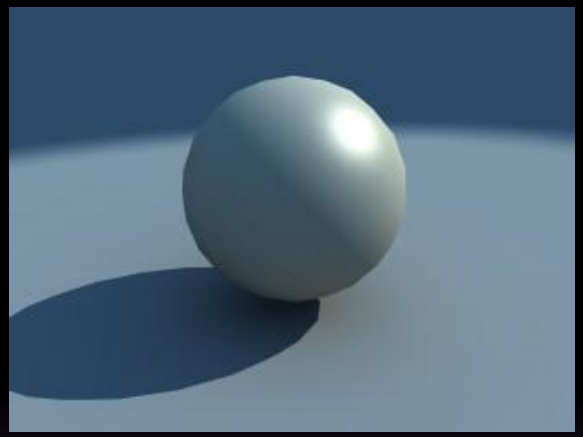
Primitive type that this Geometry Shader will be receiving and emitting

Maximum number of vertices this Geometry Shader will be emitting

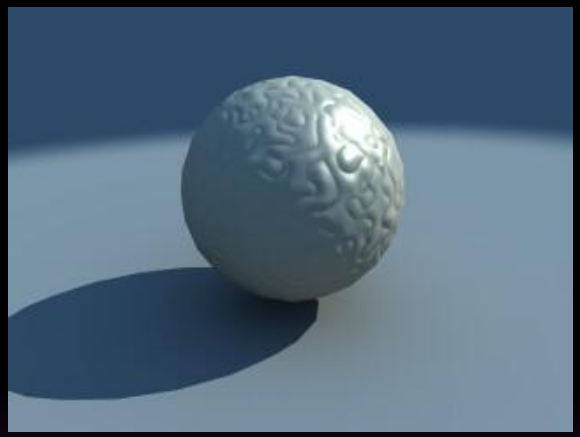
Applications

- Displacement mapping

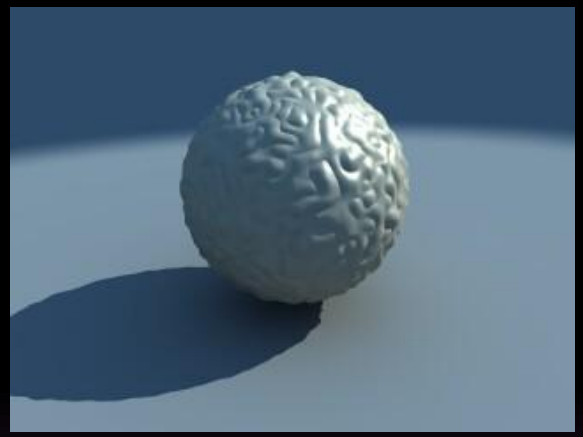
Geometry Shader



Original object



Bump mapping

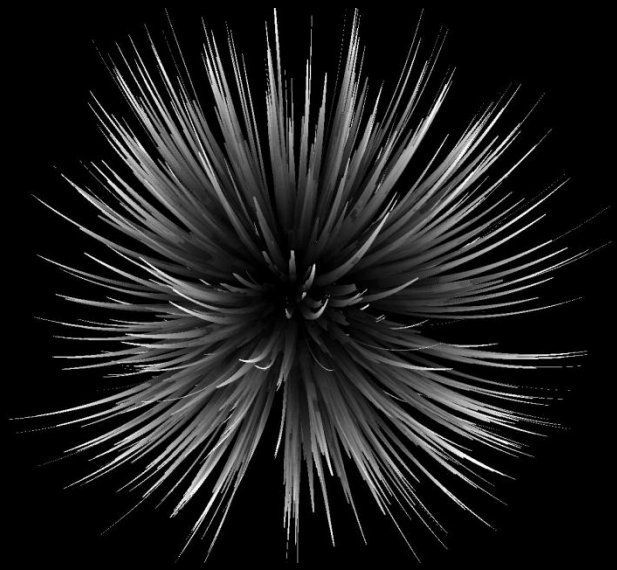


Displacement mapping

http://www.spot3d.com/vray/help/150SP1/tutorials_displacement.htm

Applications

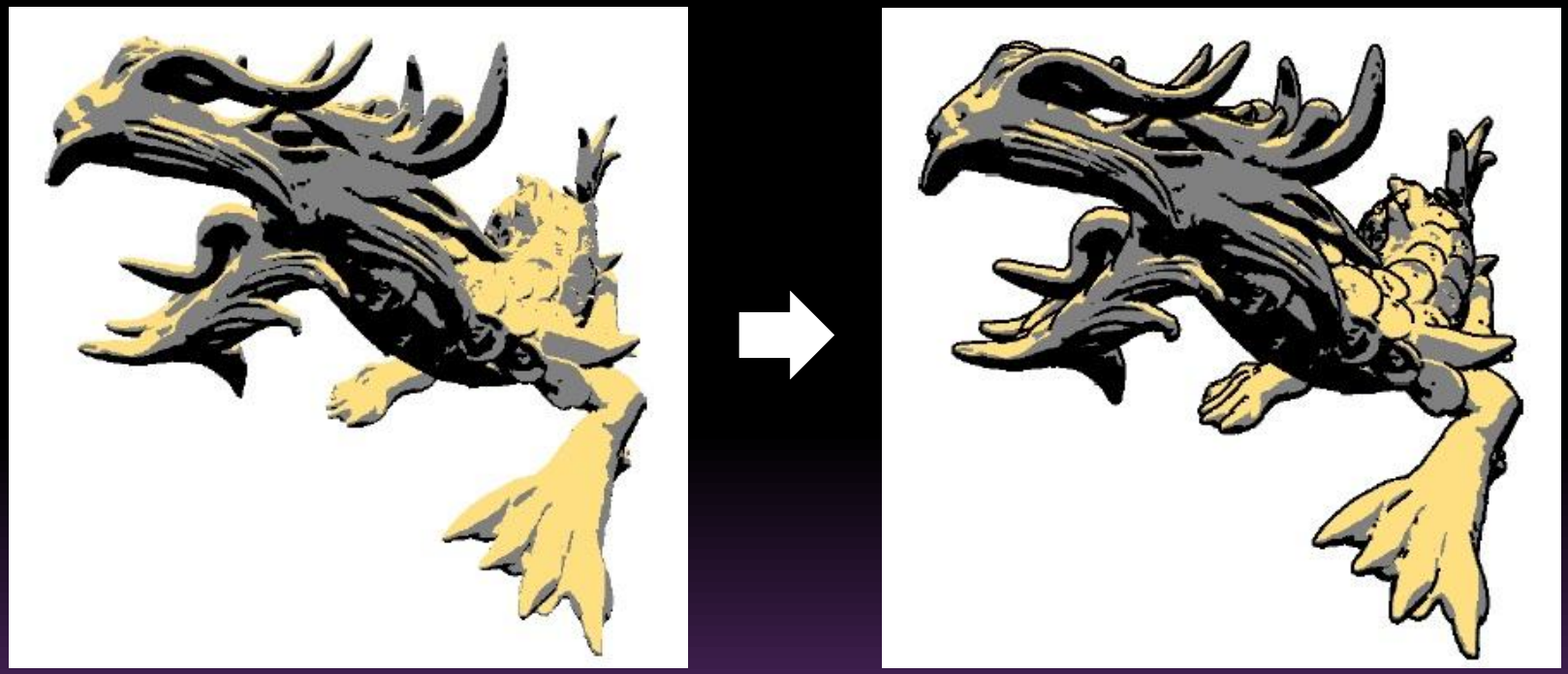
- Fur rendering



http://www.creativecrash.com/maya/downloads/scripts-plugins/rendering/c/shaders_p--2
http://www.sgtconker.com/2009/10/article-fur-rendering/
http://amiri-gualtiero.freehomeblogs.in/10051506/geometry-shader/

