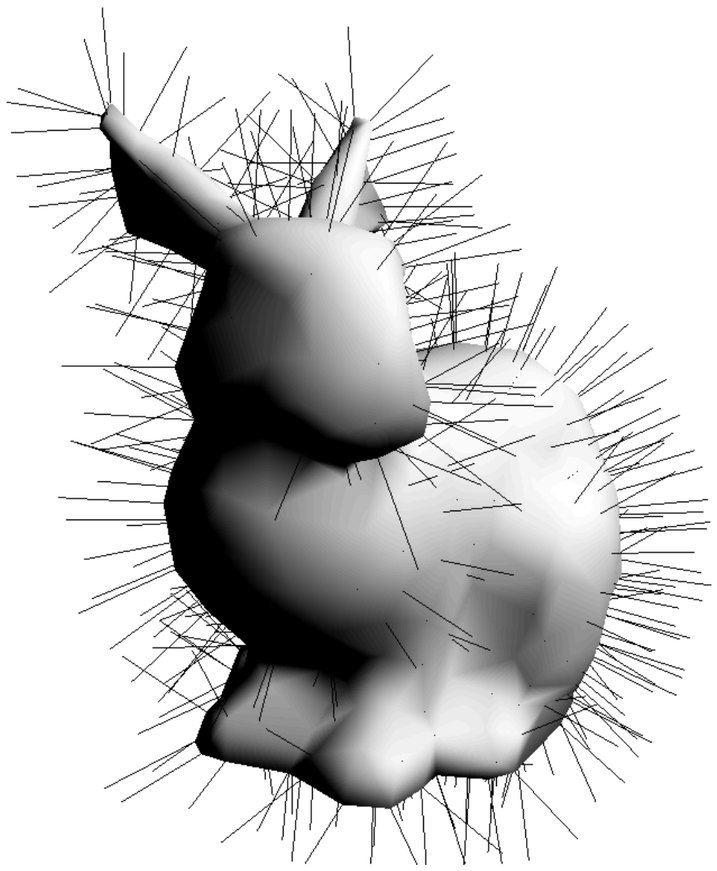
Geometry shaders and Tesselation shaders

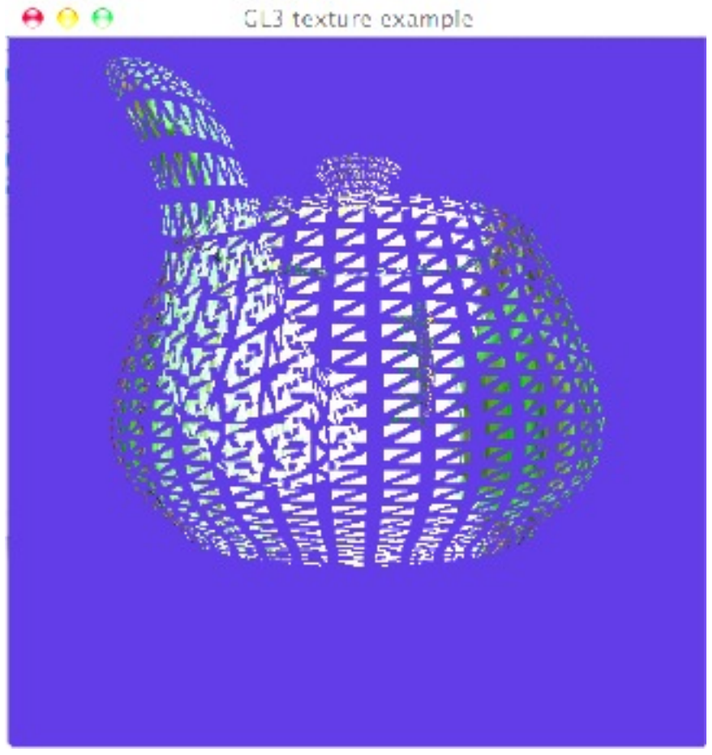
Additional shader stages in the geometry stage in the OpenGL pipeline

Can modify, add and remove geometry.

Can produce other kinds of geometry than was is entered.

Geometry shaders





Geometry shaders

OpenGL 3 (extension in GL 2)

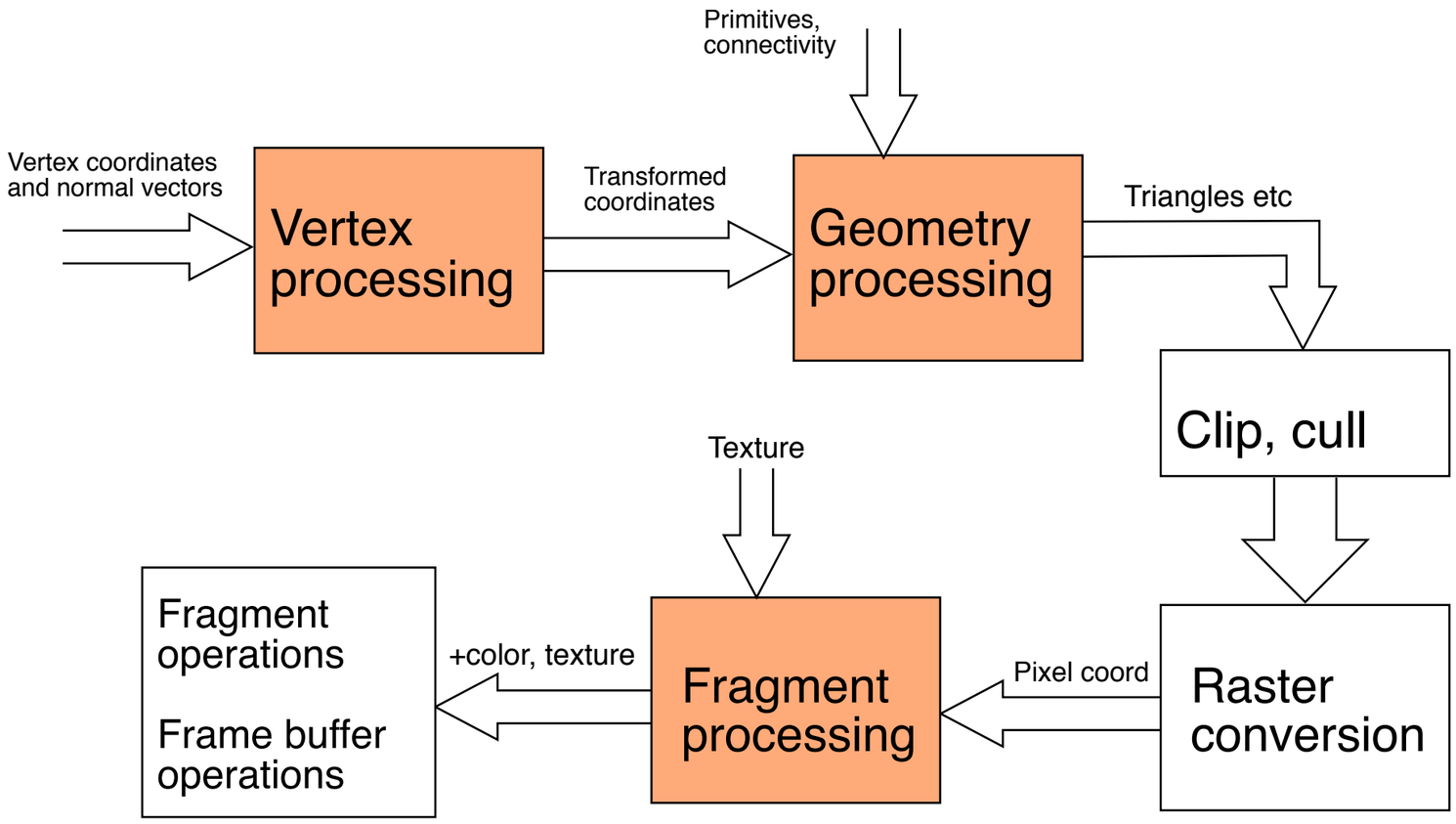
Shader between vertex and fragment, processes geometry, can add new geometry

Modest hardware demands: G80 or better (2007+)

Core since GL3.

