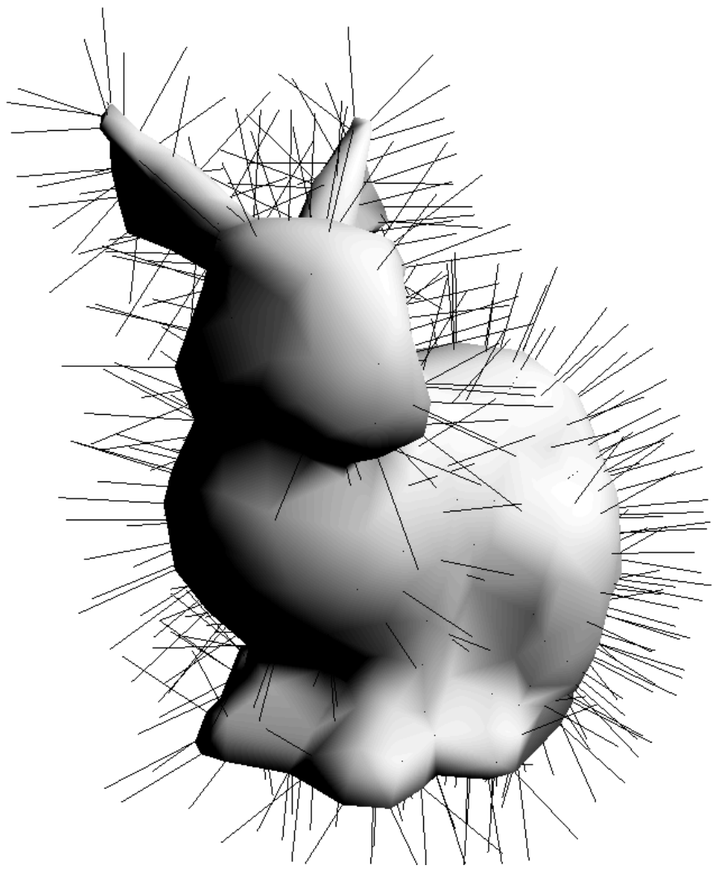
Geometry shaders och Tesselation shaders

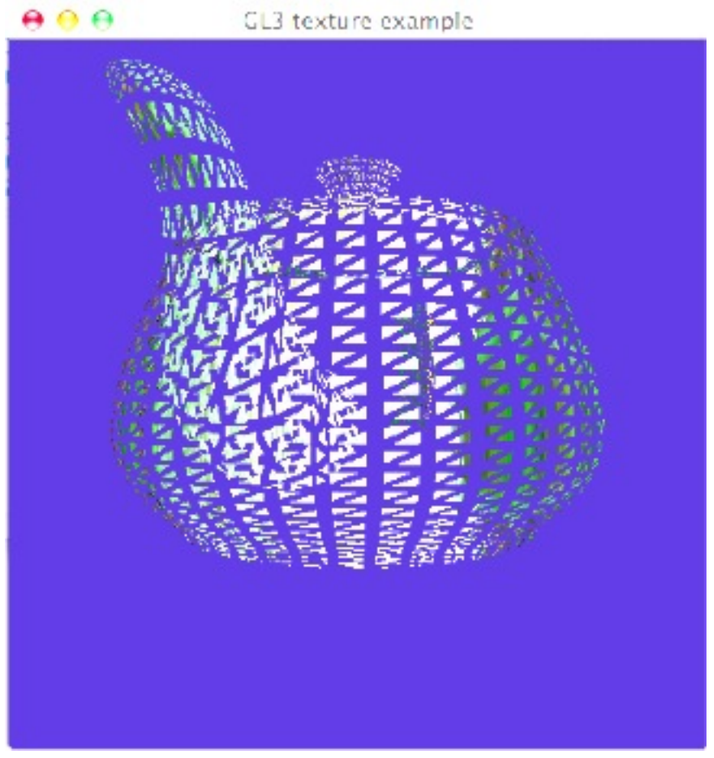
Ytterligare shadersteg i geometristeget i OpenGL-pipelinen

Kan modifiera, lägga till och ta bort geometri

Kan mata ut andra sorters geometri än vad som matas in

Geometry shaders





Geometry shaders

OpenGL 3 (extension i GL 2)

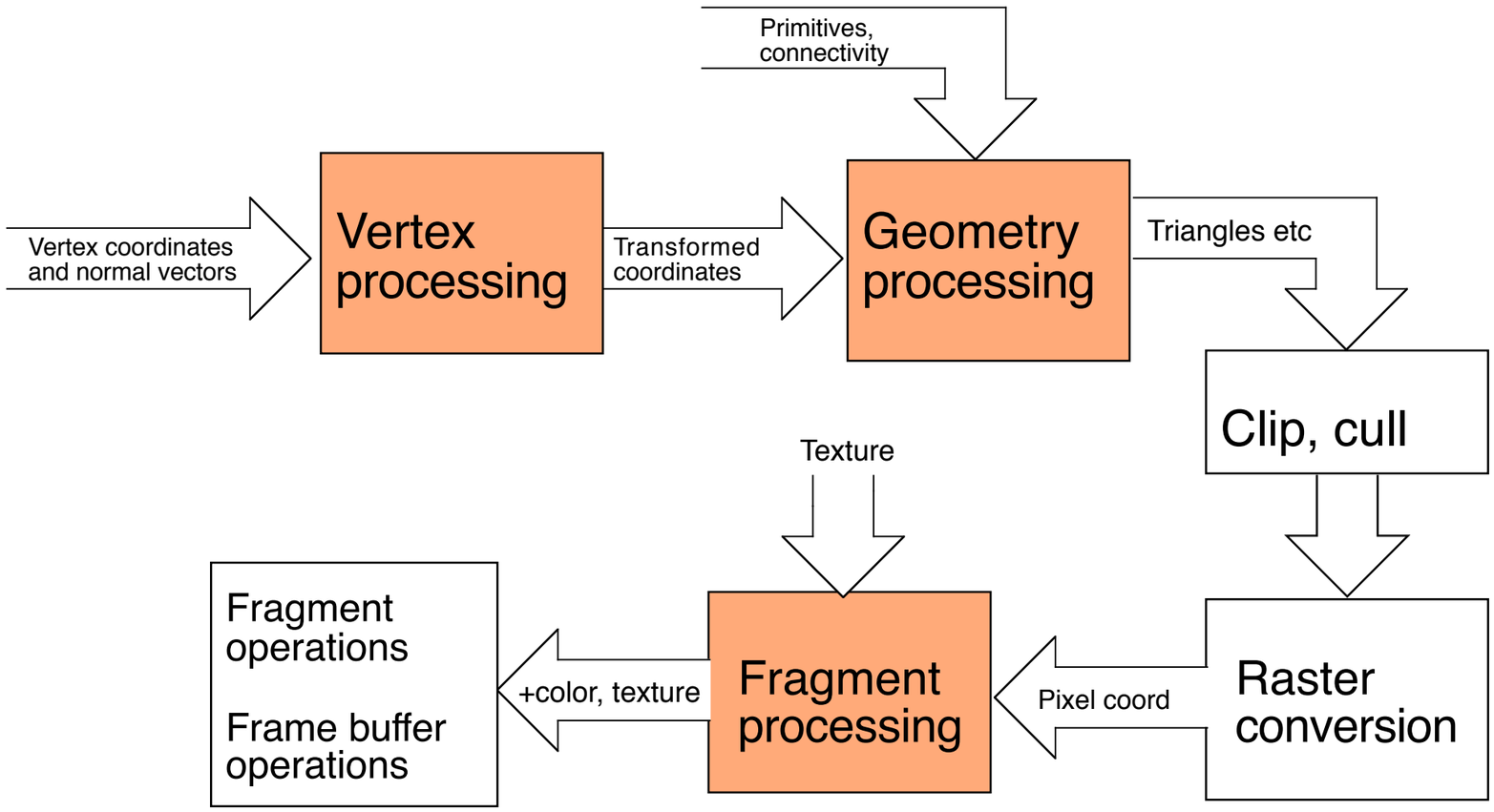
Shader mellan vertex och fragment, bearbetar geometri, kan lägga till ny geometri

Blygsamma hårdvarukrav: G80 eller bättre (2007+)

Core sedan GL3.

