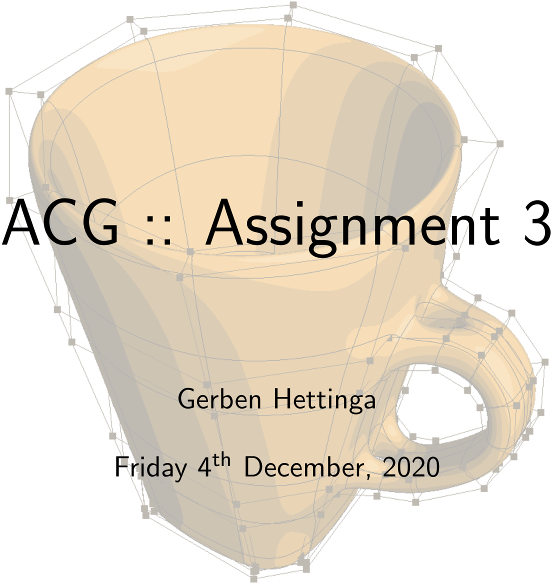

# ACG :: Assignment 3

Gerben Hettinga

Friday 4th December, 2020

# Previous Assignment

- Questions, discussion, ...

# Assignment

MA Project vertices (at any subdivision level) to their limit positions using limit stencils. Boundaries should be supported!

# Assignment

MA Project vertices (at any subdivision level) to their limit positions using limit stencils. Boundaries should be supported!

MA Render surface patches corresponding to the regular quads in the control net

- Use tessellation shaders (TCS and TES)

# Assignment

- MA Project vertices (at any subdivision level) to their limit positions using limit stencils. Boundaries should be supported!
- MA Render surface patches corresponding to the regular quads in the control net
  - Use tessellation shaders (TCS and TES)
- AF Use the true normals of the surface patches in your fragment shader

# Assignment

MA Project vertices (at any subdivision level) to their limit positions using limit stencils. Boundaries should be supported!

MA Render surface patches corresponding to the regular quads in the control net

- Use tessellation shaders (TCS and TES)

AF Use the true normals of the surface patches in your fragment shader

B Extend the GUI with a user-friendly way to control the tessellation levels

# Rendering of quads

- Render *n*-gons using glEnable(GL_PRIMITIVE_RESTART), glPrimitiveRestartIndex(maxInt) and glDrawElements with GL_TRIANGLE_FAN. The index maxInt should be inserted in the IBO after every *n* indices defining an *n*-gon.

# Rendering of quads

- Render *n*-gons using glEnable(GL_PRIMITIVE_RESTART), glPrimitiveRestartIndex(maxInt) and glDrawElements with GL_TRIANGLE_FAN. The index maxInt should be inserted in the IBO after every *n* indices defining an *n*-gon.

# Limit stencils

- Quads:

  See Equation 13 in *Mark Halstead, Michael Kass, and Tony DeRose. "Efficient, fair interpolation using Catmull-Clark surfaces." Proceedings of the 20th annual conference on Computer graphics and interactive techniques. ACM, 1993.*

# Limit stencils

- Quads:

  See Equation 13 in *Mark Halstead, Michael Kass, and Tony DeRose. "Efficient, fair interpolation using Catmull-Clark surfaces." Proceedings of the 20th annual conference on Computer graphics and interactive techniques. ACM, 1993.*

- N-gons (use this one!):

  See the expression for $p_0$ in Section 3.2 in *Charles Loop, Scott Schaefer, Tianyun Ni, and Ignacio Castao. "Approximating subdivision surfaces with Gregory patches for hardware tessellation." In ACM Transactions on Graphics (TOG). ACM, 2009.*

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}
- An input patch can contain coordinates $(x, y, z, w)$, derivatives, normals, ...

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}
- An input patch can contain coordinates $(x, y, z, w)$, derivatives, normals, ...
- The TCS is invoked for each vertex in the output patch. Use the built-in variable gl_InvocationID

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}
- An input patch can contain coordinates $(x, y, z, w)$, derivatives, normals, ...
- The TCS is invoked for each vertex in the output patch. Use the built-in variable gl_InvocationID
- Outputs of the TCS are either per-vertex or per-patch (...)

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}
- An input patch can contain coordinates $(x, y, z, w)$, derivatives, normals, ...
- The TCS is invoked for each vertex in the output patch. Use the built-in variable gl_InvocationID
- Outputs of the TCS are either per-vertex or per-patch (...)
- TCS layout example: layout (vertices = 9) out

# Tessellation shaders (TCS and TES) (1)

- Tessellation of isolines (curves), triangles and quads (surface patches)
- Use the special primitive GL_PATCH in combination with glDraw{Arrays,Elements}
- An input patch can contain coordinates $(x, y, z, w)$, derivatives, normals, ...
- The TCS is invoked for each vertex in the output patch. Use the built-in variable gl_InvocationID
- Outputs of the TCS are either per-vertex or per-patch (...)
- TCS layout example: layout (vertices = 9) out
- Individual tessellation levels can be set in the TCS (or in the OpenGL code)

# Tessellation Control Shader (TCS)

- After the Vertex shader.