Some limitations, had poor performance for some applications. Improved on newer GPUs!

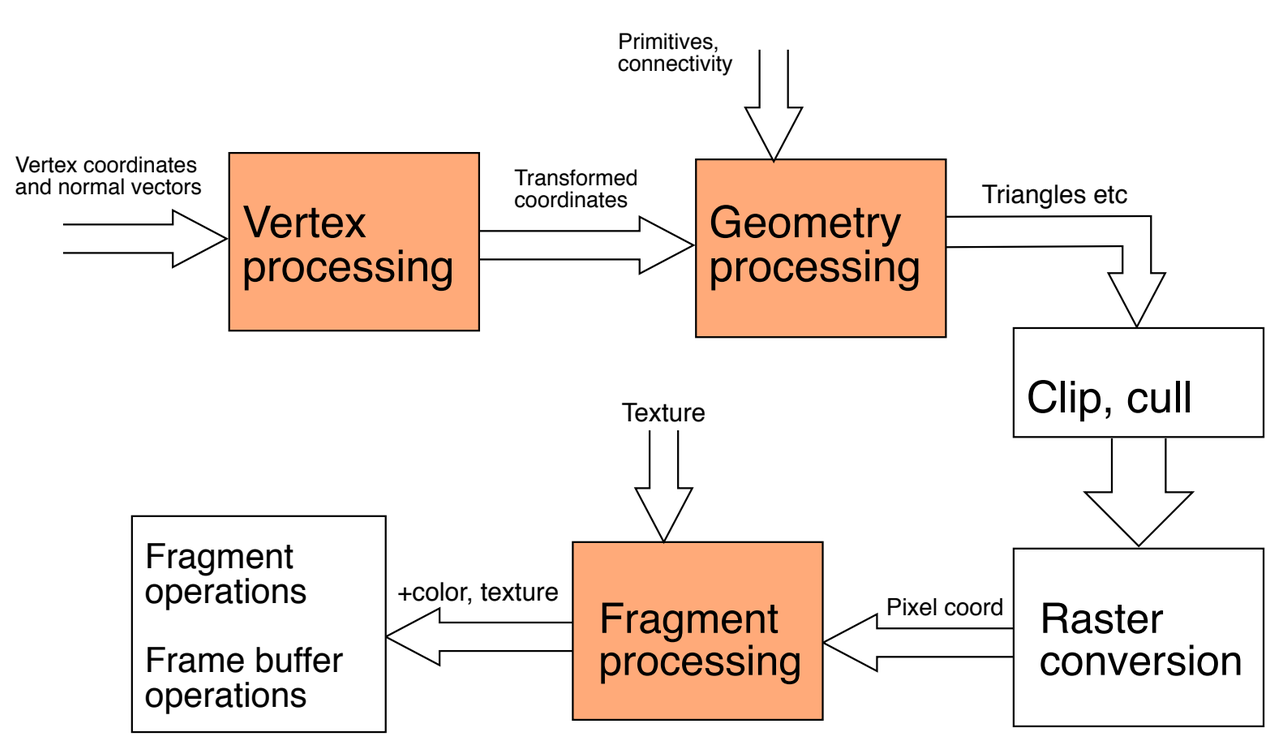
OpenGL pipeline



Geometry stage:

Shaders can modify, add or remove geometry.

Unlike the vertex stage it works on entire primitives.



Applications for geometry shaders:

- Splines/spline-ytor
- Edge extraction, silhouettes
- Shadow volumes
- Effects on polygon level (e.g. breaking up a mesh by shrinking triangles)
- Dynamic hair/grass defined from a set of "root points".
- Adaptive subdivision (including displacement mapping)
- Visualization of normal vectors etc
- Flat shading
- Wireframes

Geometry shader example

Input: A single triangle (as first example)

Load geometry shader together with vertex and fragment.

Indata to shader: triangles, lines or points

Output: triangle strips, line strips

Pass-through geometry shader

```
#version 150

layout(triangles) in;
layout(triangle_strip, max_vertices = 3) out;

void main()
{
  for(int i = 0; i < gl_in.length(); i++)
  {
    gl_Position = gl_in[i].gl_Position;
    EmitVertex();
  }
}
```

Pass on same vertex

Report that a primitive is finished!

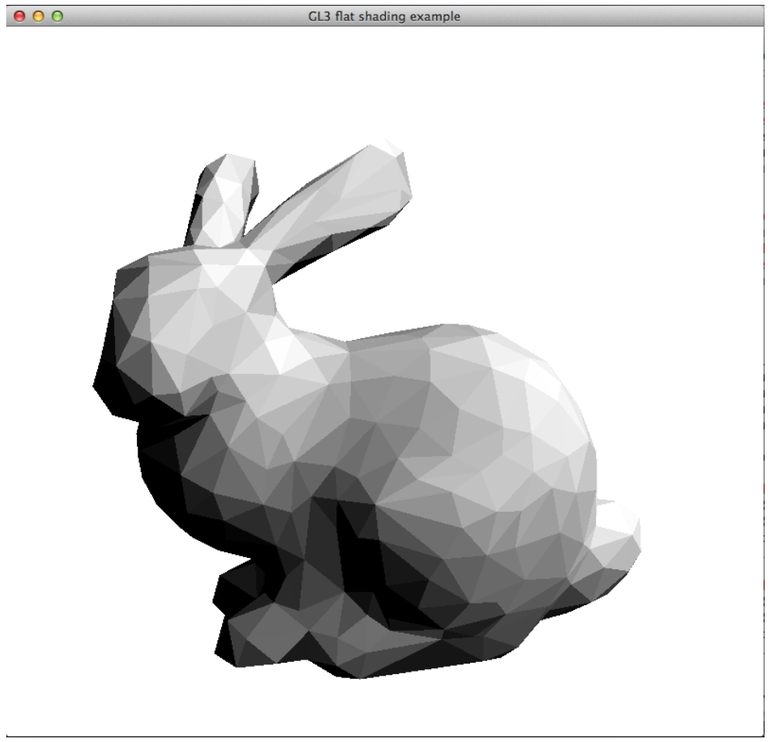
Flat shading

Easy with geometry shaders: Take the average of all vertex normals in GS, calculate light from that.

```
void main()
{
  vec3 avgNormal = vec3(0.0);

  for(int i = 0; i < gl_in.length(); i++)
  {
    avgNormal += exNormal[i];
  }
  avgNormal /= gl_in.length();
  avgNormal = normalize(avgNormal);

  for(int i = 0; i < gl_in.length(); i++)
  {
    gl_Position = gl_in[i].gl_Position;
    texCoordG = texCoord[i];
    exNormalG = avgNormal;
    EmitVertex();
  }
  EndPrimitive();
}
```



A bit of a hassle to create something trivial...

