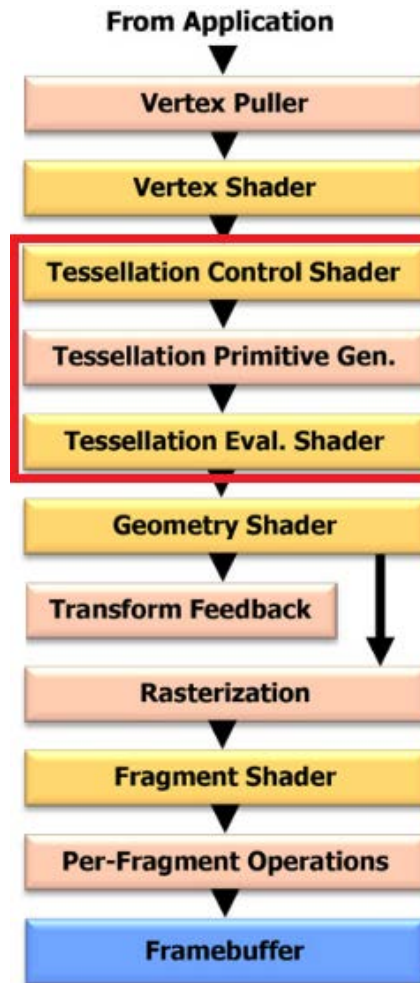


Tessellation Shaders/ LOD

Graphics Pipeline



Overview

- Create additional vertices during execution.
- Uses a new geometry primitive type called *patch*
- Tessellation stage adds two more shading stages to the OpenGL pipeline to generate a mesh
 - Tessellation Control Shader
 - Tessellation Evaluation Shader
- Tessellation Control Shader
 - Executes first
 - Operates on the patch and specify how much geometry should be generated from the patch
 - The patch can be a triangle or a quad

Overview

- Tessellation Evaluation Shader
 - Positions the vertices of the generated mesh using the tessellation coordinates
 - And sends them to rasterization stage
 - Or for more processing to the geometry shader

Tessellation Patches

- Tessellation process doesn't operate on OpenGL's classic geometry primitives like points, lines and triangles.
- Uses a new primitive type called *patch*
- They are processed by all active stages in the Pipeline.
- If you have a tessellation stage and any other primitive type is passed in it will give an error with `hGL_INVALID_OPERATION`.
- This is also true otherwise if you try rendering a patch with a tessellation stage.
- Patches are a list of vertices

Tessellation Patch

- When rendering with tessellation and patches use `glDrawArrays()` and specify the total number of vertices to be read from the bound vertex buffer.

```
glDrawArrays(GL_PATCHES, 0, 4); // in render function
```

- When using a patch OpenGL needs to be told how many vertices from the vertex array to use to make one patch. This is done using the `glPatchParameteri()` function.

```
glPatchParameteri(GL_PATCH_VERTICES, 4);
```

- 3 by default. Used for Triangle patch

```
TessModel::TessModel(GLuint program, Camera* camera){
```

```
    GLfloat points[] = {  
        -1.0f, -1.0f, 0.0f,  
        1.0f, -1.0f, 0.0f,  
        1.0f, 1.0f, 0.0f,  
        -1.0, 1.0, 0.0f  
    };
```

TessModel.cpp Constructor

```
    glPatchParameteri(GL_PATCH_VERTICES, 4); //comment for tri patch
```

```
    glGenBuffers(1, &vbo);
```

```
    glGenVertexArrays(1, &vao);
```

```
    glBindVertexArray(vao);
```

```
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

```
    glBufferData(GL_ARRAY_BUFFER, sizeof(points), &points, GL_STATIC_DRAW);
```

```
    glEnableVertexAttribArray(0);
```

```
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
```

```
    glBindVertexArray(0);
```

```
}
```

Tess Model.cpp Render

```
void TessModel::render(){  
  
    glUseProgram(this->program);  
  
    glm::mat4 model;  
    model = glm::translate(model, position);  
    glm::mat4 mvp = camera->getprojectionMatrix() * camera->getViewMatrix() *  
    model;  
  