Applications

- Silhouettes for refinement



Toon shaded Asian Dragon model without and with silhouettes

Single Pass GPU Stylized Edges, *P. Hermosilla & P.P. Vázquez*

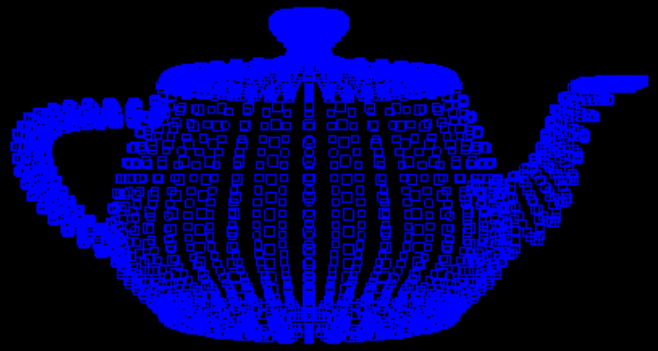
Applications

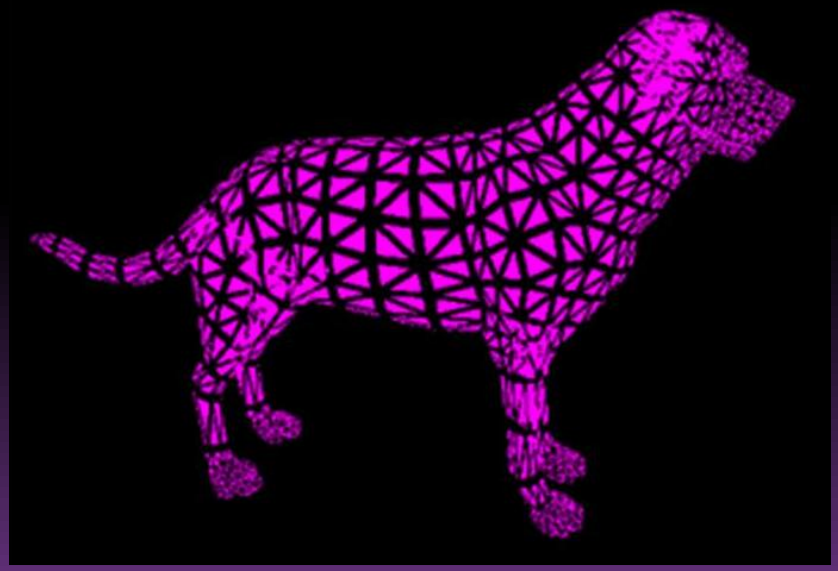
- IFS simulation

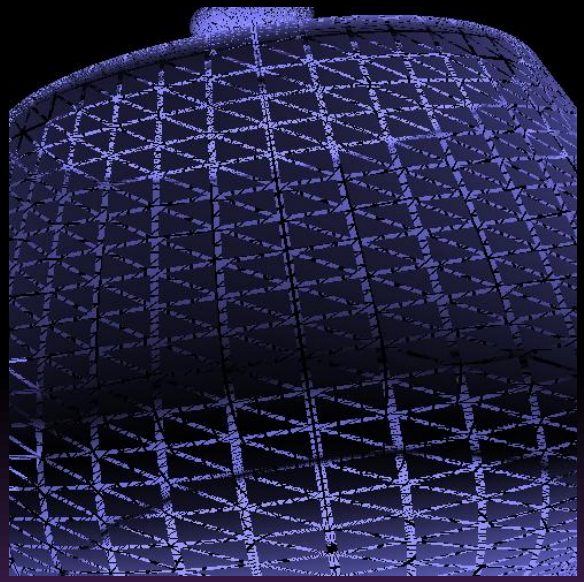
http://msdn.microsoft.com/en-us/library/ee416554(VS.85).aspx

Applications

- Triangles shrinking







http://www.derektanderson.com/Intersession2008ShaderProgramming/index.html
http://www.evl.uic.edu/aej/594/lecture01.html

Applications

- Tessellation

(a) An untessellated character

(b) Character rendered with GPU tessellation

(c) An untessellated character close-up

(d) Tessellated character close-up

Programming for Real-Time Tessellation on GPU, *Natalya Tatarchuk Joshua Barczak Bill Bilodeau*

A simple geometry shader

Keep the same

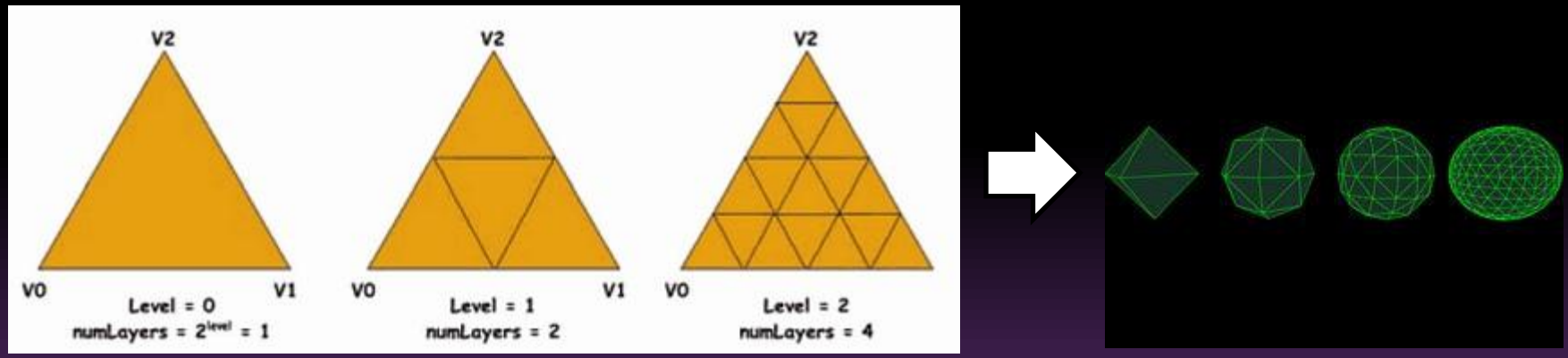
Inverse x/y

Inverse y/z

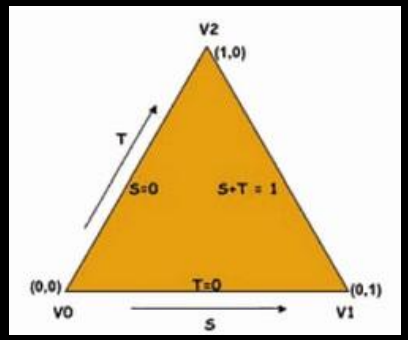
```
#version 120
#extension GL_EXT_geometry_shader4 : enable

// A simple 3D coordinate inversion for lines.
void main(void)
{

    //increment variable