Wireframe

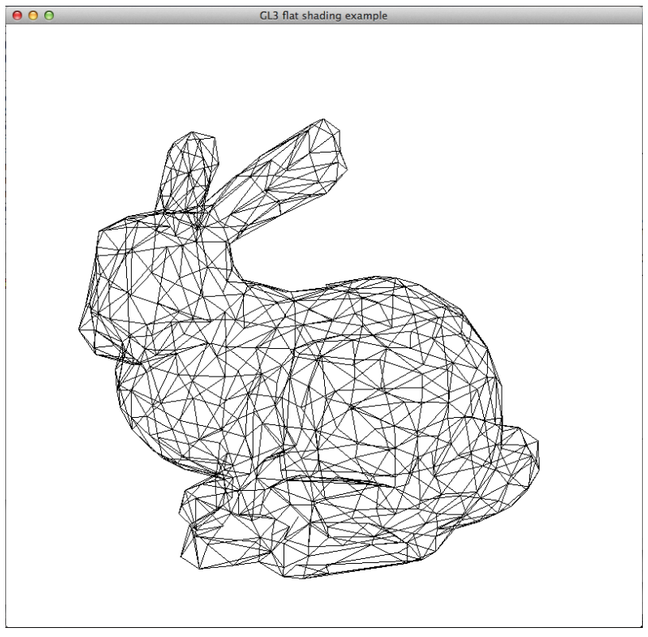
Wireframes can be generated by the geometry shader directly from polygons.

Convert every triangle to line strips in GS.

Very useful for visualizing geometry.

Easy: line_strip out instead of triangle_strip!

```
layout(triangles) in;
//layout(triangle_strip, max_vertices
    = 90) out; // Normal, solid
layout(line_strip, max_vertices = 90)
    out; // Wireframe
```



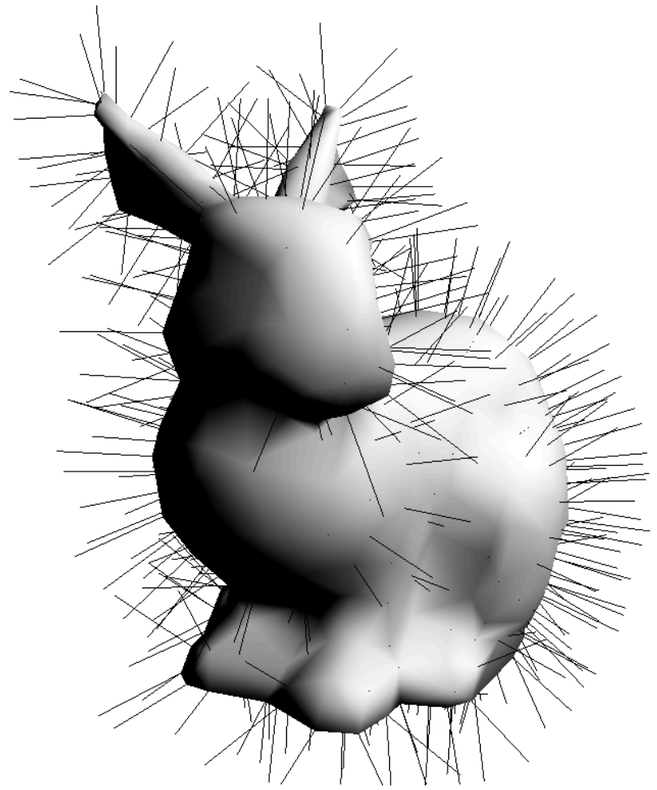
Hair and grass

Use the geometry as starting points, generate lines/curves from these

```
  for(int i = 0; i < gl_in.length(); i++)
  {
    gl_Position = projMatrix * gl_in[i].gl_Position;
    texCoordG = texCoord[i];
    exNormalG = exNormal[i];
    EmitVertex();
    // I could add more lines here!
    gl_Position = projMatrix * (gl_in[i].gl_Position +
vec4(normalize(exNormal[i])*0.3, 0.0));
    texCoordG = texCoord[i];
    exNormalG = exNormal[i];
    EmitVertex();

    EndPrimitive();
  }
```

Notable problem: Must do projection after the extraction of "hair"



(The figure only shows normal vectors)

Crack shader

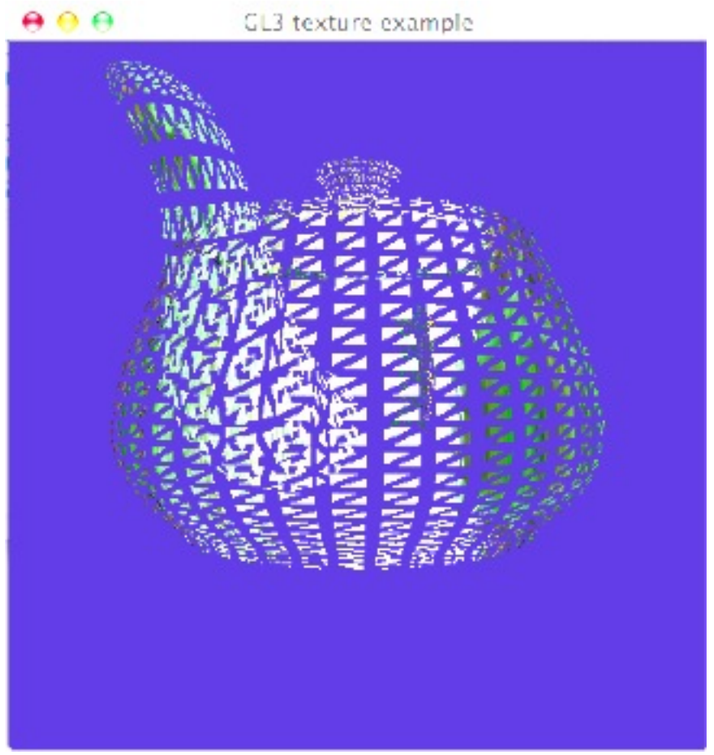
Not so useful but an amusing effect. Move all vertices closer to the center of the triangle!

```
void main()
{
  vec4 middleOfTriangle = vec4(0.0);
  float tw = 1.0 / offs;

  for(int i = 0; i < gl_in.length(); i++)
    middleOfTriangle += gl_in[i].gl_Position;
  middleOfTriangle /= gl_in.length();

  for(int i = 0; i < gl_in.length(); i++)
  {
    gl_Position = (gl_in[i].gl_Position * offs) +
(middleOfTriangle) * (1.0 - offs);

    texCoordG = texCoord[i];
    exNormalG = exNormal[i];
    EmitVertex();
  }
  EndPrimitive();
}
```



How about tesselation...?

Add more geometry in order to

1) make a rough polygon model smoother

2) add detail (e.g. displacement mapping etc)

3) Handle level-of-detail

Often based on splines

Can be performed both in geometry and tesselation shaders

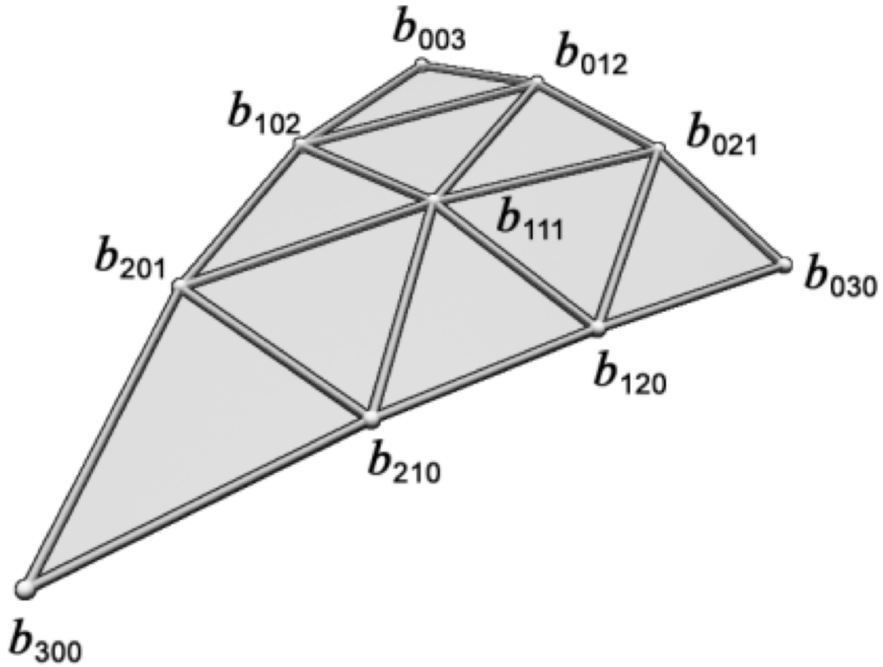
Curved PN Triangles

Suitable for geometry shaders

Only needs vertices and normal vectors for one triangle at a time. Not even adjacency! Comfortable.