Vissa begränsningar, kritiserat för dålig prestanda. Dock förbättrat på nyare GPUer!

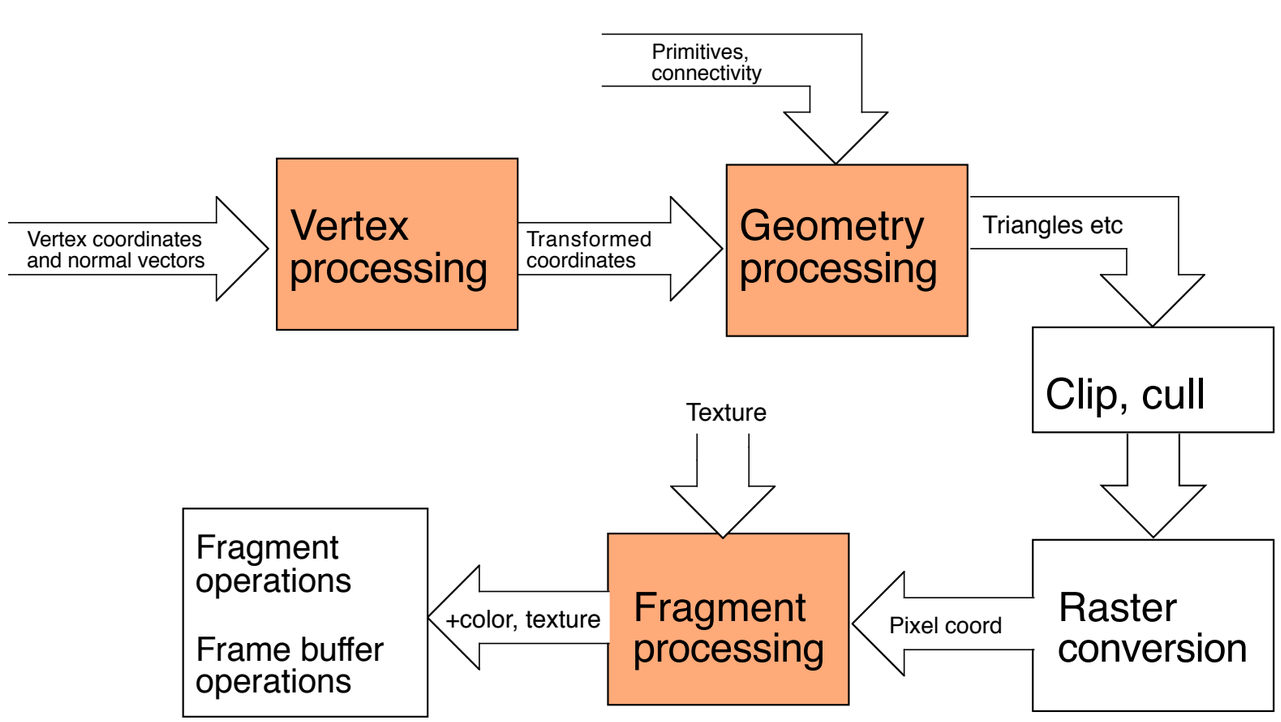
OpenGL pipeline



Geometry stage:

Shaderprogram kan modifiera, lägga till eller ta bort geometri.

TIll skillnad från vertexsteget jobbar vi på hela primitiver.



Tillämpningar för geometry shaders:

- Splines/spline-ytor
- Kantextraktion, silhuetter
- Shadow volumes
- Effekter på polygonnivå (t.ex. uppbruten mesh genom att krympa trianglar)
- Dynamiskt hår/gräs definierat från en uppsättning "rotpunkter".
- Adaptive subdivision (inkl displacement mapping)
- Visualisering av normalvektorer etc
- Flat shading
- Wireframes

Geometry shader-exempel

Input: En enda triangel (som första exempel)

Ladda geometry shader tillsammans med vertex och fragment.

Indata till shader: triangel, linje eller punkt

Output: triangle strip, line strip

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();
  }
}
```

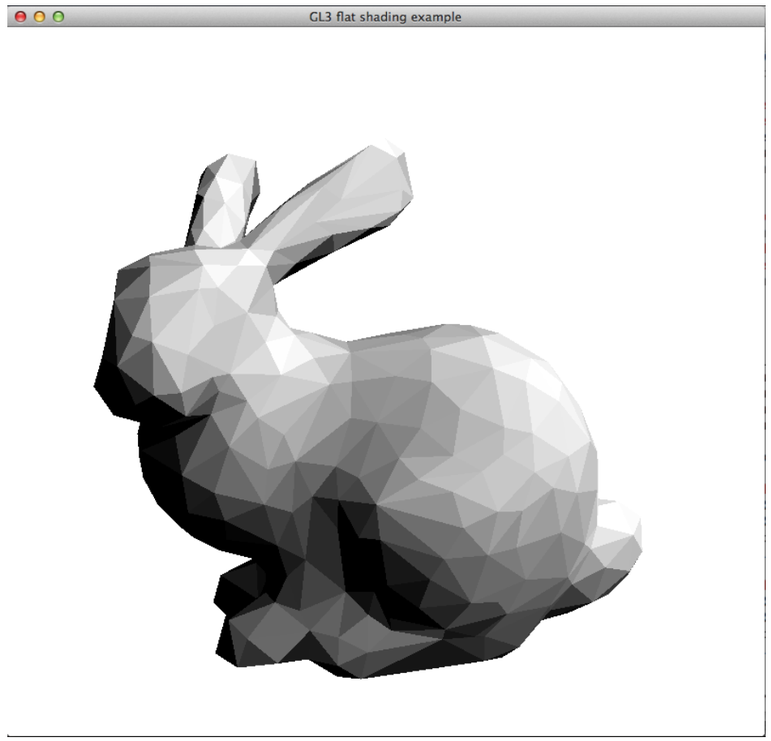
Skicka vidare samma vertex

Tala om att en primitiv är klar!

Flat shading

Lätt med geometry shaders: Tag medelvärde av alla normaler i GS, beräkna ljus från detta.

```
void main()
{
  vec4 middleOfTriangle = vec4(0.0);
  vec3 avgNormal = vec3(0.0);;

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



En aning bök för att få till något trivialt...