    GLint mvLoc = glGetUniformLocation(program, "mvp");  
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvp));  
  
    glBindVertexArray(vao);  
    glDrawArrays(GL_PATCHES, 0, 4);  
    glBindVertexArray(0);  
}
```


Tessellation Control Shader

- The Tessellation control Shader is responsible
 - Generate tessellation output patch vertices that are passed to the tessellation evaluation shader as well as update any per-vertex or per patch attribute values as necessary
 - Specify the tessellation level factor that control the operation of the primitive generator.
 - There are 2 levels of tessellation
 - Inner tessellation level, specified by `gl_TessLevelInner`
 - Outer Tessellation level, specified by `gl_TessLevelOuter`
 - Tessellation level specifies by how much the inner and outer vertices needed to be subdivided by

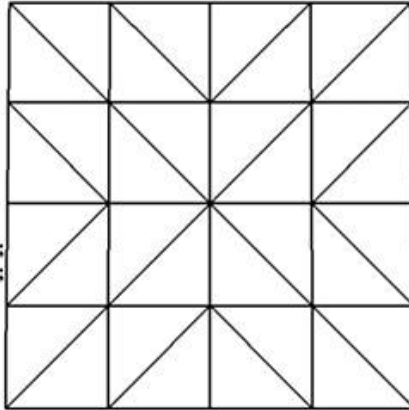
Quad Tessellation

- Used when input patches are rectangular in shape.
- The quad domain subdivides the unit square using all of the inner and outer tessellation levels.
- Outer level corresponds to the number of segments for each edge around the perimeter.
- Inner tessellation level specify how many regions are in the horizontal and vertical directions.
- In case of quad domain the tessellation cords will have 2 cords (u , v). Which will both be in the range of $[0, 1]$.

Quad Tesselation

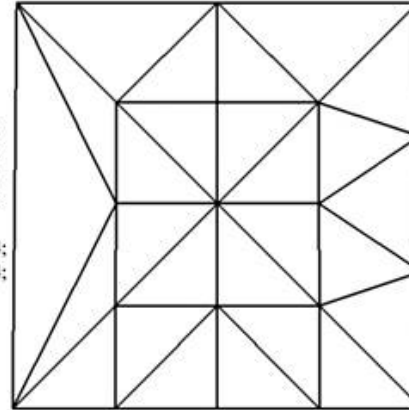
pt.EdgeTess[0] = 4;
 pt.EdgeTess[1] = 4;
 pt.EdgeTess[2] = 4;
 pt.EdgeTess[3] = 4;

 pt.InsideTess[0] = 4;
 pt.InsideTess[1] = 4;



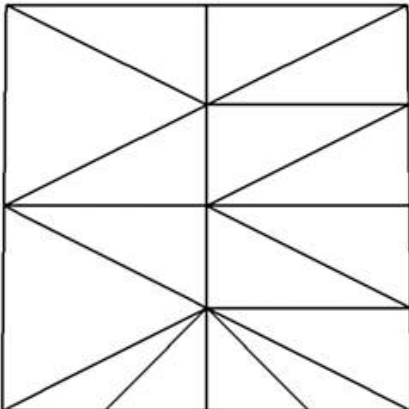
pt.EdgeTess[0] = 1;
 pt.EdgeTess[1] = 2;
 pt.EdgeTess[2] = 3;
 pt.EdgeTess[3] = 4;

 pt.InsideTess[0] = 4;
 pt.InsideTess[1] = 4;



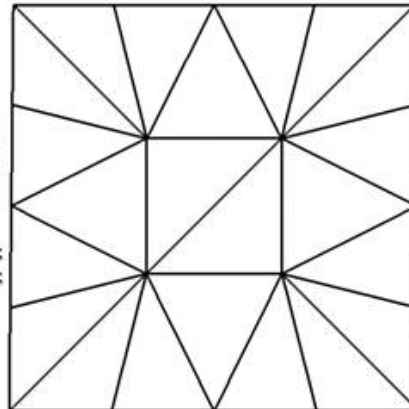
pt.EdgeTess[0] = 2;
 pt.EdgeTess[1] = 2;
 pt.EdgeTess[2] = 4;
 pt.EdgeTess[3] = 4;

 pt.InsideTess[0] = 2;
 pt.InsideTess[1] = 4;



pt.EdgeTess[0] = 4;
 pt.EdgeTess[1] = 4;
 pt.EdgeTess[2] = 4;
 pt.EdgeTess[3] = 4;

 pt.InsideTess[0] = 3;
 pt.InsideTess[1] = 3;