PN = Point + Normal

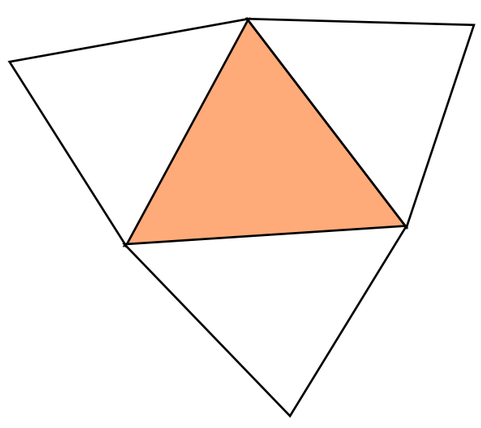
Curved PN Triangles



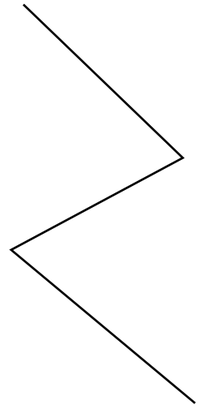
Adjacency

Geometry can be passed together with information of neighbor geometry

This can be used e.g. for tesselation



Triangle with adjacency



Line with adjacency

More applications: Shadow volumes, edge extraction

Creating adjacency data

Most efficiently: Use half-edge or similar structure to find neighbors efficiently

I was lazy and brute forced it...

```
// Brute force adjacency builder.
// A better method would be to collect all triangles for all vertices (complete navigation information) and work from there.
// Loop through all triangles. For each vertex, count number of triangles.
// Allocate lists of triangles.
// Loop through all triangles. For each vertex, build a list of triangles.
// An alternative is to dynamically change the triangle list, with realloc or Vector push_back().

GLint *buildAdjacencyIndex(Model *m)
{
    GLint *newIndexArray = malloc(sizeof(GLint) * m->numIndices * 2); // 6 per triangle instead of 3

    for (int i = 0; i < m->numIndices; i+=3)
    {
        // Find any triangle with both these EXCEPT the current one!
        int c = findThird(m, m->indexArray[i], m->indexArray[i+1], m->indexArray[i+2]); // Third argument = the one to avoid
        // Put these on newIndexArray!
        newIndexArray[i*2] = m->indexArray[i];
        newIndexArray[i*2+1] = c;
        newIndexArray[i*2+2] = m->indexArray[i+1];

        c = findThird(m, m->indexArray[i+1], m->indexArray[i+2], m->indexArray[i]); // Third argument = the one to avoid
        newIndexArray[i*2+3] = c;
        newIndexArray[i*2+4] = m->indexArray[i+2];

        c = findThird(m, m->indexArray[i+2], m->indexArray[i], m->indexArray[i+1]); // Third argument = the one to avoid
        newIndexArray[i*2+5] = c;
    }
    return newIndexArray;
}
```

Drawing with adjacency

The adjacency data replaced the original index list:

```
// Build an index array with adjecency
adjacencyIndex = buildAdjacencyIndex(m);

// Replace the index array
glBindVertexArray(m->vao);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m->ib);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, m->numIndices*sizeof(GLuint)*2,
adjacencyIndex, GL_STATIC_DRAW);
```

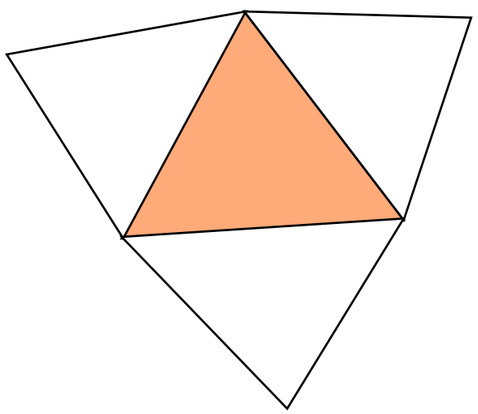
Then I made a variant of DrawModel (copied from LOL and edited), only needed change was:

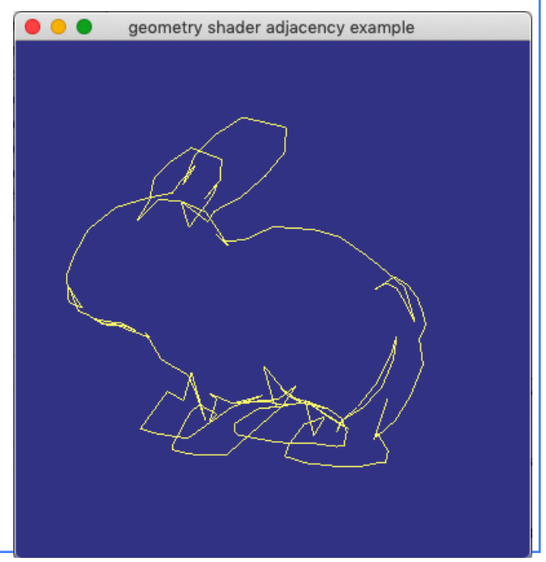
```
glDrawElements(GL_TRIANGLES_ADJACENCY, m->numIndices*2,
GL_UNSIGNED_INT, 0L);
```

Adjacency example

Edge extraction

For each of the three edges of the center triangle, compare Z sign of nomal of the two neighbors! If different, draw that edge!





Instancing of geometry shaders

Performance problem when few elements go in and many out!

Instancing run g.s. several times per input primitive!

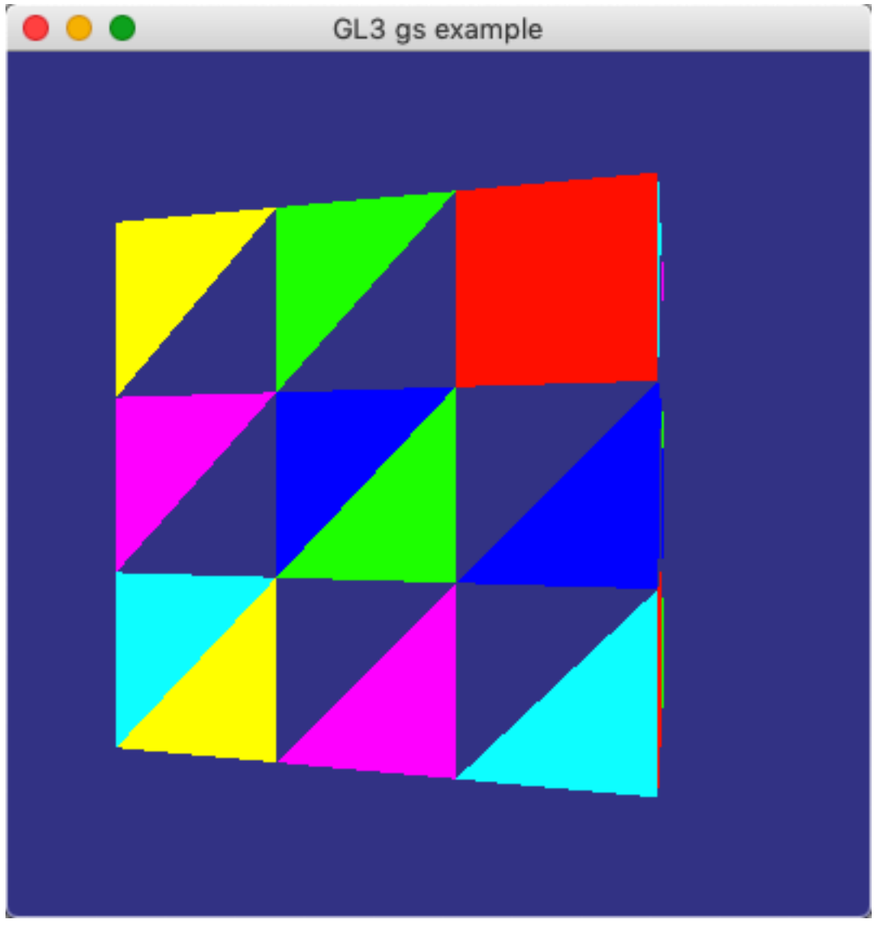
layout(invocations = num_instances) in;

gl_InvocationID

At least 32 instances are guaranteed to be possible!

