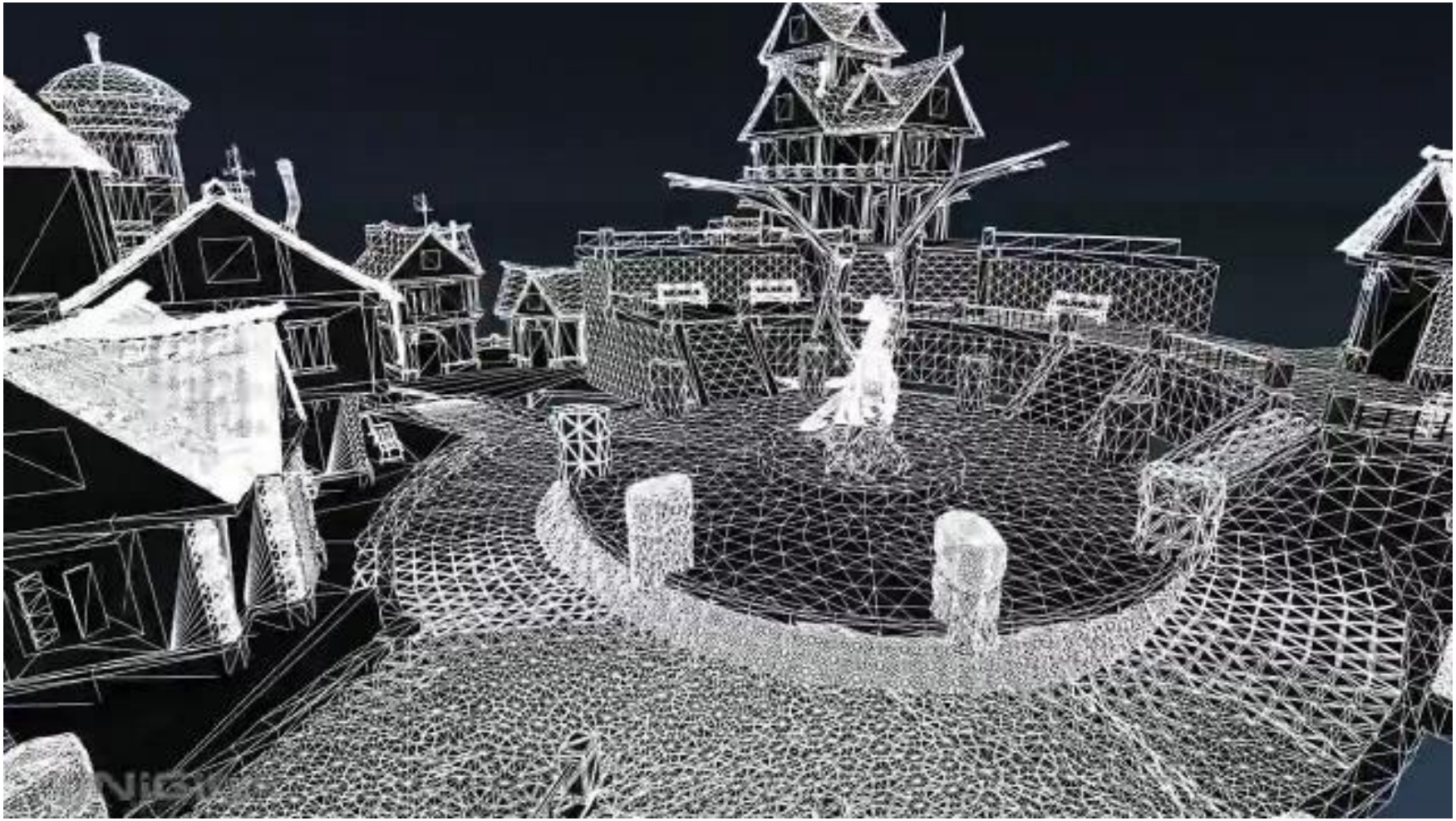
Hardware-Tessellation

Klemens Jahrmann

Institute of Computer Graphics and Algorithms

Vienna University of Technology

Tessellation



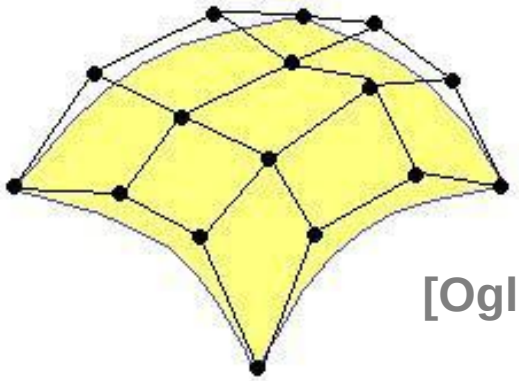
[Unigine Heaven Benchmark]

Hardware-Tessellation

- Subdivision of polygons or lines
- Available as new shader stages since OpenGL 4.0 core / DirectX 11
- Tessellation shader are faster than geometry shader
- Smooth meshes even with non-uniform subdivision levels

Typical Usage of Tessellation

- Dynamic Level of Detail (LOD)
  - Adjust tessellation level according to camera distance or screen size of the polygon
- Rendering of algebraic surfaces / curves
  - Send only control points to the GPU and evaluate the surface / curve on the fly (e.g. Bezier-Patch, Bezier-Curve)



[OglDev]