    int i;

    //purple
    gl_FrontColor = vec4(1.0,0.0,1.0,1.0);
    //Pass-through
    for(i=0; i< gl_VerticesIn; i++){
        gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_PositionIn[i];
        EmitVertex();
    }
    EndPrimitive();

    //yellow
    gl_FrontColor = vec4(1.0,1.0,0.0,1.0);
    //invert x and y
    for(i=0; i< gl_VerticesIn; i++){
        gl_Position = gl_PositionIn[i];
        gl_Position.xyz = gl_Position.yxz;
        gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Position;
        EmitVertex();
    }
    EndPrimitive();

    //light-blue
    gl_FrontColor = vec4(0.0,1.0,1.0,1.0);
    //invert y and z
    for(i=0; i< gl_VerticesIn; i++){
        gl_Position = gl_PositionIn[i];
        gl_Position.xyz = gl_Position.xzy;
        gl_Position = gl_ProjectionMatrix*gl_ModelViewMatrix*gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```

Our geometry shader

- LoD on a sphere
  - Use few vertices for the far triangles
  - Add vertices when the triangles are near the viewer



Our geometry shader



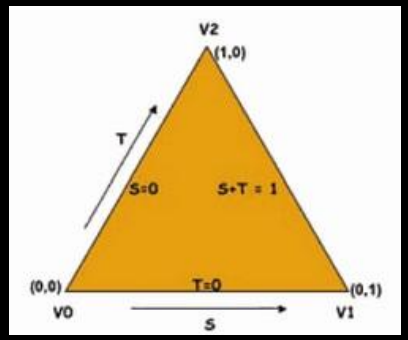
- Top of the shader

Version, extensions

Variables