Standard from OpenGL 4.0!

Geometry shader instancing example




```
#version 410

layout(triangles) in;
layout(triangle_strip, max_vertices = 3) out;
layout(invocations = 6) in;

out vec4 exColor;

void main()
{
  vec4 middleOfTriangle = vec4(0.0);
  vec4 v1,v2,v3;

  // Every triangle is split in 6 triangles
  // but only one is made by each instance.
  // Pretty stupid; no claims of good performance.

  switch (gl_InvocationID)
  {
     case 0:
       v1 = gl_in[0].gl_Position;
       v2 = (gl_in[0].gl_Position*2 + gl_in[1].gl_Position)/3;
       v3 = (gl_in[0].gl_Position*2 + gl_in[2].gl_Position)/3;
       exColor = vec4(1,0,0,1);
       break;
     case 1:
```

Dynamic texturing example

Simple scene builder. Arbitrary sized blocks.

But I want the texture to be the same on all sides regardless of size!

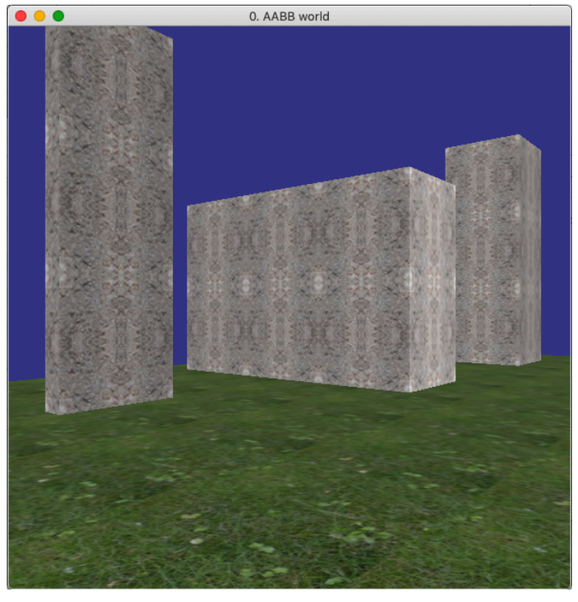
Can't I just scale by the size...? But I do not know the direction of polygons just from vertex position!

Solution: Geometry shader!

My AABB world

I just state the size and the geometry shader scales the texture from the size of the polygon being drawn!

Such a simple problem... when solved by the geometry shader!



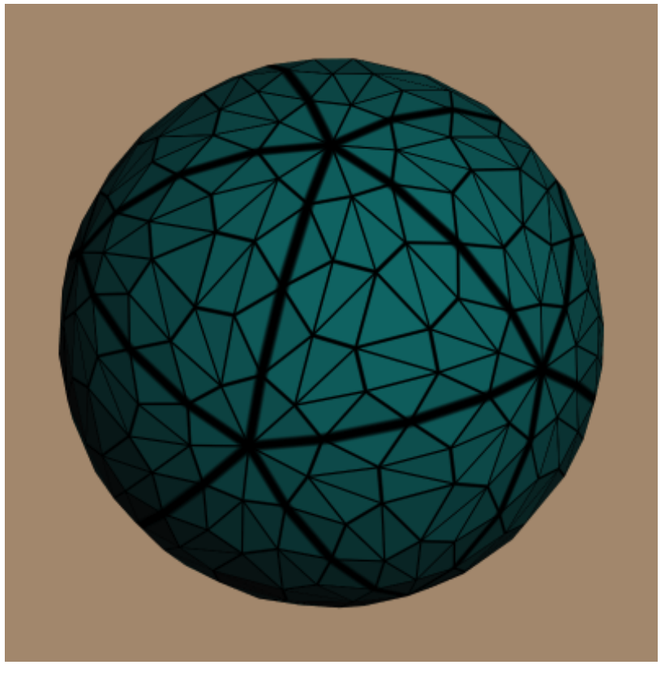
Conclusions

Geometry shaders are highly useful with many applications.

One extra shader stage does not complicate things much.

Enough for a lot - but not everything.

Tesselation shaders



Tesselation shaders

Problem with early geometry shaders:
Much data from little indata inefficient, serialized.

Remedy 1: Instancing in newer hardware, faster geometry shaders

Remedy 2: Tesselation shaders

Tesselation shaders

Located between the vertex shader and the geometry shader

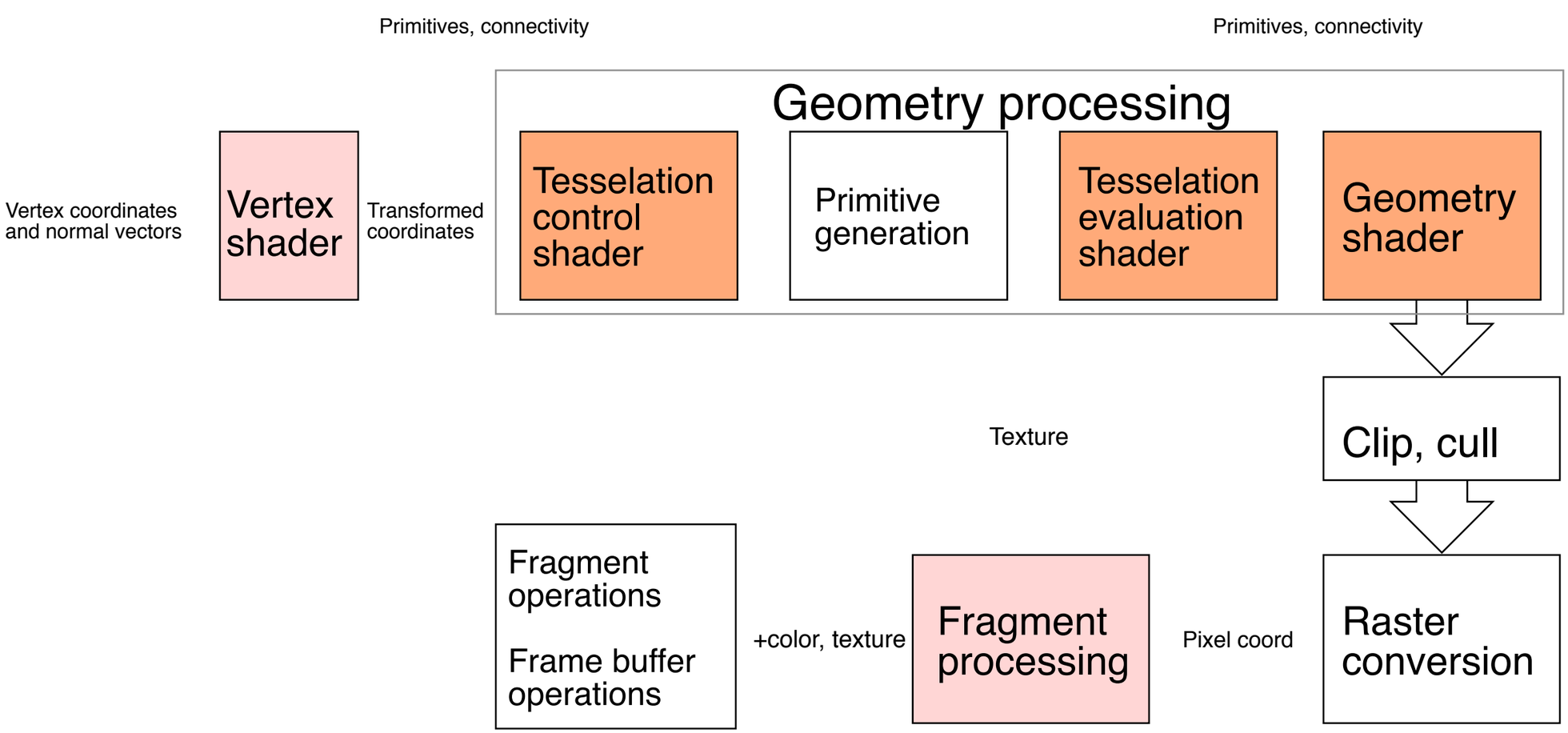
Not one shader but two, plus non-programmable stages between.