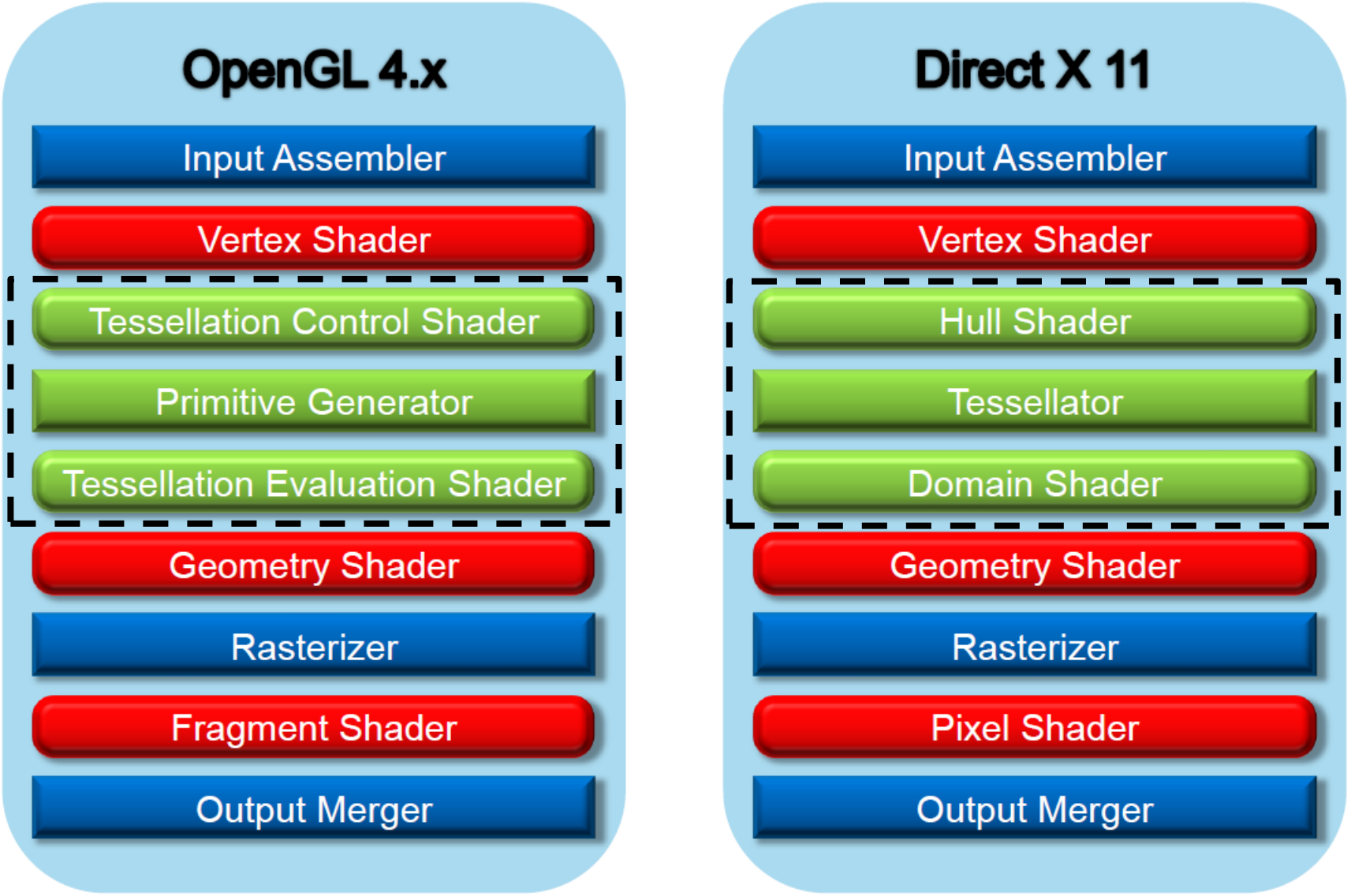
Typical Usage of Tessellation

- Displacement mapping
  - Changes the objects silhouette not just the lighting

vs

[Unigine Heaven Benchmark]

Tessellation shader stages



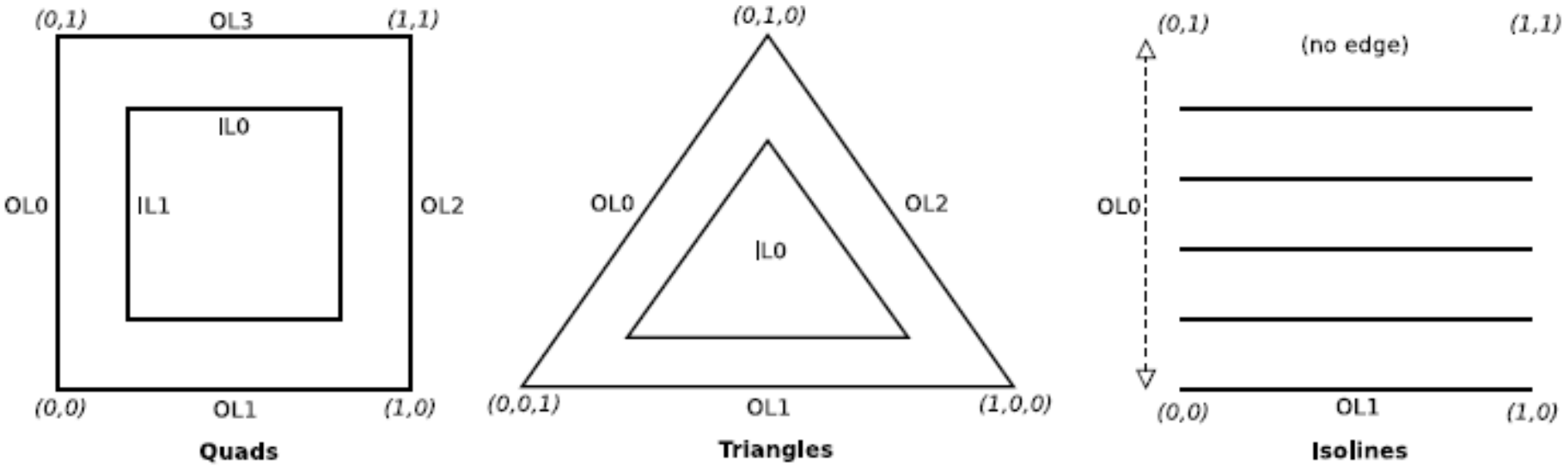
Tessellation Control Shader

- Defines how often a polygon / line is subdivided
  - `gl_TessLevelOuter[0-3]`
  - `gl_TessLevelInner[0-1]`
- Executed for each vertex from the vertex shader
- Access to all patch vertices