```
#version 430 core
```

```
//size of output patch & no. of times the TCS will be executed  
layout (vertices = 4) out;
```

```
Void main() {
```

```
    if (gl_InvocationID == 0) {
```

```
        gl_TessLevelInner[0] = 8.0;
```

```
        gl_TessLevelInner[1] = 8.0;
```

```
        gl_TessLevelOuter[0] = 4.0;
```

```
        gl_TessLevelOuter[1] = 4.0;
```

```
        gl_TessLevelOuter[2] = 4.0;
```

```
        gl_TessLevelOuter[3] = 4.0;
```

```
    }
```

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

```
}
```



Invocation ID

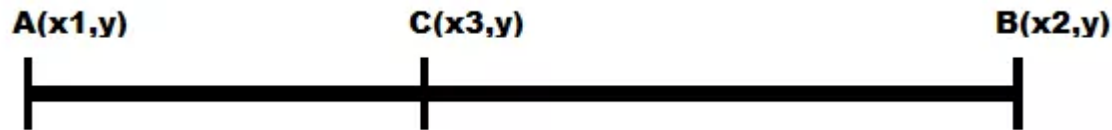
- In order to determine which output vertex is being processed, the tessellation control shader can use the *gl_InvocationID* variable.
- It has access to all patch vertex data---both input and output.
- This can lead to issues where a shader invocation might need data values from a shader invocation that hasn't happened yet.
- tessellation control shaders can use the GLSL **barrier()** function, which causes all of the control shaders for an input patch to execute and wait until all of them have reached that point
- A common idiom of tessellation control shaders is just passing the input-patch vertices out of the shader.

Tessellation Evaluation Shaders

- Final stage of the OpenGL tessellation pipeline.
- The TES is executed for each tessellation coordinate that the primitive generator emits.
- Determines the position of the vertex derived from the tessellation coordinate. Transforms the vertices into the screen space.
- Specify
 - The **in** patch type using **layout**
 - The domain: triangle, quad or isoline
 - face winding: CW / CCW
 - spacing
 - equal_spacing
 - fractional_even_spacing
 - fractional_odd_spacing

TES for Quad patch

- To determine the intermediate points we use bilinear interpolation.



- C is at 0.4 from A.
- $X_3 = 0.4(x_2 - x_1) + x_1$
- First interpolate in the x axis and then on that vertical line do it for y axis.

TES for Quad patch

```
#version 430 core
```

```
uniform mat4 mvp;
```

```
layout (quads, equal_spacing, ccw) in;
```

```
void main(){
```

```
    vec4 p1 = mix(gl_in[1].gl_Position, gl_in[0].gl_Position, gl_TessCoord.x);
```