1) Tesselation control shader

2) Primitive generation

3) Tesselation evaluation shader

OpenGL pipeline - extended



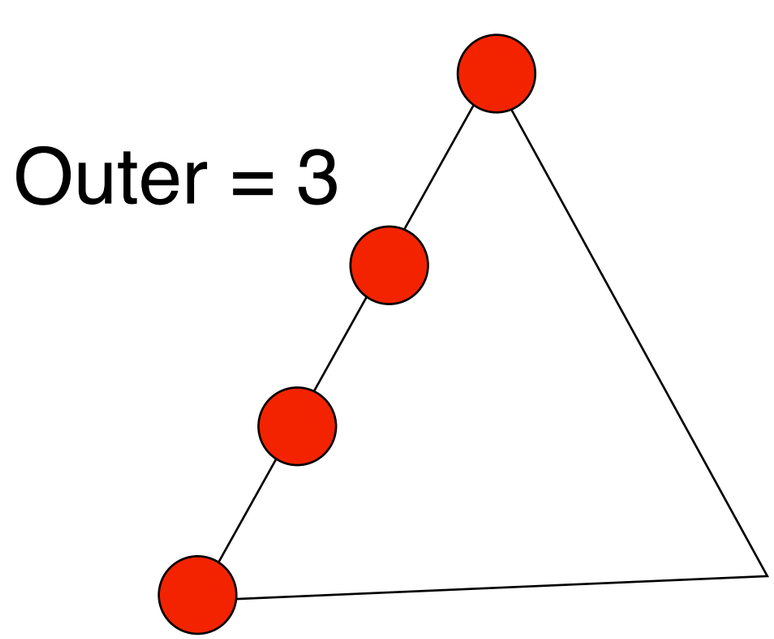
Tasks for tesselation shaders

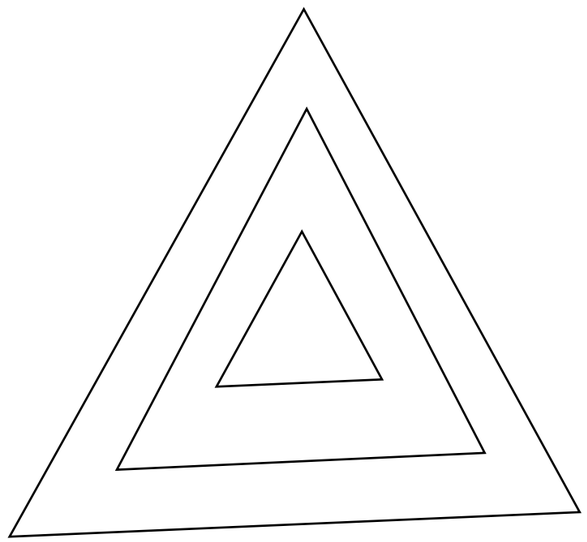
Indata: ”Patches”, a number of vertices without any given constellation.

The TC shader sets a desired tesselation level per edge and number of inner levels.

The TE shader calculates the final positions in the desired way (e.g. using a spline)

Example: Triangle





Inner = 3

Example: Tesselation Control

```
#version 400
layout(vertices = 3) out;
in vec3 vPosition[]; // From vertex shader
out vec3 tcPosition[]; // Output of TC

uniform float TessLevelInner; // Sent from main program
uniform float TessLevelOuter;

void main()
{
    tcPosition[gl_InvocationID] = vPosition[gl_InvocationID]; // Pass on vertex
    if (gl_InvocationID == 0)
    {
        gl_TessLevelInner[0] = TessLevelInner; // Decide tesselation level
        gl_TessLevelOuter[0] = TessLevelOuter;
        gl_TessLevelOuter[1] = TessLevelOuter;
        gl_TessLevelOuter[2] = TessLevelOuter;
    }
}
```

Example: Tesselation Evaluation

```
#version 400

layout(triangles, equal_spacing, cw) in;
in vec3 tcPosition[]; // Original patch vertices

void main()
{
    vec3 p0 = gl_TessCoord.x * tcPosition[0]; // Barycentric!
    vec3 p1 = gl_TessCoord.y * tcPosition[1];
    vec3 p2 = gl_TessCoord.z * tcPosition[2];
    gl_Position = vec4(p0 + p1 + p2, 1);
// Sum with weights from the barycentric coords any way we like

// Apply vertex transformation here if we want
}
```

Barycentric coordinates

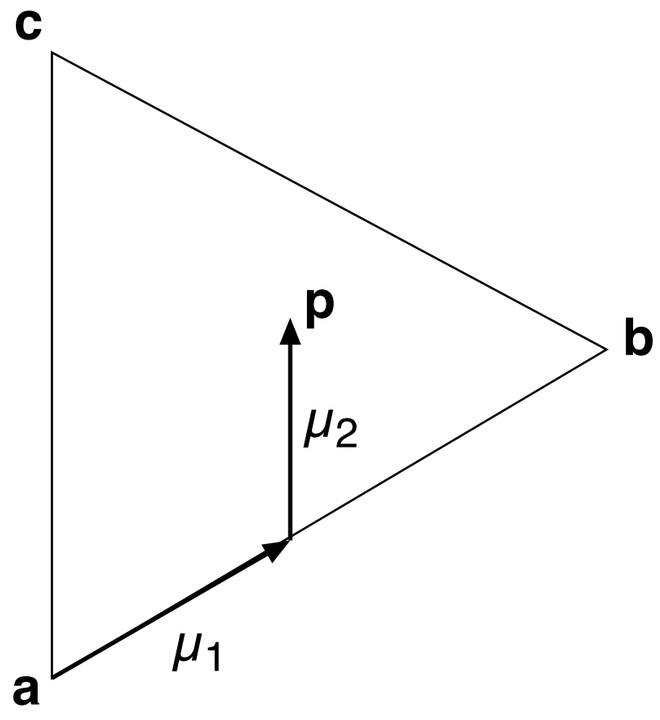
A coordinate system that is defined by a simplex
(i.e. triangle)

$$\mathbf{p} = a \cdot \mathbf{p_1} + b \cdot \mathbf{p_2} + c \cdot \mathbf{p_3}$$

a, b, c uniquely describes a point from the vertices

if a, b, c all are ≥ 0, we are inside the triangle

I figured out barycentric coordinates like this:



$$\mathbf{p} = \mathbf{a} + (\mathbf{b}-\mathbf{a})\mu_1 + (\mathbf{c}-\mathbf{a})\mu_2 = \mathbf{a}\,(1 - \mu_1 - \mu_2) + \mathbf{b}\,\mu_1 + \mathbf{c}\,\mu 2$$