Primitive Generator

- Subdivides the polygon / line and outputs abstract patch coordinates
  - based on the tessellation levels specified in the Tessellation Control Shader
- Type of subdivision is specified in the Tessellation Evaluation Shader
  - Possible types:
    - triangles
    - quads
    - isolines

Primitive Generator Subdivision



Abstract output coordinates 2D for quads and isolines (u,v) and 3D for triangles (u,v,w)
OL = outer tessellation level
IL = inner tessellation level

- **Important:** The Primitive Generator subdivides only an abstract patch, which does not correspond to the input vertices!

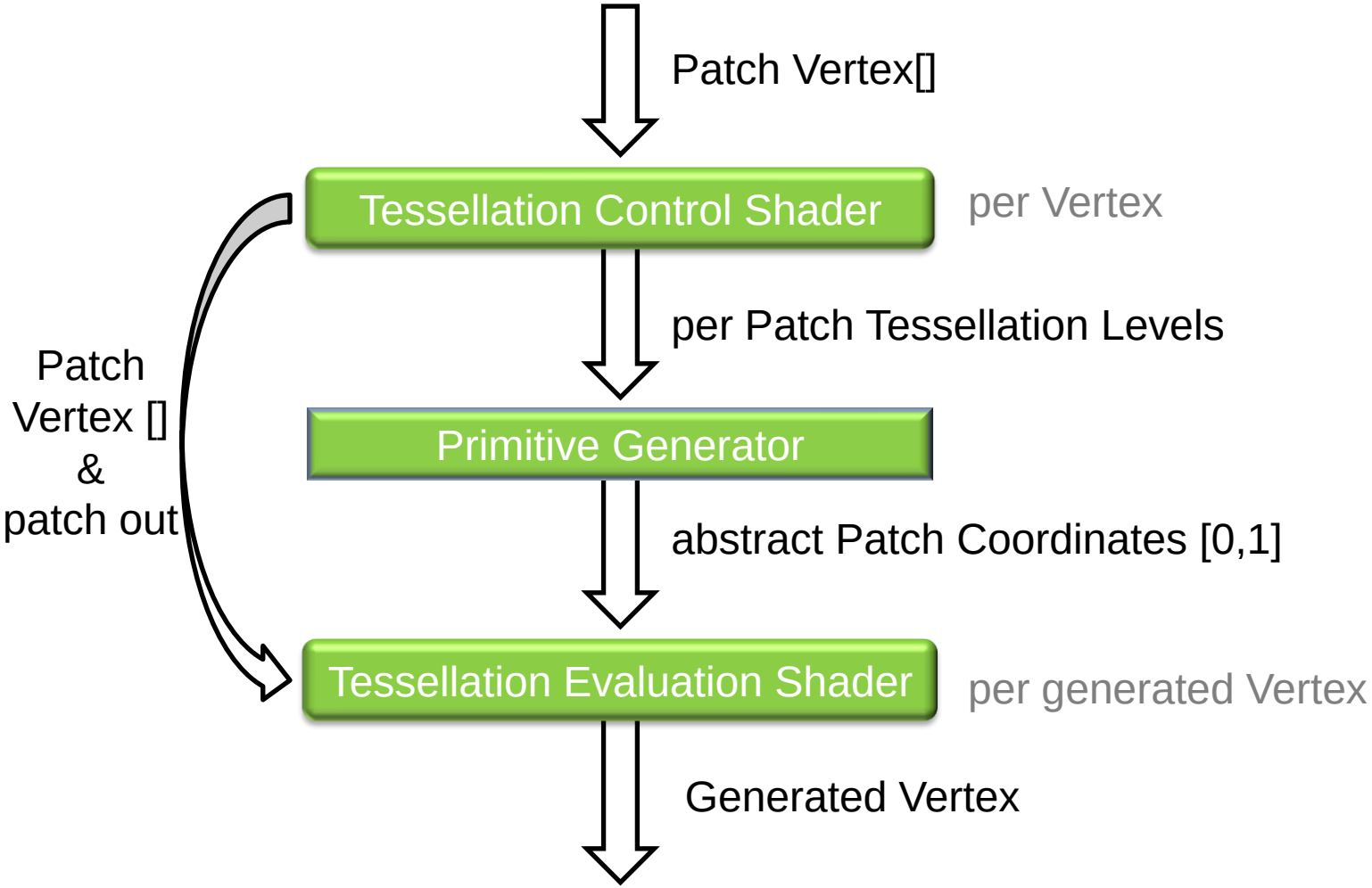
Tessellation Evaluation Shader

- Specifies the type of the tessellation
  - `layout(`*`domain`*`, `*`spacing`*`, `*`winding`*`) in;`
    - where
      - *`domain`* can be: `triangles`, `quads`, `isolines`
      - *`spacing`* can be: `equal_spacing`, `fractional_even_spacing`, `fractional_odd_spacing`
      - *`winding`* can be: `cw`, `ccw`

Tessellation Evaluation Shader

- Executed for each vertex generated from the Primitive Generator
- Input:
  - Abstract patch coordinates
  - Vertex[] from Tessellation Control Shader
  - patch out variables from Tessellation Control Shader
- Output:
  - One vertex per invocation

Tessellation data flow



Example: Displacement Mapping

[ethereal3d.com]

Application Code: Shader

- Create tessellation shader (basically same procedure as for other shader):
  - `glCreateShader(GL_TESS_CONTROL_SHADER);`
  - `glCreateShader(GL_TESS_EVALUATION_SHADER);`