```
#version 120
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable

// Light intensity factor, passed to fragment shader
varying float LightIntensity;

// Vectors required for computation on the triangle
vec3 V0, V01, V02;
```

Our geometry shader



- Main function

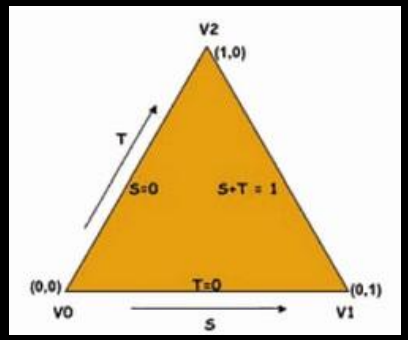
Compute the distance then the required level of detail

Compute the position vectors

Determine the length of the segments

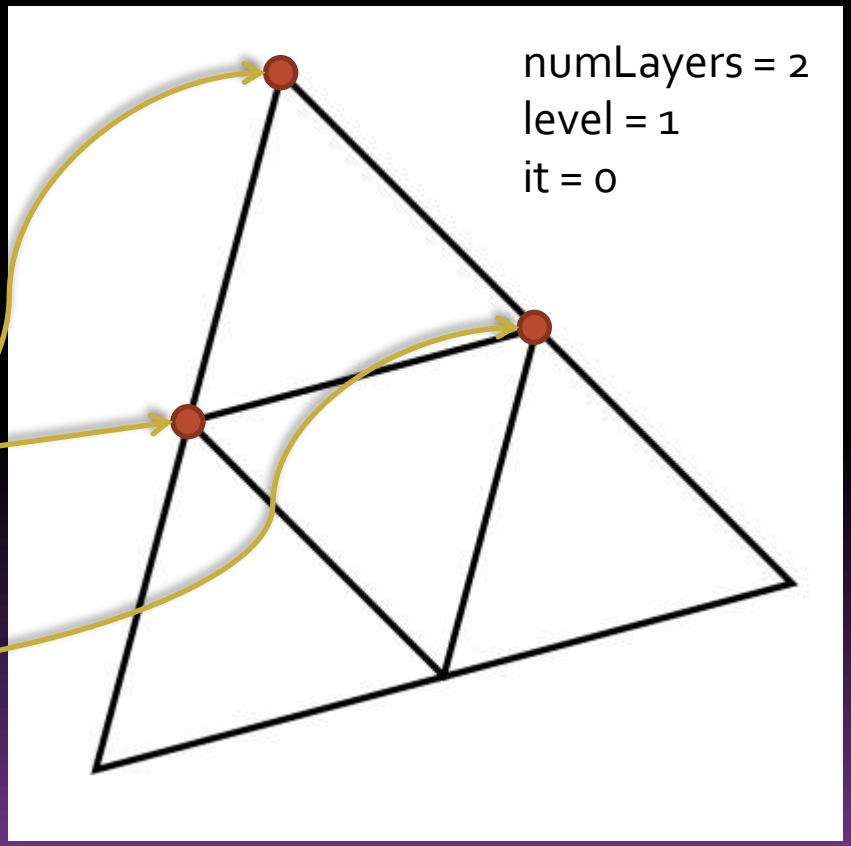
```
void main(){
    // Init to higher Level of Detail
    int level = 3;

    // Distance from triangle to viewer, in xyz space
    float dist = distance((gl_PositionIn[0]+gl_PositionIn[1]+gl_PositionIn[2])/3,
       gl_ModelViewMatrixInverse[3]);
    // Level of Detail selection
    if(dist > 25.){
      level = 0;
    } else if(dist > 15.){
      level = 1;
    } else if(dist > 5.){
      level = 2;
    }

    // The 3 vectors in XYZ space (not changed in vertex shader)
    V01 = ( gl_PositionIn[1] - gl_PositionIn[0] ).xyz;
    V02 = ( gl_PositionIn[2] - gl_PositionIn[0] ).xyz;
    V0  =   gl_PositionIn[0].xyz;

    // Number of subdivisions
    int numLayers = 1 << level;

    // Slice size
    float dt = 1. / float( numLayers );
```

Our geometry shader

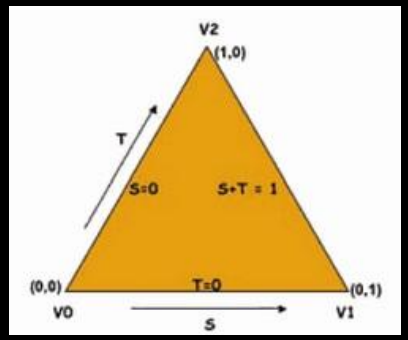


- Main function cont'd

```
    float t_top = 1.;

    for( int it = 0; it < numLayers; it++ ) {
        float t_bot = t_top - dt;
        float smax_top = 1. - t_top;
        float smax_bot = 1. - t_bot;

        int nums = it + 1;
        float ds_top = smax_top / float( nums - 1 );
        float ds_bot = smax_bot / float( nums );

        float s_top = 0.;
        float s_bot = 0.;

        for( int is = 0; is < nums; is++ ){
            ProduceVertex( s_bot, t_bot );
            ProduceVertex( s_top, t_top );
            s_top += ds_top;
            s_bot += ds_bot;
        }