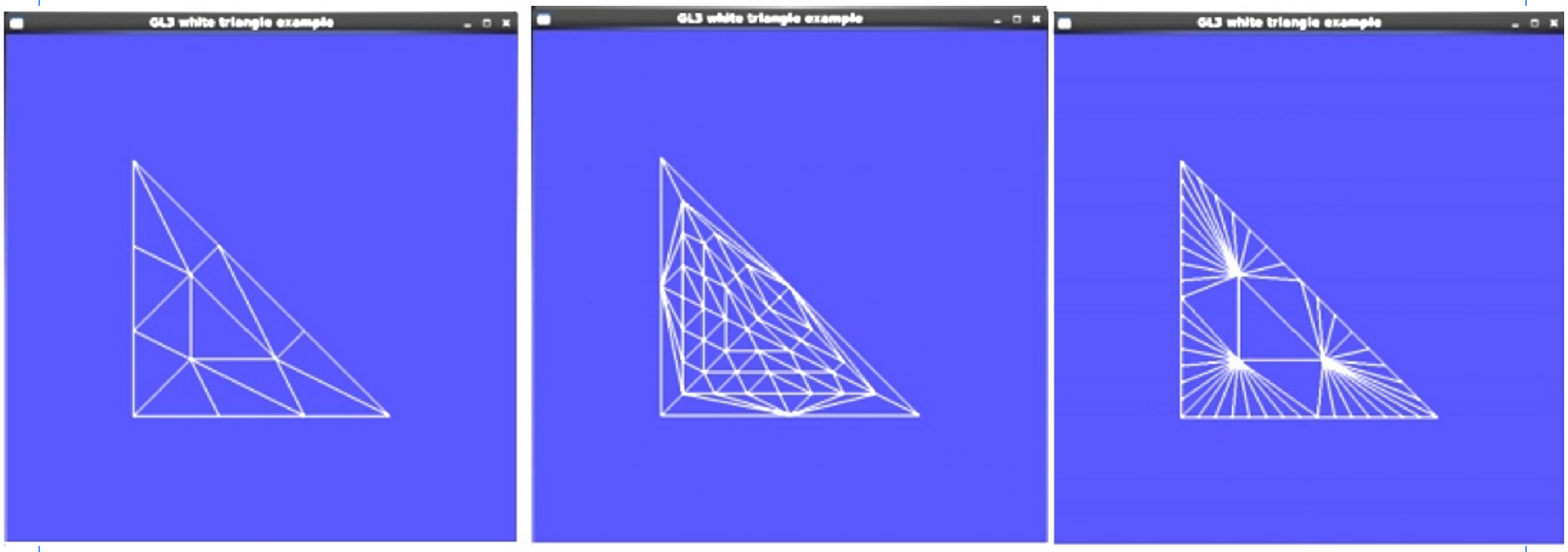
Control + Evaluation

Control decides how many levels of tesselation that is desired.

Evaluation is called multiple times. Each time we get unique coordinated from which we should calculate the resulting vertex position

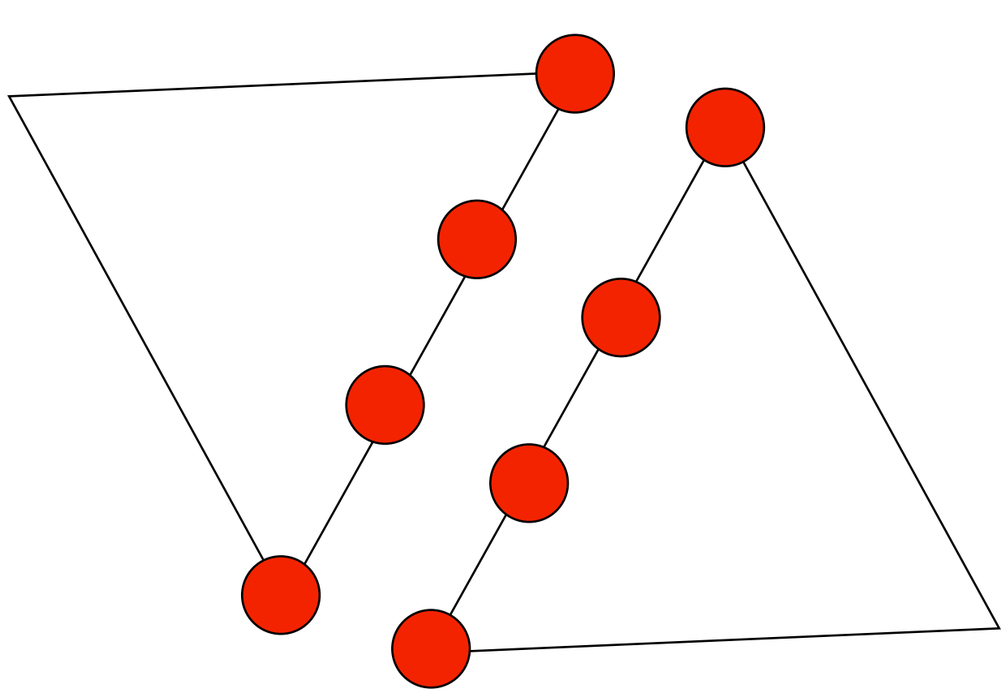
Result

Variation by outer and inner



”Outer” is by edge

Important!

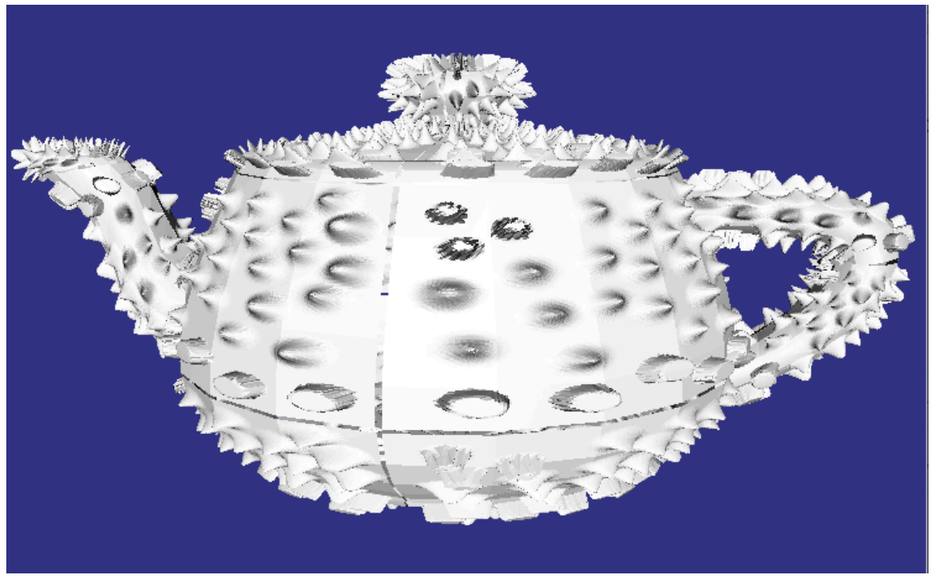


If you calculate the edge in a way that gives the same resolution in both patches for an edge shared by neighbor patches, you can avoid gaps!

Application: The heavy metal teapot

Displacement mapping combined with tesselation

A bump map shows where to displace the geometry.