- Attach them to the shader programm:
  - `glAttachShader(prog, tess_control_shader);`
  - `glAttachShader(prog, tess_evaluation_shader);`

Application Code: Draw

- Use primitive mode **GL_PATCHES** for rendering:

```
glDrawArrays(GL_PATCHES, first_vert, count);
glDrawElements(GL_PATCHES, idx_length, GL_UNSIGNED_INT, 0);
```

- Configure number of vertices in patch

```
glPatchParameteri(GL_PATCH_VERTICES, n)
```

- Query for the maximum allowable numer **n**

```
Glint MaxPatchVertices;
glGetIntegerv(GL_MAX_PATCH_VERTICES, &MaxPatchVertices);
```


Vertex Shader

```
#version 410 core

in vec4 in_Position_VS;  // attribute 0: object space vertex position
in vec3 in_Normal_VS;    // attribute 1: object space normal
in vec2 in_TextCoord_VS; // attribute 2: texture coordinate

// variables to pass down information from VS to TCS
out vec4 in_Position_CS;
out vec3 in_Normal_CS;
out vec2 in_TextCoord_CS;

void main(void) {
    in_Position_CS = in_Position_VS;
    in_Normal_CS = in_Normal_VS;
    in_TextCoord_CS = in_TextCoord_VS;
}
```


Tessellation Control Shader (1/2)

```
#version 410 core

// define the number of Vertices in the output patch
// (can be different from the input patch size)
layout (vertices = 3) out;

// attributes of the input Vertices (from Vertex Shader)
in vec4 in_Position_CS[];
in vec3 in_Normal_CS[];
in vec2 in_TextCoord_CS[];

// attributes of the output Vertices (to Tessellation Evaluation Shader)
out vec4 in_Position_ES[];
out vec3 in_Normal_ES[];
out vec2 in_TextCoord_ES[];
```