Wireframe

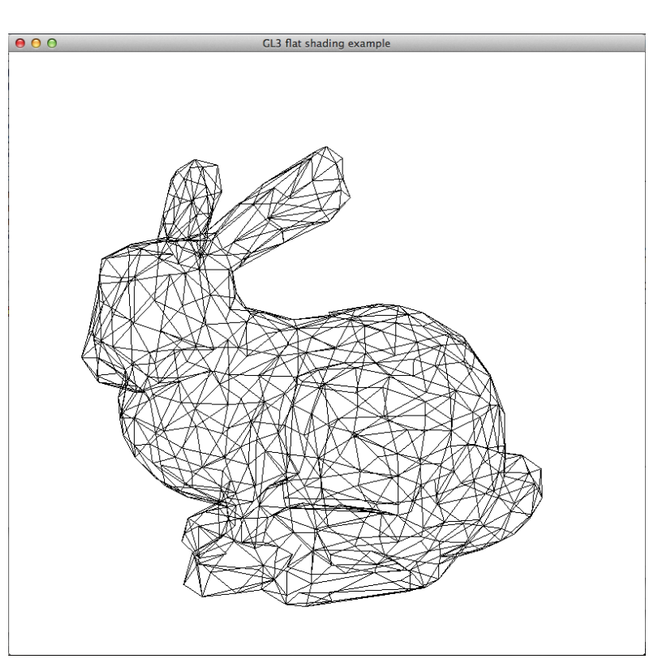
Wireframes kan genereras av geometry shader direkt från polygoner.

Omvandla varje triangel till line strip i GS.

Mycket användbart för visualisering av geometri.

Lätt: line_strip ut i stället för triangle_strip!

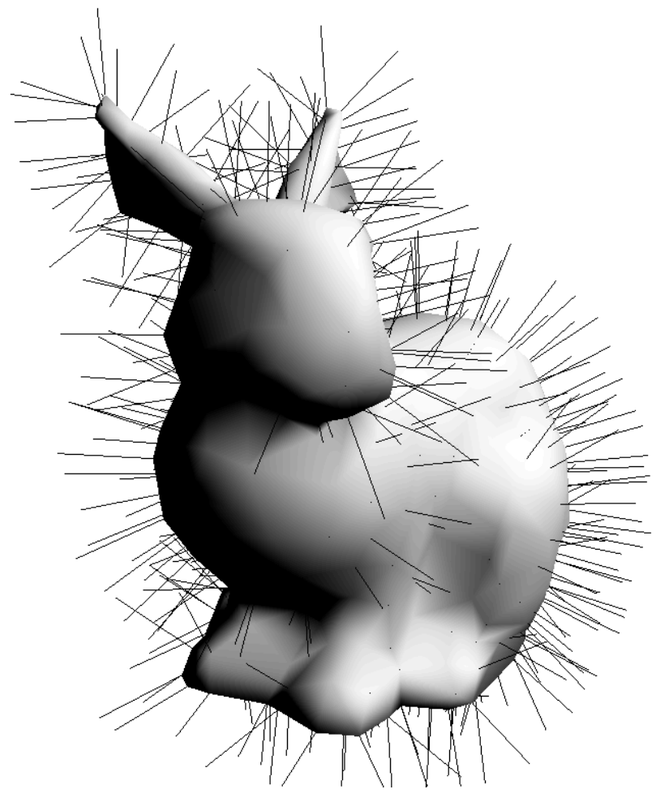
```
layout(triangles) in;
//layout(triangle_strip, max_vertices
      = 90) out; // Normal, solid
layout(line_strip, max_vertices = 90)
      out; // Wireframe
```



Hår och gräs

Använd geometrin som startpunkter, generera linjer/kurvor från dessa

```
for(float u = 0.0; u < 1.0; u += offs)
for(float v = 0.0; u+v < 1.0; v += offs)
{
   float u0 = u;
   float u1 = u + offs;
   float v0 = v;
   float v1 = v + offs;
   emit(u0,v0);
   EndPrimitive();
}
```

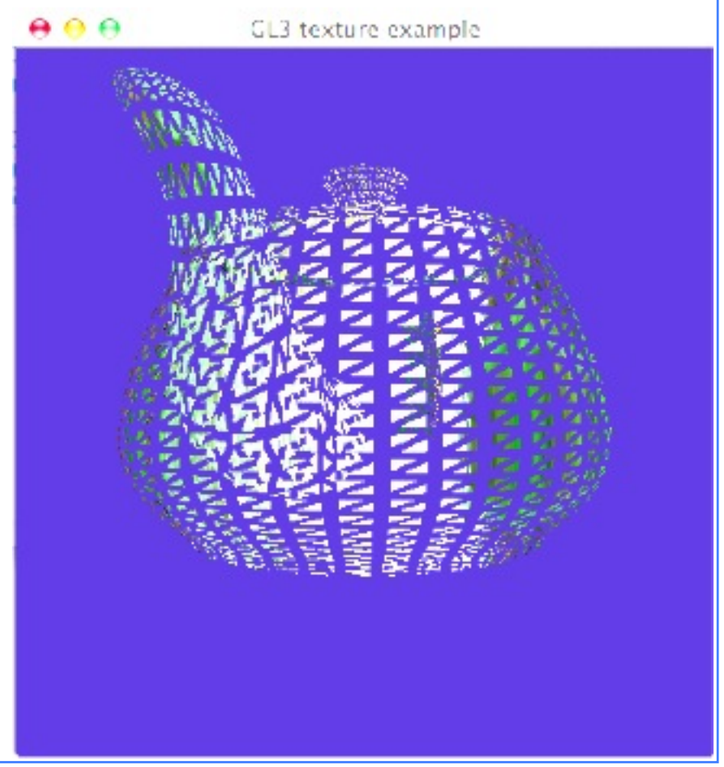


(Figuren visar enbart normalvektorer)

Crack shader

Mindre generellt användbart men rolig effekt. Flytta alla vertexar närmare centrum av triangeln.

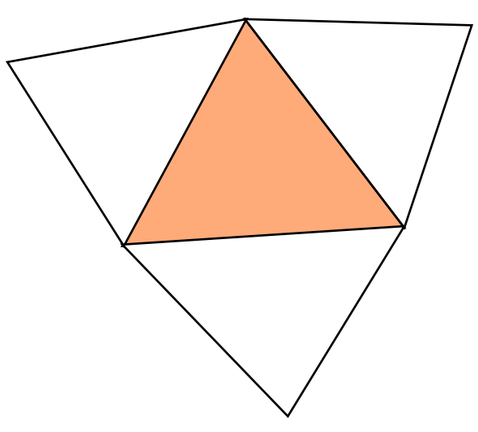
```
void main()
{
  vec4 middleOfTriangle = vec4(0.0);
  float tw = 1.0 / offs;

  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();
}
```