Conclusions on tesselation shaders

Double stages - mode complicated than geometry shaders

Vertex shader gets superfluous

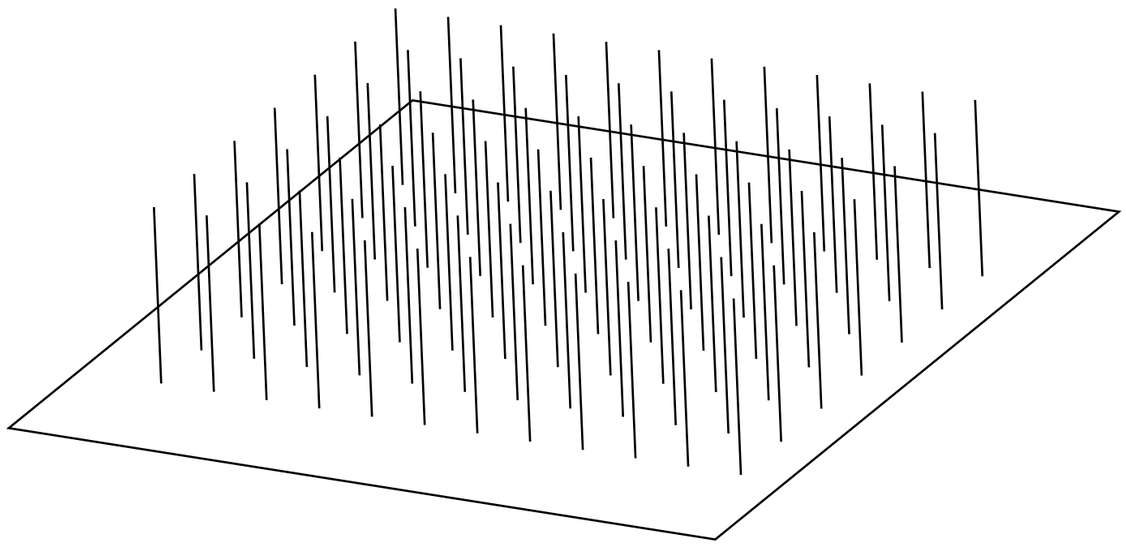
The point is tesselation, not much else. More applications for Geometry shaders?

Do we have enough shader stages yet?

Grass and fur

Nice application of geometry shaders

Spawn strands over given polygons

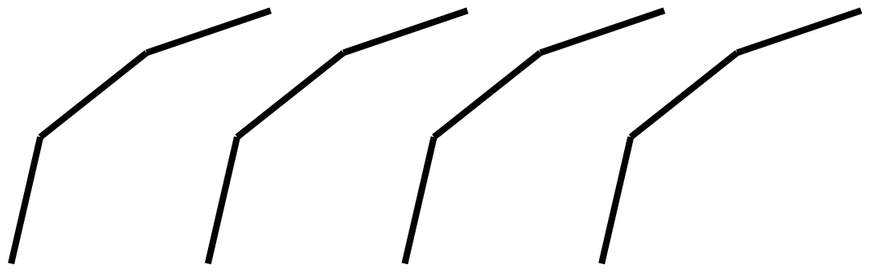


Bending grass

Bend, wave with wind - use several sections

Simple case or *tropism*

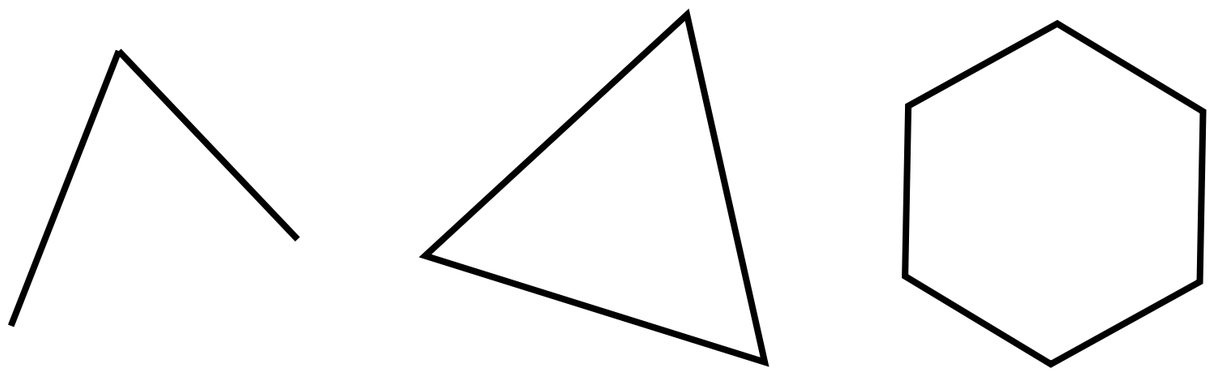
The wind needs to vary (consider harmonic functions) to vary the bend over the grass.



Don't make it disappear

Flat polygons get invisible when watches from the side

It needs thickness



Just two edges (4 triangles per section)

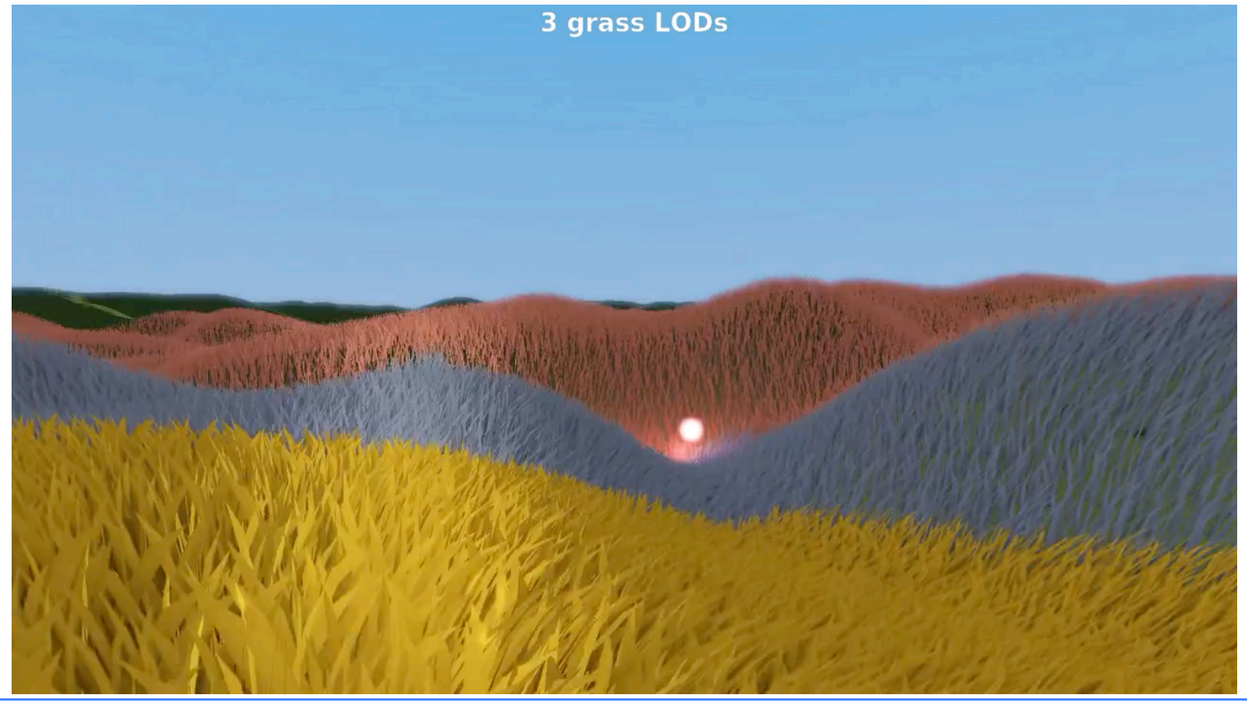
Triangular pipe

More detailed pipe

Level of detail on distance

Fewer strands

or switch to billboards (more about these next time)



Light

Make the grass darker by depth to approximate the occlusion effect

You can also use ambient occlusion (dampen based on proximity)

That's all, folks!