Tessellation Control Shader (2/2)

```
void main(void) {
    // Set the control points (vertices) of the output patch
    in_Position_ES[gl_InvocationID] = in_Position_CS[gl_InvocationID];
    in_Normal_ES[gl_InvocationID] = in_Normal_CS[gl_InvocationID];
    in_TextCoord_ES[gl_InvocationID] = in_TextCoord_CS[gl_InvocationID];

    // the next snippet just sketches the calculations...
    // based on the vertex distances to the camera, we choose the TLs
    // see [OglDev] for an example implementation

    // Calculate the tessellation levels
    if (gl_InvocationID == 0) {
        gl_TessLevelOuter[0] = calc_TL(in_Position_CS[1], in_Position_CS[2]);
        gl_TessLevelOuter[1] = calc_TL(in_Position_CS[2], in_Position_CS[0]);
        gl_TessLevelOuter[2] = calc_TL(in_Position_CS[0], in_Position_CS[1]);
        gl_TessLevelInner[0] = calc_inner_TL( ... );
    }
}
```


Tessellation Evaluation Shader (1/3)

```
#version 410 core

// tell PG to emit triangles in counter-clockwise order with equal spacing
layout(triangles, equal_spacing, ccw) in;

uniform mat4 mvpMatrix;

uniform sampler2D dispTexture; // texture for displacement values
uniform float displacement_factor;

// these vertex attributes are passed down from the TCS
in vec4 in_Position_ES[];
in vec3 in_Normal_ES[];
in vec2 in_TextCoord_ES[];

out vec2 in_TextCoord_FS;
```


Tessellation Evaluation Shader (2/3)

```
// Interpolate values v0-v2 based on the barycentric coordinates
// of the current vertex within the triangle
vec2 interpolate2D(vec2 v0, vec2 v1, vec2 v2) {
    return vec2(gl_TessCoord.x) * v0 +
           vec2(gl_TessCoord.y) * v1 +
           vec2(gl_TessCoord.z) * v2;
}

// Interpolate values v0-v2 based on the barycentric coordinates
// of the current vertex within the triangle
vec3 interpolate3D(vec3 v0, vec3 v1, vec3 v2) {
    return vec3(gl_TessCoord.x) * v0 +
           vec3(gl_TessCoord.y) * v1 +
           vec3(gl_TessCoord.z) * v2;
}
```