        ProduceVertex( s_bot, t_bot );
        EndPrimitive();

        t_top = t_bot;
        t_bot -= dt;
    }
}
```

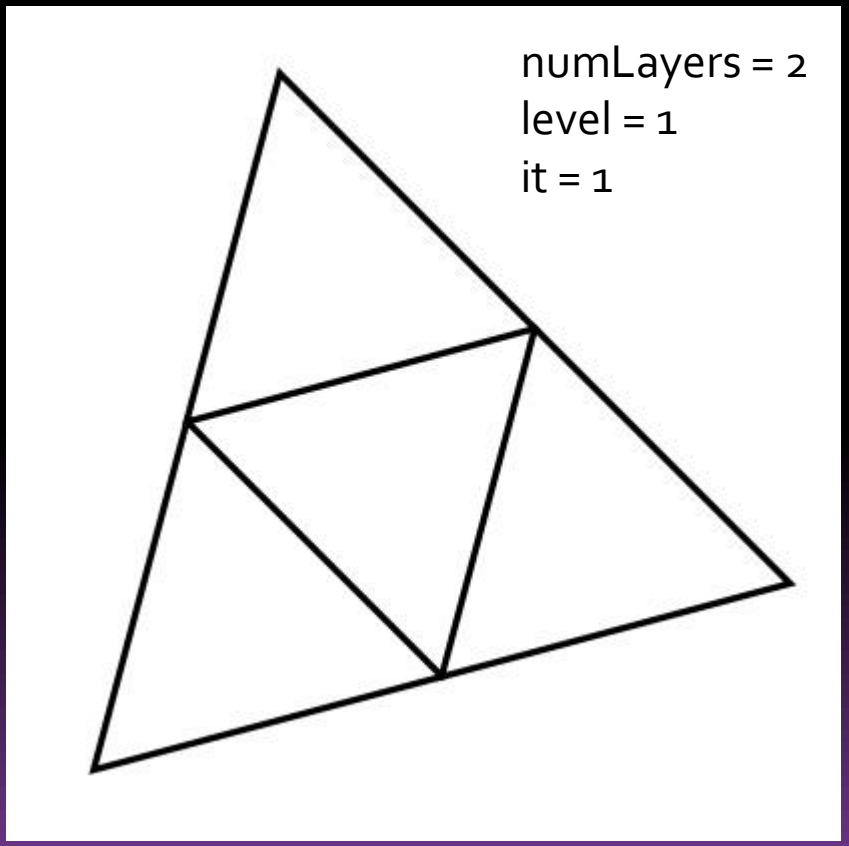


Our geometry shader

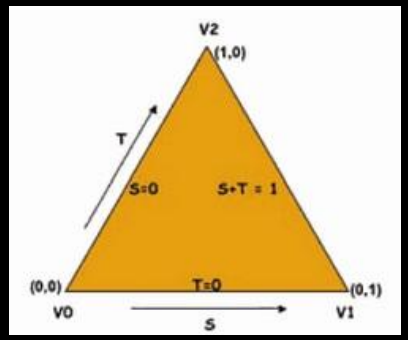


- Main function cont'd

```
    float t_top = 1.;

    for( int it = 0; it < numLayers; it++ ) {
        float t_bot = t_top - dt;
        float smax_top = 1. - t_top;
        float smax_bot = 1. - t_bot;

        int nums = it + 1;
        float ds_top = smax_top / float( nums - 1 );
        float ds_bot = smax_bot / float( nums );

        float s_top = 0.;
        float s_bot = 0.;

        for( int is = 0; is < nums; is++ ){
            ProduceVertex( s_bot, t_bot );
            ProduceVertex( s_top, t_top );
            s_top += ds_top;
            s_bot += ds_bot;
        }

        ProduceVertex( s_bot, t_bot );
        EndPrimitive();