```
    vec4 p2 = mix(gl_in[2].gl_Position, gl_in[3].gl_Position, gl_TessCoord.x);
```

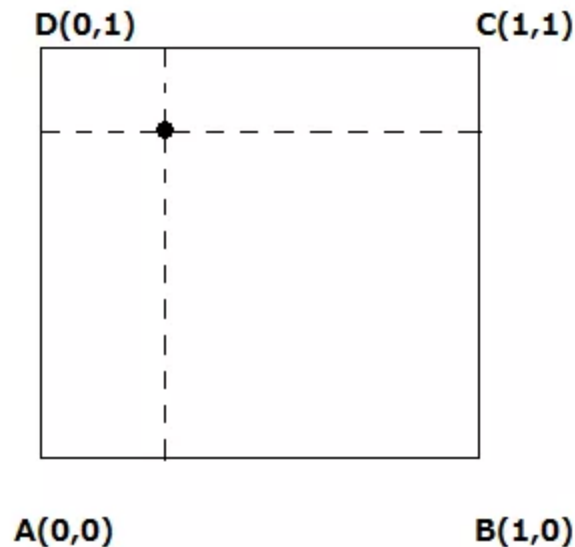
```
    vec4 pos = mix(p1, p2, gl_TessCoord.y);
```

```
    gl_Position = mvp * pos;
```

```
}
```


TES for Quad patch

- `gl_TessCoord`
- Is the texture coordinate with respect to the local quad.



Vertex Shader