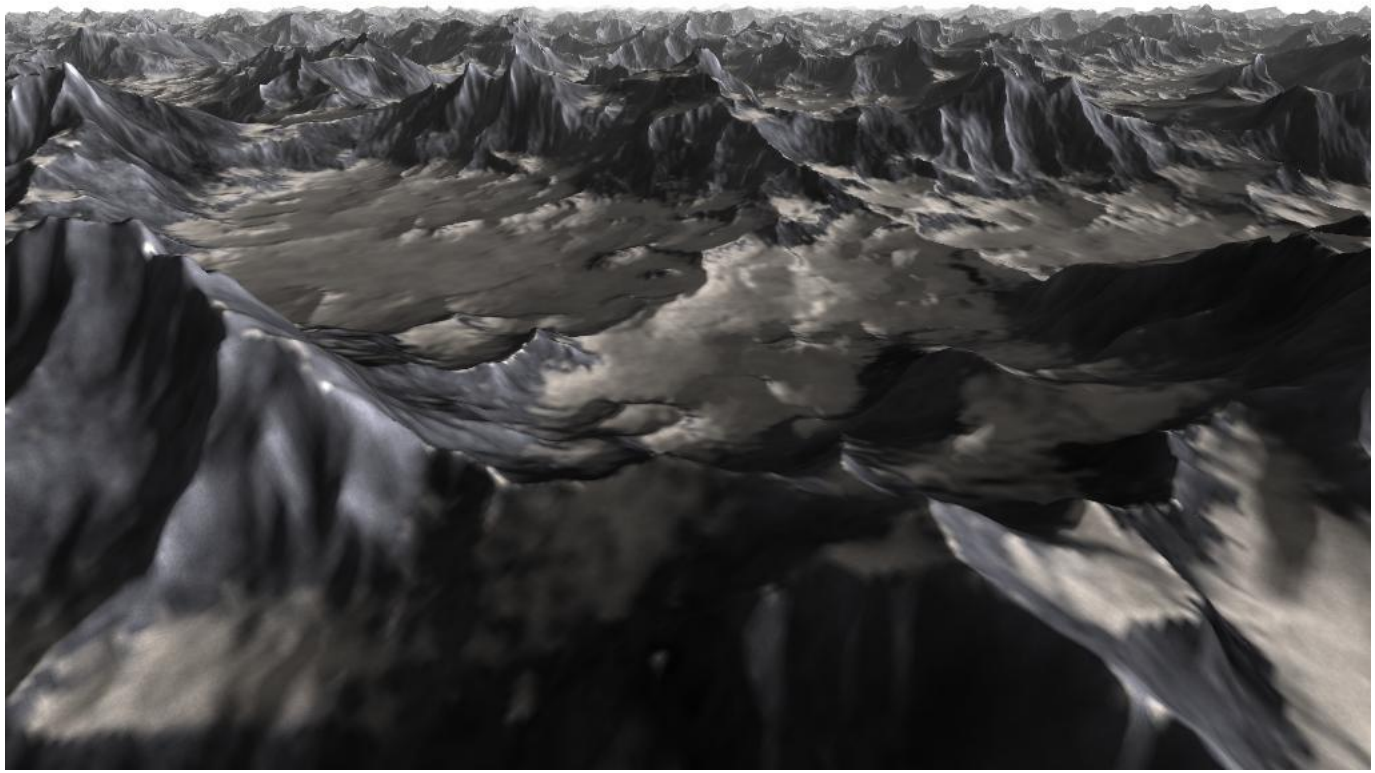

Tessellation Evaluation Shader (3/3)

```
void main(void) {
    // Interpolate attribs of output vertex using its barycentric coords
    vec4 position = vec4( interpolate3D( in_Position_ES[0].xyz,
         in_Position_ES[1].xyz, in_Position_ES[2].xyz ), 1.0);
    vec4 normal = normalize(vec4( interpolate3D( in_Normal_ES[0],
         in_Normal_ES[1], in_Normal_ES[2]), 0.0));
    vec2 textCoord = interpolate2D(in_TextCoord_ES[0], in_TextCoord_ES[1],
         in_TextCoord_ES[2]);

    // Displace the vertex along the normal
    float displacement = texture(dispTexture, textCoord).x;
    position += normal * displacement * displacement_factor;

    // transform to NDC
    gl_Position = mvpMatrix * position;
    in_TextCoord_FS = textCoord; // pass texture coordinate to FS
}
```