        t_top = t_bot;
        t_bot -= dt;
    }
}
```

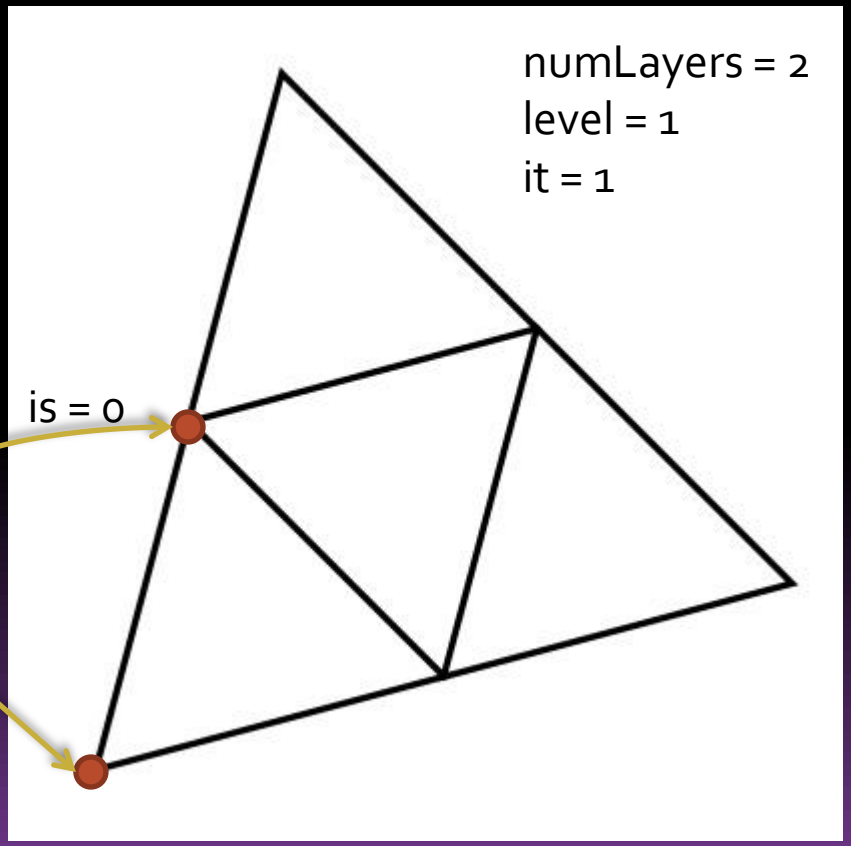


Our geometry shader

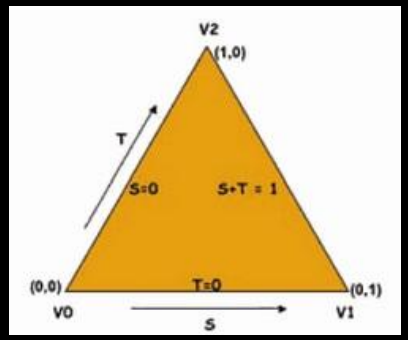


- Main function cont'd

```
    float t_top = 1.;

    for( int it = 0; it < numLayers; it++ ) {
        float t_bot = t_top - dt;
        float smax_top = 1. - t_top;
        float smax_bot = 1. - t_bot;

        int nums = it + 1;
        float ds_top = smax_top / float( nums - 1 );
        float ds_bot = smax_bot / float( nums );

        float s_top = 0.;
        float s_bot = 0.;

        for( int is = 0; is < nums; is++ ){
            ProduceVertex( s_bot, t_bot );
            ProduceVertex( s_top, t_top );
            s_top += ds_top;
            s_bot += ds_bot;
        }

        ProduceVertex( s_bot, t_bot );
        EndPrimitive();

        t_top = t_bot;
        t_bot -= dt;
    }
}
```

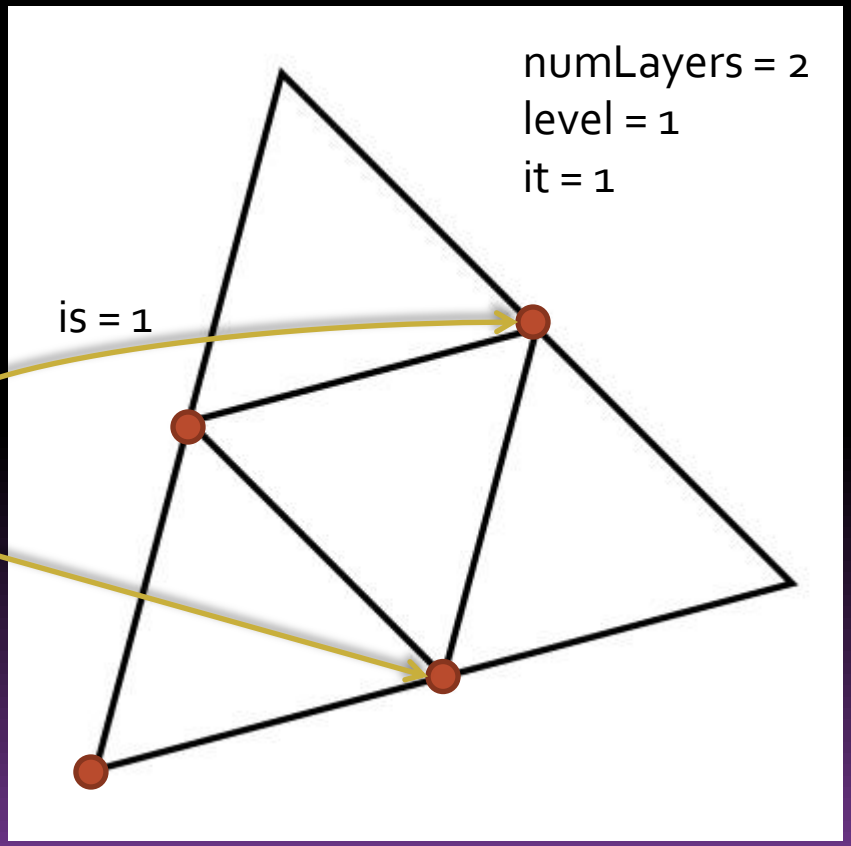


Our geometry shader

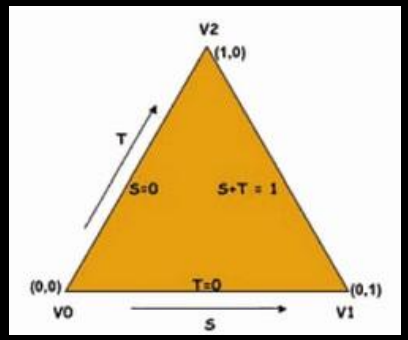


- Main function cont'd