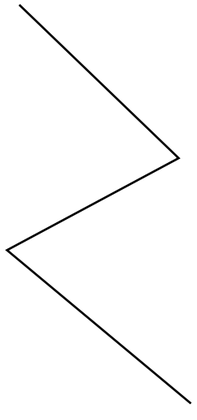
Adjacency

Geometri kan skickas tillsammans med information om grannpunkter

Tillämpningar: Shadow volumes, konturextraktion, tesselering



Triangel med adjacency



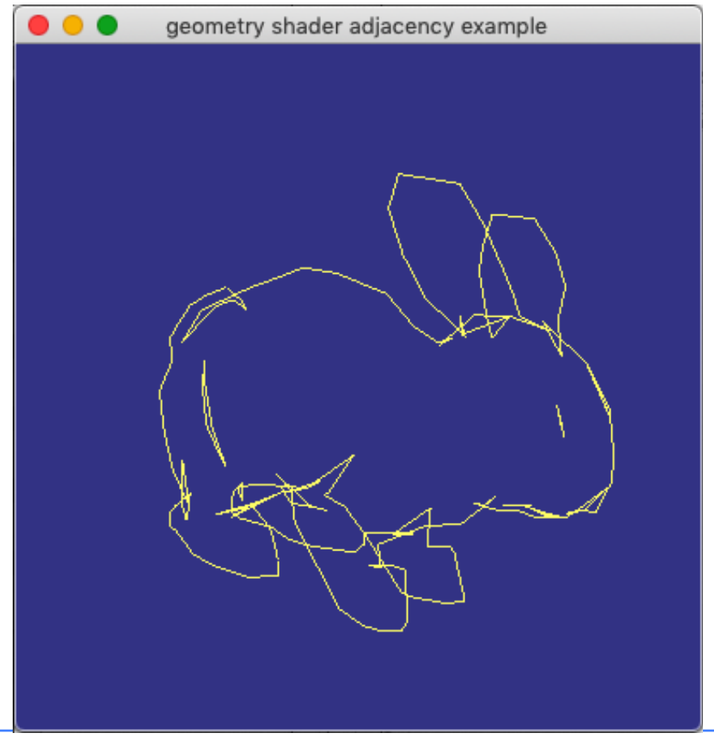
Linje med adjacency

Adjacency, exempel

Skapar adjacency genom att generera ny indexlista. Hittar kanter från grannpolygoners riktning (normalvektor).

```
vec3 a = vec3(gl_in[0].gl_Position - gl_in[2].gl_Position);
vec3 b = vec3(gl_in[0].gl_Position - gl_in[4].gl_Position);
vec3 nc = -cross(a, b);
float c = dot(nc, vec3(gl_in[0].gl_Position));

a = vec3(gl_in[0].gl_Position - gl_in[1].gl_Position);
b = vec3(gl_in[0].gl_Position - gl_in[2].gl_Position);
vec3 n1 = cross(a, b);
float c1 = dot(n1, vec3(gl_in[0].gl_Position));

if (c1*c > 0)
{
    gl_Position = projMatrix * gl_in[0].gl_Position;
    exNormalG = exNormal[0];
    EmitVertex();
    gl_Position = projMatrix * gl_in[2].gl_Position;
    exNormalG = exNormal[2];
    EmitVertex();
}
EndPrimitive();
```

Samma för övriga två kanter och grannar



Tesselering då...

Dela upp geometri för att

1) runda till en grov polygonmodell

2) lägga till detalj (displacement mapping mm)

3) Hantera level-of-detail

Baseras ofta på splines

Möjligt i både geometry och tesselation shaders

Displacement mapping

nu på geometrin, inte bump mapping!

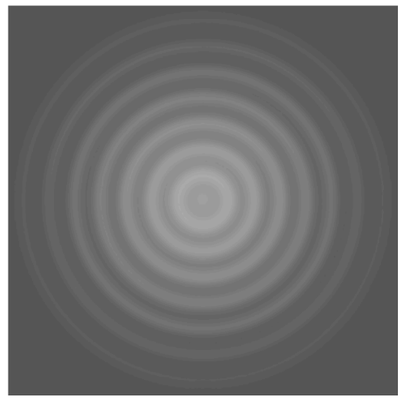


Figure 4: *Texture used as a displacement map.*

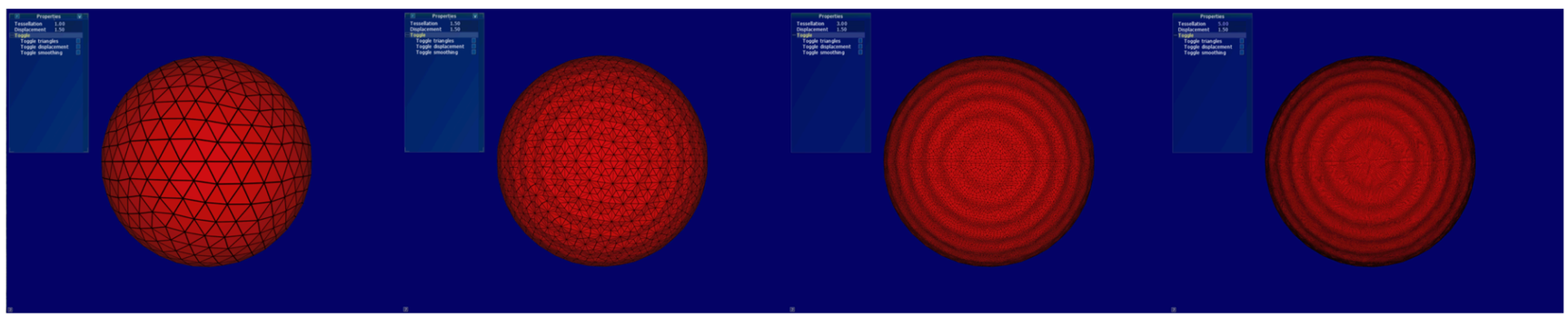
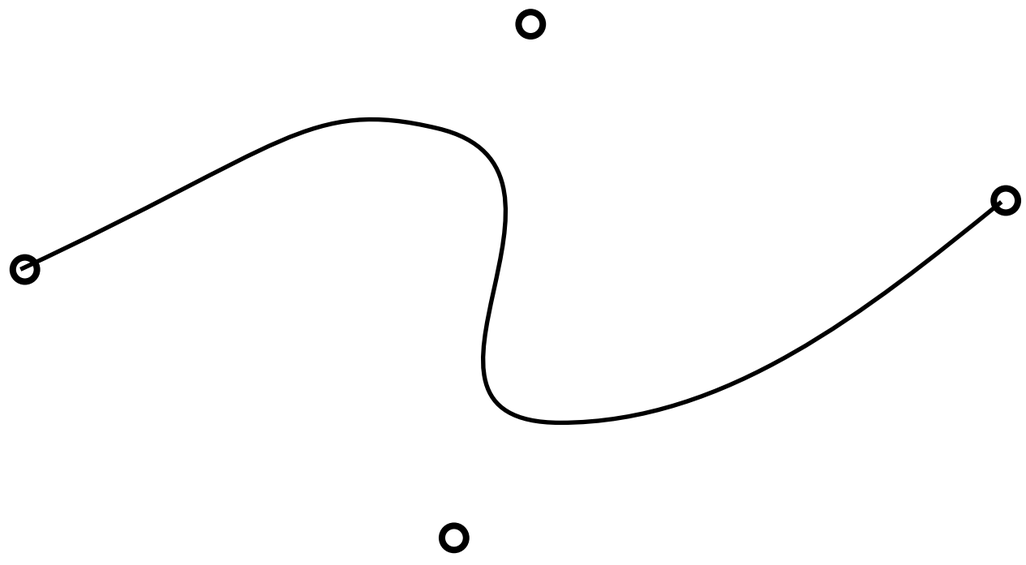


Figure 8: *The displacement map from Figure 4 applied to a sphere. From left to right: Sphere tessellated with a scaling factor of 1.0, 1.5, 3.0 and 5.0, and displaced with $C = 1.5$.*