# Tessellation Control Shader (TCS)

- After the Vertex shader.
- On the CPU side set the number of input vertex using `glPatchParameteri(GL_PATCH_VERTICES, 4)`

# Tessellation Control Shader (TCS)

- After the Vertex shader.
- On the CPU side set the number of input vertex using `glPatchParameteri(GL_PATCH_VERTICES, 4)`
  - This is also the number of invocations of the TCS

```
#version 400 core
in vec3[] pos;

layout(vertices = 4) out;
out vec3[] tc_p;

void main()
{
    // set inner outer tess level
    if (gl_InvocationID == 0){
        //do something to set the tessellation levels
    }

    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;

    tc_p[gl_InvocationID] = pos[gl_InvocationID];
}
```

# Tessellation Evaluation Shader (2)

- The Tessellation Module tessellates isolines, triangles or quads using the defined tessellation levels (TCS) and parameters (TES)

# Tessellation Evaluation Shader (2)

- The Tessellation Module tessellates isolines, triangles or quads using the defined tessellation levels (TCS) and parameters (TES)
- The TES receives the barycentric or bilinear coordinates of points in your patch, gl_TessCoord

# Tessellation Evaluation Shader (2)

- The Tessellation Module tessellates isolines, triangles or quads using the defined tessellation levels (TCS) and parameters (TES)
- The TES receives the barycentric or bilinear coordinates of points in your patch, gl_TessCoord
- Inputs of the TES are either per-vertex or per-patch

# Tessellation Evaluation Shader (2)

- The Tessellation Module tessellates isolines, triangles or quads using the defined tessellation levels (TCS) and parameters (TES)
- The TES receives the barycentric or bilinear coordinates of points in your patch, gl_TessCoord
- Inputs of the TES are either per-vertex or per-patch
- TES layout example: layout (quads, equal_spacing, ccw) in (latter two are optional and default values)

# Resources) (2)

```
#version 400 core
layout(quads, fractional_odd_spacing, ccw) in;
in vec3[] tc_p;

out vec3 position;

vec3 calcPos(float u, float v) {
  return (1.0 - u)*(1.0 - v)  * tc_p[0] +
         u*(1.0 - v)          * tc_p[1] +
         u*v                  * tc_p[2] +
         (1.0 - u)*v          * tc_p[3];
}

void main() {
  position = calcPos(gl_TessCoord[0], gl_TessCoord[1]);
}
```

Good on-line sources: https://www.opengl.org/wiki/Tessellation, http://prideout.net/blog/?tag=tessellation-shaders, http://in2gpu.com/2014/07/12/tessellation-tutorial-opengl-4-3/, http://ogldev.atspace.co.uk/www/tutorial30/tutorial30.html

# Tessellation workflow

1. Tessellation Control Shader:
   1. set tessellation level
   2. compute additional control points/structure data
   3. pass data to TES
2. Tessellation Evaluation Shader
   1. use data from TCS (aggragated in arrays)
   2. use `glTessCoords` to compute a point on your patch

# Tensor product patches

Two ways to render the regular regions of the mesh

- Tensor product uniform cubic B-spline
- Tensor product Bézier patch
  - Have to convert control net to control points

# Cubic B-spline to Bézier Conversion

Tensor Product Cubic B-spline surface:

$$S(u, v) = UMGM^T V^T$$

where $U = [u^3, u^2, u, 1]$, $V = [v^3, v^2, v, 1]$, $M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$, and

$G$ is the 4x4 grid of control vertices

# Cubic B-spline to Bézier Conversion Stencils

Apply masks to the 4x4 Grid of the control net to get the control points for the Bézier patches.

- vertices = limit positions
- edge control points = $\begin{bmatrix} 2 & 1 \\ 8 & 4 \\ 2 & 1 \end{bmatrix}$ /18, flip and rotate for other points
- mid control points = $\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$ /9, rotate for other mid control points

This process can be done in the tessellation control shader

# Uniform Basis Functions

| | B-spline | Bezier |
|---|---|---|
| $B_0(u)$ | $(1-u)^3/6$ | $(1-u)^3$ |
| $B_1(u)$ | $4-6u^2+3u^3/6$ | $3(1-u)^2u$ |
| $B_2(u)$ | $1+3u+3u^2-3u^3/6$ | $3(1-u)u^2$ |
| $B_3(u)$ | $u^3/6$ | $u^3$ |

To create the tensor product basis functions from two variables $u$ and $v$:

$$\begin{pmatrix} B_0(u)B_3(v) & B_1(u)B_3(v) & B_2(u)B_3(v) & B_3(u)B_3(v) \\ B_0(u)B_2(v) & B_1(u)B_2(v) & B_2(u)B_2(v) & B_3(u)B_2(v) \\ B_0(u)B_1(v) & B_1(u)B_1(v) & B_2(u)B_1(v) & B_3(u)B_1(v) \\ B_0(u)B_0(v) & B_1(u)B_0(v) & B_2(u)B_0(v) & B_3(u)B_0(v) \end{pmatrix}$$

# To conclude...

- Next two practicals
  - I'll be available on Collaborate for questions

# To conclude...

- Next two practicals
  - I'll be available on Collaborate for questions
- See Time Schedule on Nestor for the remaining deliverables and deadlines (summary, slides, draft report, code, report).

# To conclude...

- Next two practicals
  - I'll be available on Collaborate for questions
- See Time Schedule on Nestor for the remaining deliverables and deadlines (summary, slides, draft report, code, report).
- We're always looking for TAs for Computer Graphics (Feb - March 2020), assignments are on Raytracing and OpenGL