Fragment Shader

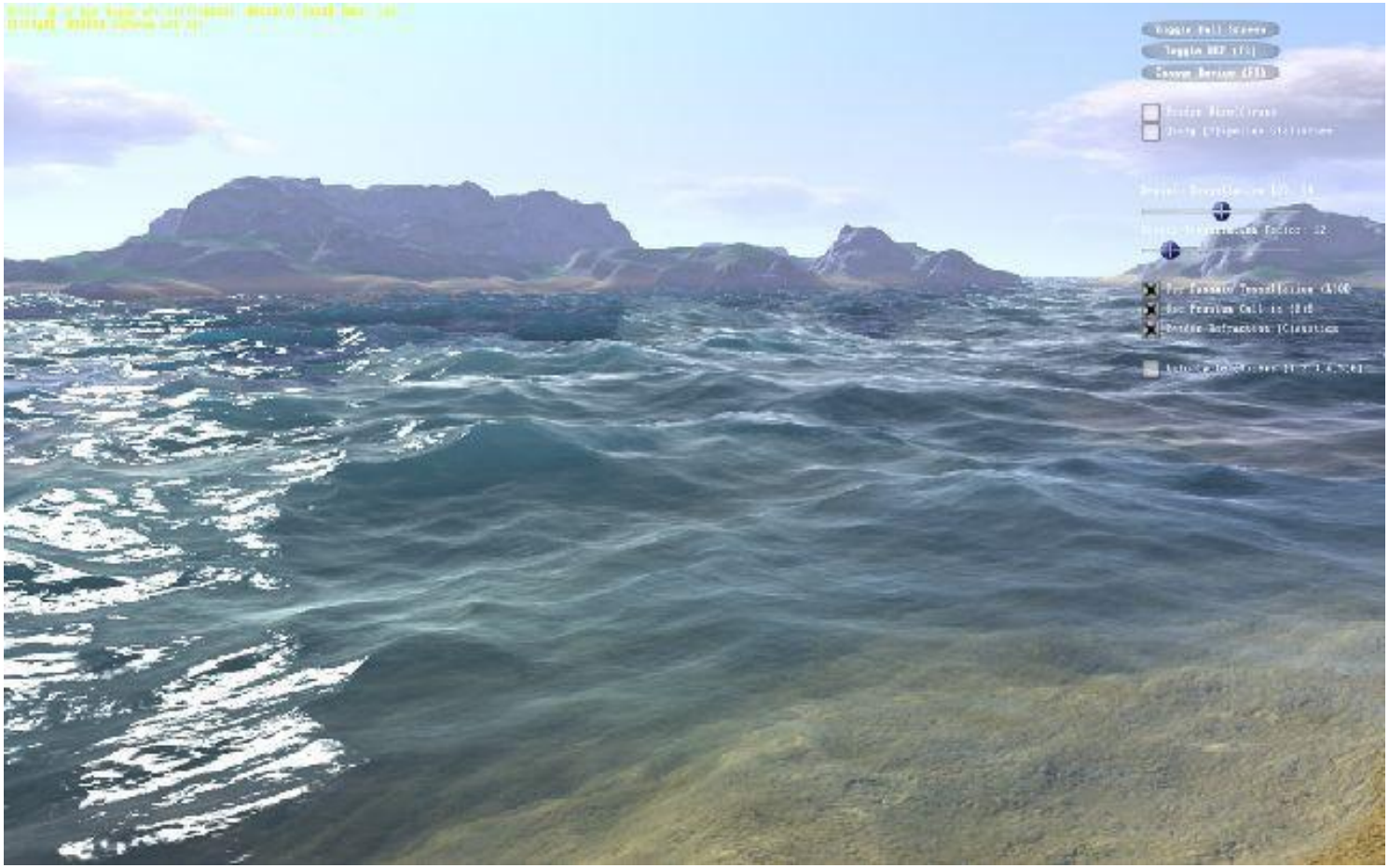
```
#version 410 core

uniform sampler2D colorTexture; // color texture

in vec2 in_TextCoord_FS; // passed down from the TES

out vec4 out_Color; // the final fragment color

void main(void) {
    out_Color = texture(colorTexture, in_TextCoord_FS);
}
```


Tutorials and Examples



http://codeflow.org/entries/2010/nov/07/opengl-4-tessellation

Tutorials and Examples



http://www.geforce.com/games-applications/pc-applications/fermi-water-demo

Tutorials and Examples

http://www.cg.tuwien.ac.at/research/publications/2013/JAHRMANN-2013-IGR/

References

- https://www.opengl.org/wiki/Tessellation

- Other Tutorials:
  - ogldev.atspace.co.uk/www/tutorial30/tutorial30.html
  - prideout.net/blog/?p=48
  - www.geeks3d.com/20100730/test-first-contact-with-opengl-4-0-gpu-tessellation/
  - web.engr.oregonstate.edu/~mjb/cs519/Handouts/tessellation.6pp.pdf
  - rastergrid.com/blog/2010/09/history-of-hardware-tessellation/