Bézierkurvor

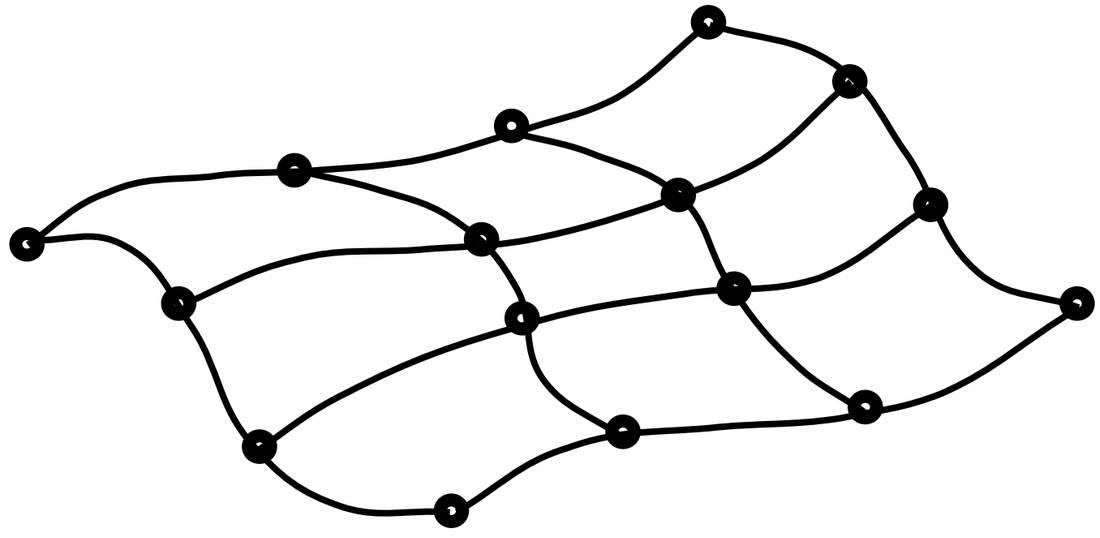
Typiskt 3 eller 4 kontrollpunkter per sektion



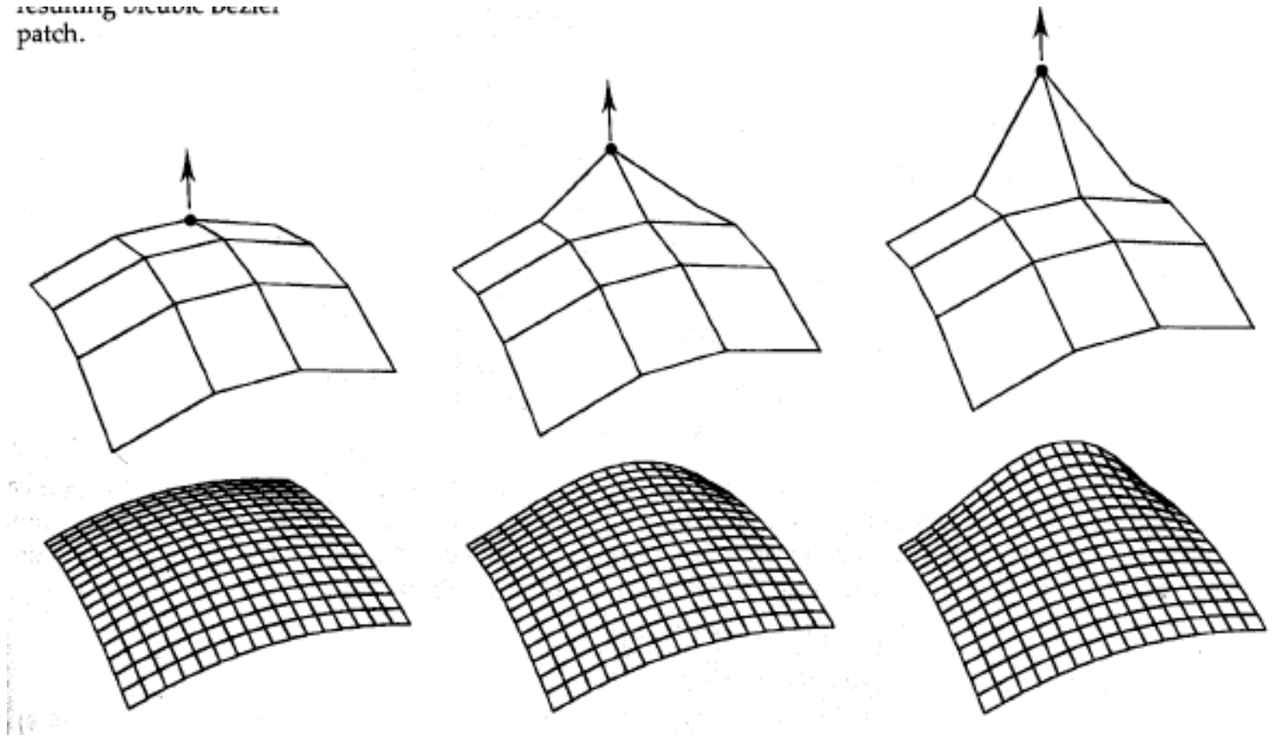
Bézierytor

En yta byggd av en uppsättning Bézier-ytsektioner (Bézier patches)

En Bézierpatch består av 16 kontrollpunkter i ett 4x4-rutnät



Exempel på Bézierytor



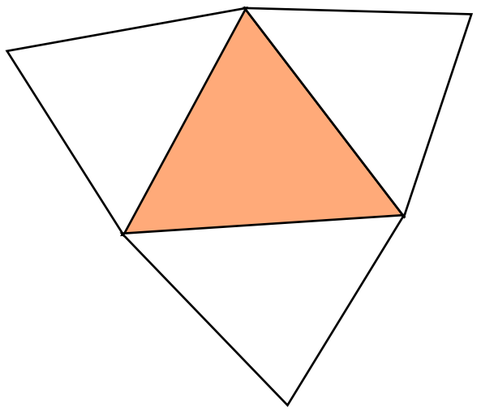
Bezierkurva i en geometry shader

```
// Simple 3-point spline geometry shader

#version 150

layout(triangles) in;
layout(line_strip, max_vertices = 50) out;

// quadratic bezier 0..1
vec4 bez3(vec3 a, vec3 b, vec3 c, float u)
{
   float aw = (1-u)*(1-u);
   float bw = 2*(1-u)*u;
   float cw = u*u;
   return vec4(a*aw+b*bw+c*cw, 1.0);
}
```

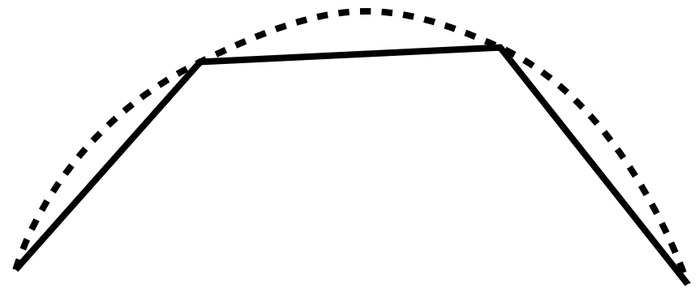
```
void main()
{
   for (int i = 0; i <= 20; i++)
   {
      gl_Position = bez3(
         vec3(gl_in[0].gl_Position),
         vec3(gl_in[1].gl_Position),
         vec3(gl_in[2].gl_Position),
         float(i)/20.0);
      EmitVertex();
   }
}
```

Adjacency

Kan underlätta tesselering genom att ge mer information om formen



Triangel med adjacency



Bezierpatches (4x4 punkter) med geometry shader?

Passar dåligt!

Geometry shader arbetar med trianglar.

Adjacency räcker inte till, och kräver mycket extraarbete på modeller för att använda.