```
    float t_top = 1.;

    for( int it = 0; it < numLayers; it++ ) {
        float t_bot = t_top - dt;
        float smax_top = 1. - t_top;
        float smax_bot = 1. - t_bot;

        int nums = it + 1;
        float ds_top = smax_top / float( nums - 1 );
        float ds_bot = smax_bot / float( nums );

        float s_top = 0.;
        float s_bot = 0.;

        for( int is = 0; is < nums; is++ ){
            ProduceVertex( s_bot, t_bot );
            ProduceVertex( s_top, t_top );
            s_top += ds_top;
            s_bot += ds_bot;
        }

        ProduceVertex( s_bot, t_bot );
        EndPrimitive();

        t_top = t_bot;
        t_bot -= dt;
    }
}
```

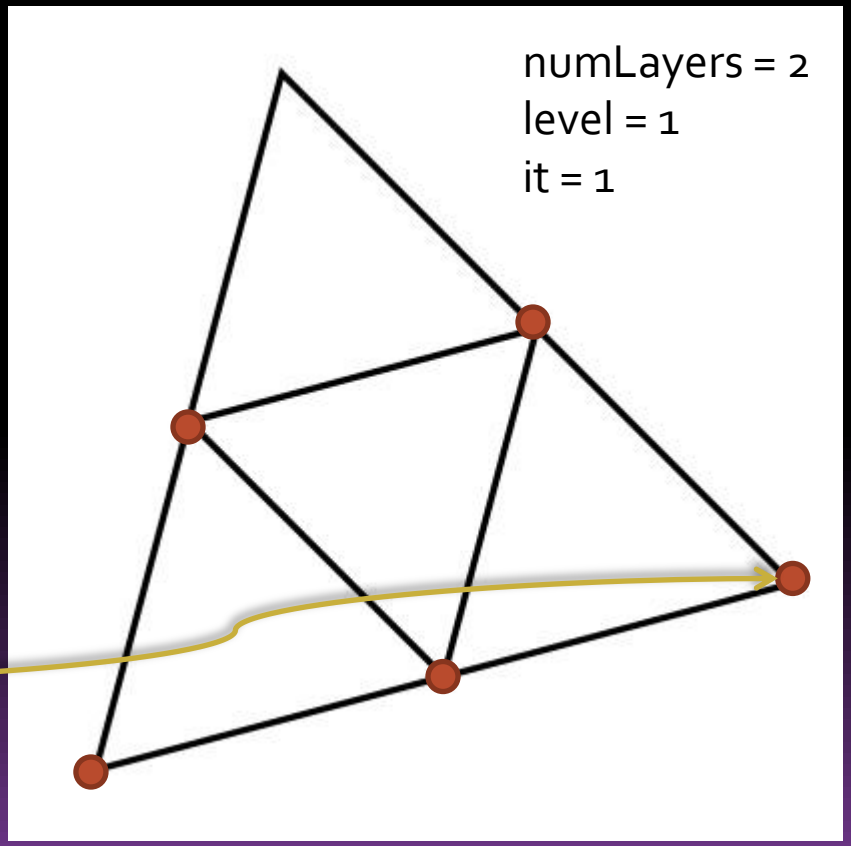


Our geometry shader

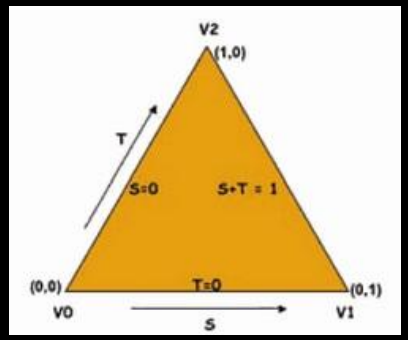


- Main function cont'd

```
    float t_top = 1.;

    for( int it = 0; it < numLayers; it++ ) {
        float t_bot = t_top - dt;
        float smax_top = 1. - t_top;
        float smax_bot = 1. - t_bot;

        int nums = it + 1;
        float ds_top = smax_top / float( nums - 1 );
        float ds_bot = smax_bot / float( nums );

        float s_top = 0.;
        float s_bot = 0.;

        for( int is = 0; is < nums; is++ ){
            ProduceVertex( s_bot, t_bot );
            ProduceVertex( s_top, t_top );
            s_top += ds_top;
            s_bot += ds_bot;
        }

        ProduceVertex( s_bot, t_bot );
        EndPrimitive();