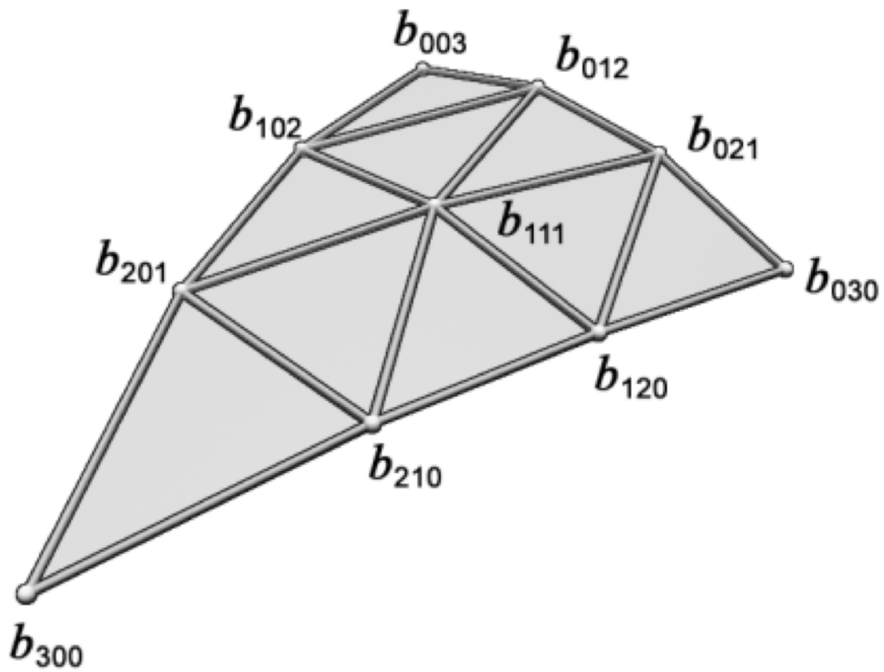
Curved PN Triangles

Passar bra för geometry shaders

Behöver bara vertexar samt normalvektorer för en triangel i taget - ej adjacency. Bekvämt.

PN = Point + Normal

Curved PN Triangles



Instancing av geometry shaders

Prestandaproblem vid få element in och många ut.

Instancing kör g.s. flera gånger på samma primitiv!

layout(invocations = num_instances) in;

gl_InvocationID

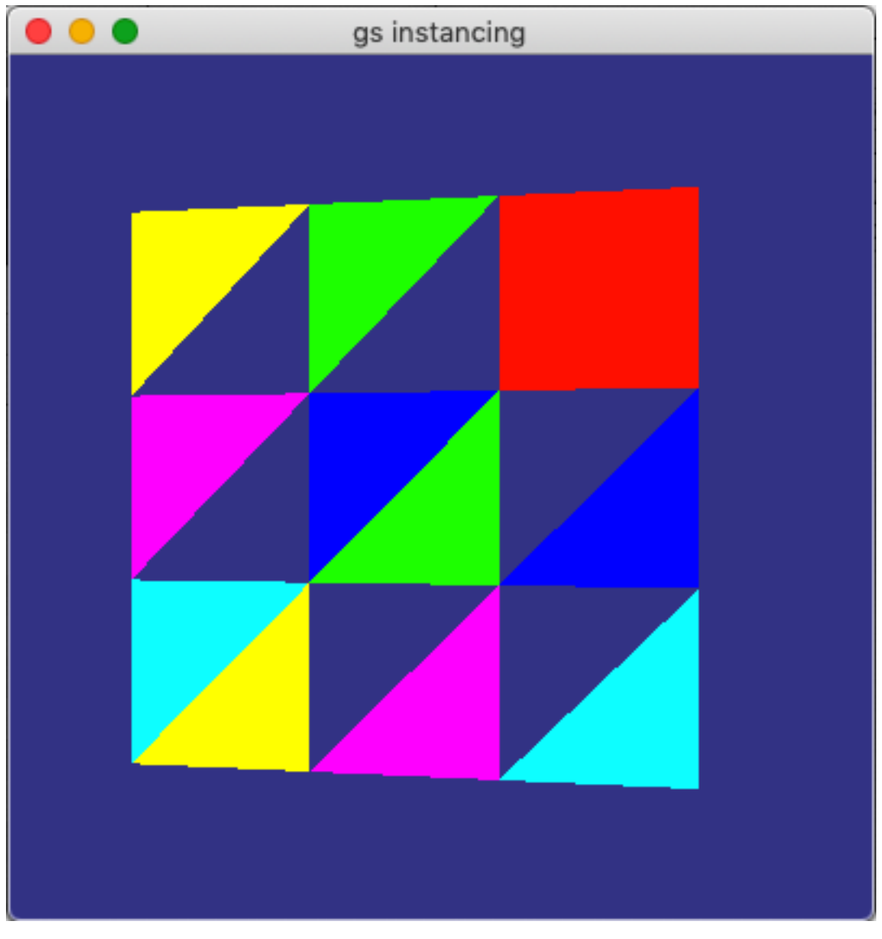
Minst 32 instanser garanterade!

Standard från OpenGL 4.0!

Exempel på instancing i geometry shader

Använder olika färg beroende på instans

```
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:
```



Slutsatser

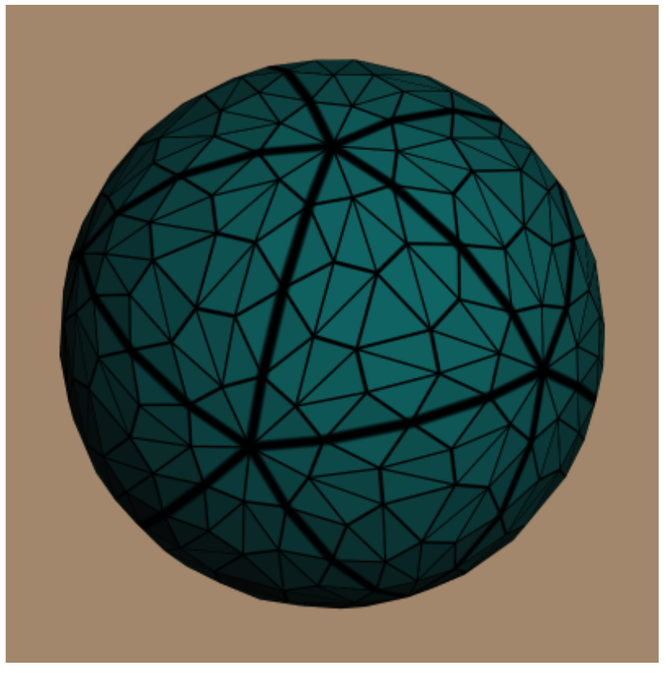
Geometry shader *mycket* användbar, med många tillämpningar.

Ett extra shadersteg krånglar inte till det så farligt.

Rekommenderas för kreativa idéer, kul projekt!

Räcker till det mesta - men inte allt.

Tesselation shaders



Tesselation shaders

Problem med tidiga geometry shaders:
Mycket geometri från få indata ineffektivt, seriellt.

Åtgärd 1: Instancing i nyare hårdvara, snabbare geometry shaders

Åtgärd 2: Tesselation shaders

Tesselation shaders

Ligger mellan vertex shader och geometry shader

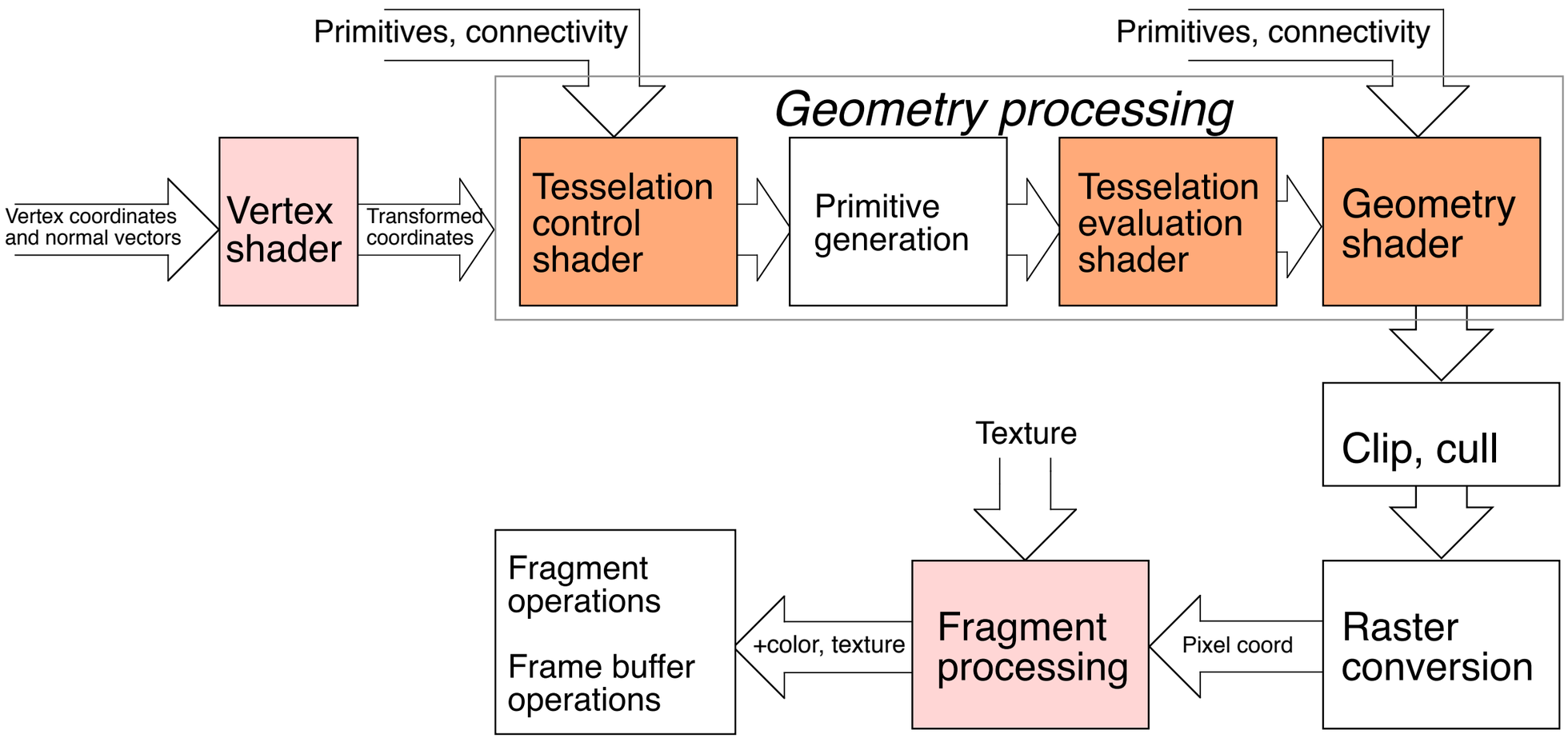
Inte en shader utan två, plus mellansteg.

1) Tesselation control shader

2) Primitive generation

3) Tesselation evaluation shader

OpenGL pipeline - extended



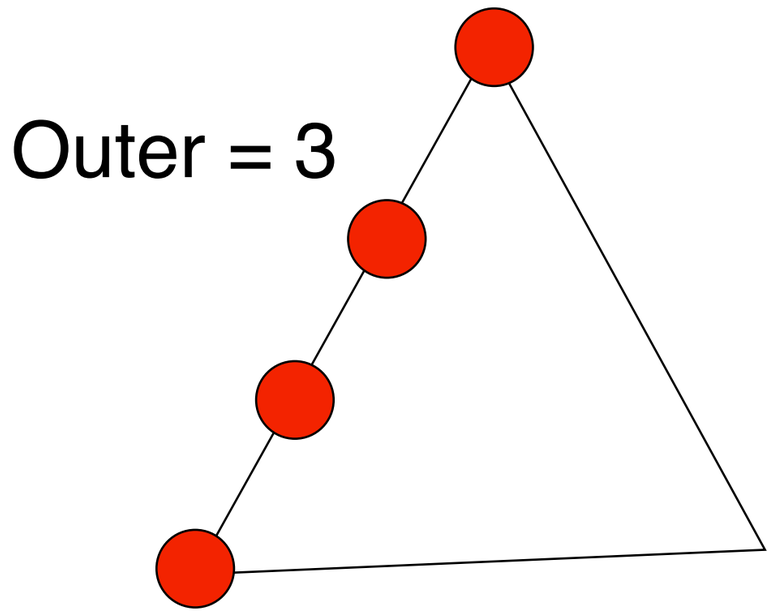
Uppgifter hos tesselation shaders

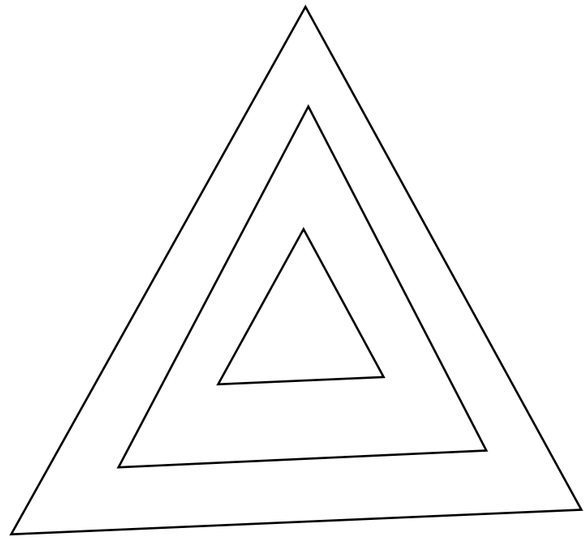
Indata: ”Patchar”, ett antal vertexar utan given konstellation.