        t_top = t_bot;
        t_bot -= dt;
    }
}
```



Our geometry shader



- ProduceVertex function

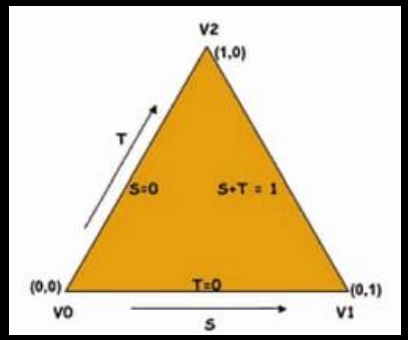
With parameters, compute the vertex position

Because of radius of 1, simply normalize the vector to have the radius

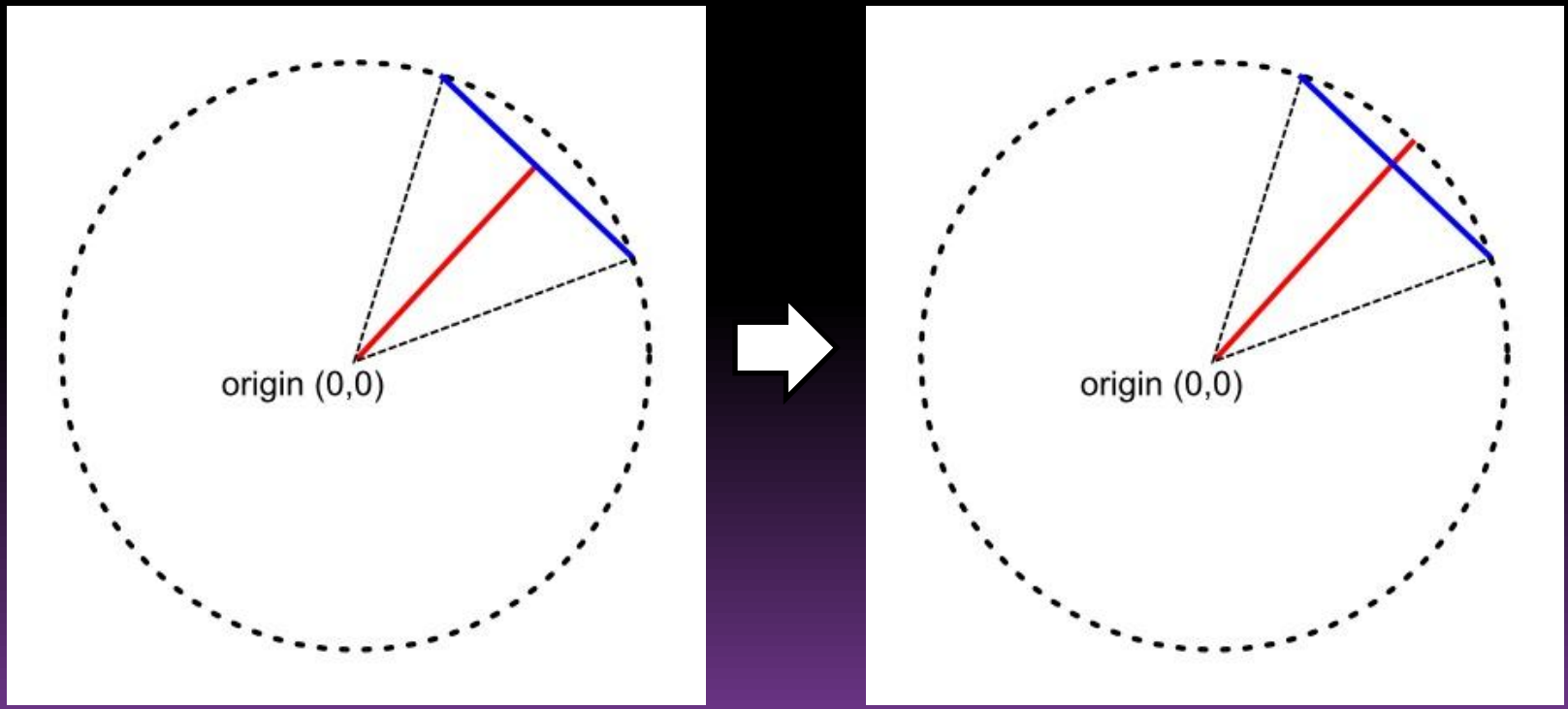
Light part (non-realistic)

```
void ProduceVertex( float s, float t ) {
    // Light position & front color
    const vec3 lightPos = vec3( 50., 40., 20. );
    gl_FrontColor = vec4(.8,1.0,1.0,1.0);

    // Vertex position (coordinate)
    vec3 v = V0 + s*V01 + t*V02;

    // Normal vector
    v = normalize(v);
    vec3 n = v;

    // Normalized normal in Eye space
    // (usefull if modelview matrix contains a non-uniform scale)
    vec3 tnorm = normalize( gl_NormalMatrix * n );

    // Eye coordinate
    vec4 EmitPosition = gl_ModelViewMatrix * vec4( v, 1. );

    // Light calculation
    LightIntensity  = dot( normalize(lightPos - EmitPosition.xyz), tnorm );

    // 2 sided-lighting for better visibility
    LightIntensity = abs(LightIntensity);

    // Final position with emission
    gl_Position = gl_ProjectionMatrix * EmitPosition;
    EmitVertex();
}
```

Our geometry shader



- ProduceVertex function
  - Reach the border of the sphere by normalize because of the radius of the sphere is 1 and the origin is the centre.



Pros and Cons

- Pros :
  - Allow the GPU to reduce the load of the CPU
  - Modify on the fly the geometry
- Cons :
  - Limitations for LoD:
    - Limited number of new vertices: usually 1024
    - Limited access to surrounding information
    - Extension for tessellation available in openGL 4.0
      - (GeForce GTX 4** and Radeon HD 5***)
  - Only for recent graphics cards

References

http://cirl.missouri.edu/gpu/glsl_lessons/glsl_geometry_shader/index.html

Graphics Shaders: Theory and Practice by Mike Bailey, Steve Cunningham

http://en.wikipedia.org/wiki/GLSL#A_sample_trivial_GLSL_Geometry_Shader

http://developer.download.nvidia.com/opengl/specs/GL_EXT_geometry_shader4.txt

For the others references we used during the slides, please look backward