TC-shadern anger önskad detaljnivå per kant samt inre nivåer

TE-shadern beräknar slutliga positioner på önskat sätt (t.ex. med en splinefunktion)

Exempel: Triangel





Inner = 3

Exempel: 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;
    }
}
```

Exempel: 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
}
```

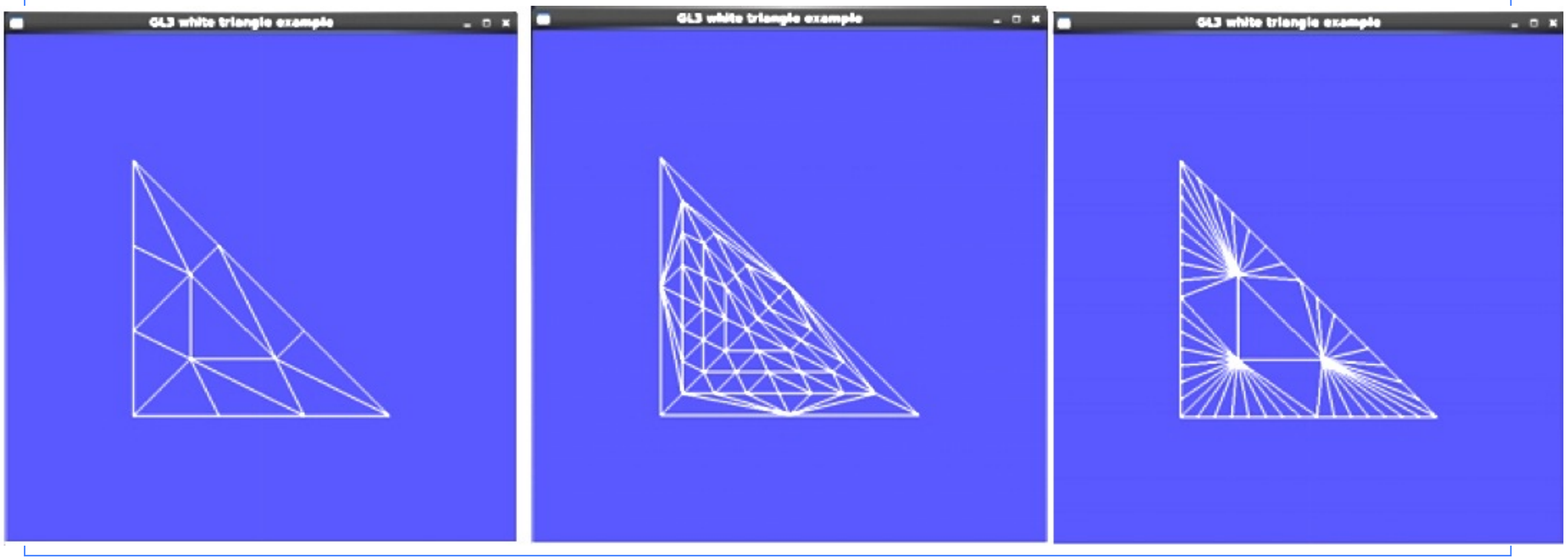
Control + Evaluation

Control avgör hur många stegs tesselering som är önskvärd

Evaluation anropas så många gånger, får unika koordinater, från vilka vi skall beräkna resulterande vertexar

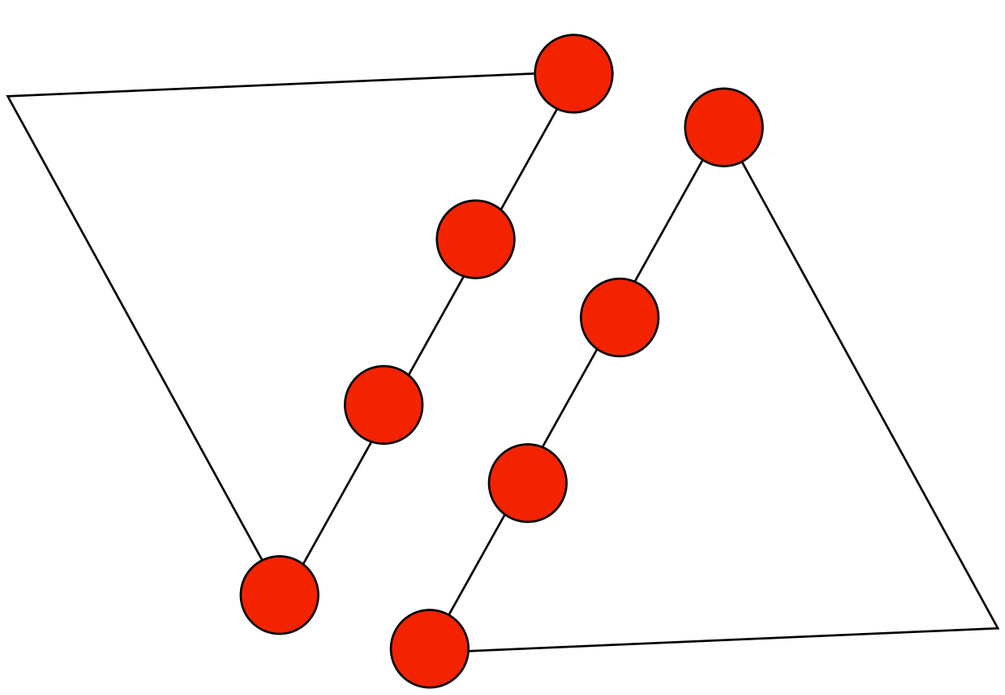
Resultat

Variation på outer och inner



”Outer” görs per kant

Viktigt!



Beräkning per kant
ger samma täthet
på gemensam kant
mellan olika delar!

Inga glipor!

Tesselering är inget nytt!

Gamla OpenGL hade ”Evaluators” (glMap)

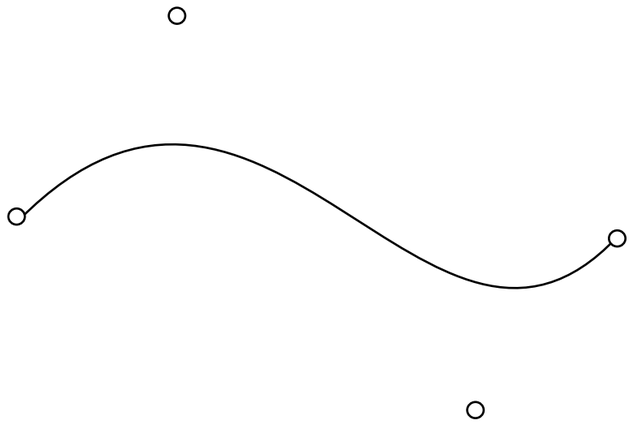
Bra stöd för Bezierkurvor och Bezierytor.

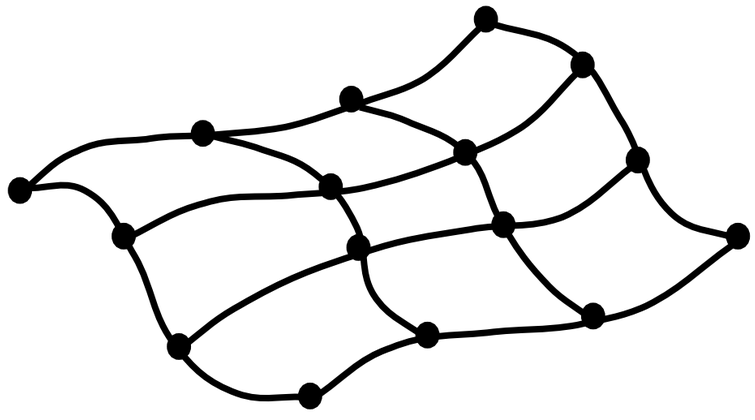
Dock begränsad kontroll från APIet

Evaluators

Gammalt API för att evaluera Bezierkurvor. Lättanvänt men numera deprecated.

Stödjer både kurvor och ytor





Evaluators

Rättframt men inte så flexibelt

```
glMap1f(GL_MAP1_VERTEX_3, u0, u1, 3, 4, &data2[0][0]);
glEnable(GL_MAP1_VERTEX_3);
glBegin(GL_LINE_STRIP);
for (int i = 0; i <= 20; i++)
glEvalCoord1f(u0 + i*(u1-u0)/20);
glEnd();
```

Control points

Evaluation, specifies vertices

Evaluators

Gammalmodiga! Enbart quads - knepigt att använda på generella modeller.

=> satsa på geometry shaders och tesselation shaders!