```
#version 430 core
```

```
in vec3 position;
```

```
void main(){
```

```
    gl_Position = vec4(position, 1.0f);
```

```
}
```

Fragment Shader

```
#version 430 core
```

```
out vec4 color;
```

```
void main(){
```

```
    color = vec4(1.0f, 1.0f, 0.0f, 1.0f);
```

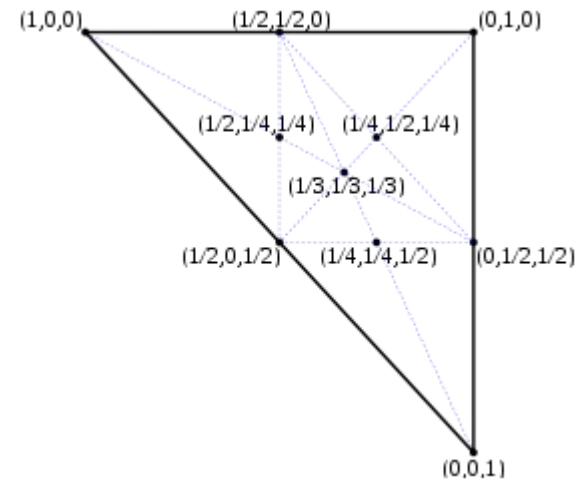
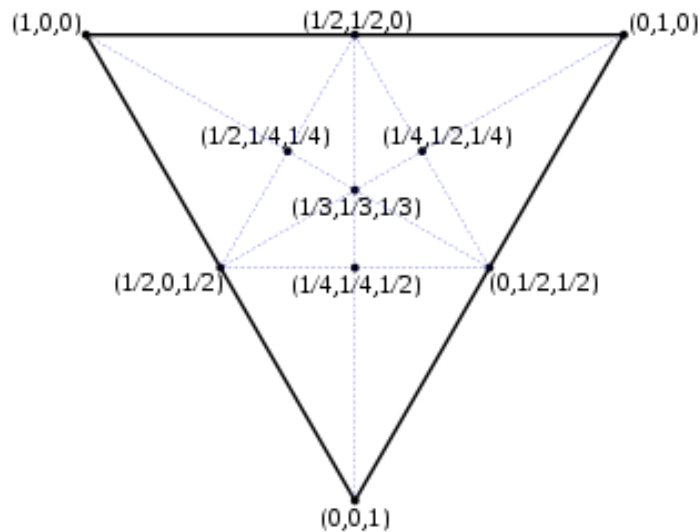
```
}
```

Triangle Tessellation

- Vertices of a triangle aren't very conveniently represented in (u, v) pair.
- Instead it uses barycentric coordinates to specify their tessellation coordinates.
- Barycentric coordinates are represented by a triplet of numbers (a, b, c) each of which range between $[0, 1]$ and also have the property that $a+b+c = 1$.
- The outer-tessellation levels control the subdivision of the perimeter of the triangle and the inner-tessellation level controls how the interior is partitioned.
- Unlike quad, the interior of the triangular domain is partitioned into a set of concentric triangles that form the regions.

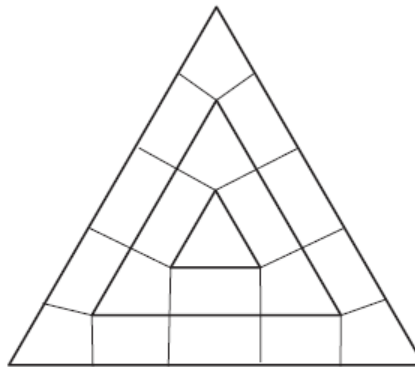
TES for triangle Patch

- Using Barry centric Coordinate system

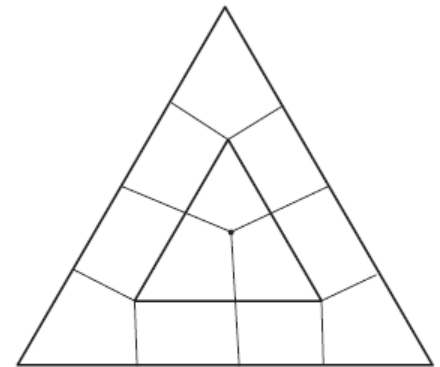


Triangle Tessellation

- let t represent the inner-tessellation level.
- If t is an even value, then $(t/2)$ concentric triangles are generated between the center point and the perimeter.
- Conversely, if t is an odd value, then $(t-1)/2$ concentric triangles are out to the perimeter.
- However, the center point (in barycentric coordinates) will not be a tessellation coordinate



Odd inner tessellation levels create a small triangle in the center of the triangular tessellation domain



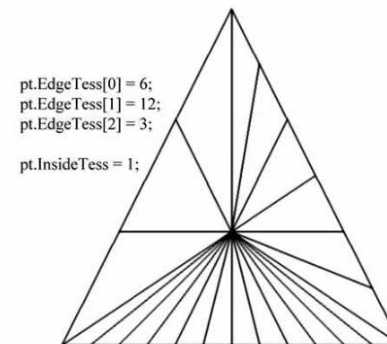
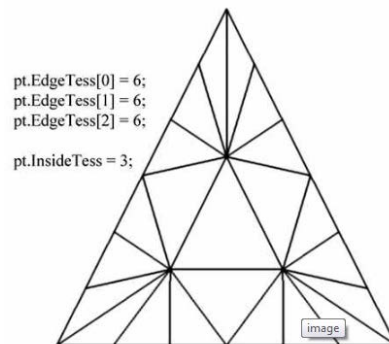
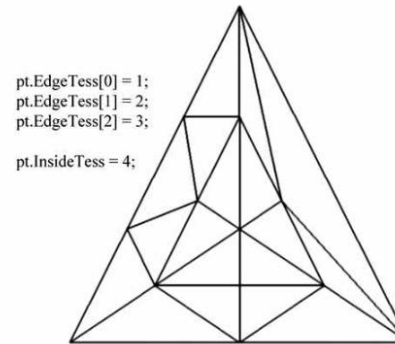
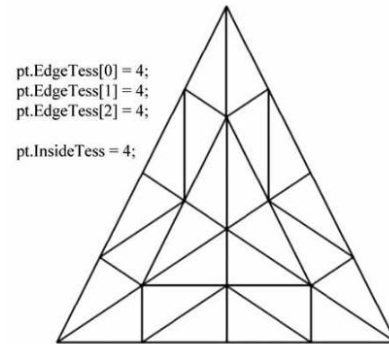
Even inner tessellation levels create a single tessellation coordinate in the center of the triangular tessellation domain

```
#version 430 core
```

```
//size of output patch & no. of times the TCS will be executed  
layout (vertices = 3) out;
```

```
void main() {  
    gl_TessLevelOuter[0] = 3.0;  
    gl_TessLevelOuter[1] = 3.0;  
    gl_TessLevelOuter[2] = 3.0;  
  
    gl_TessLevelInner[0] = 5.0;  
  
    gl_out[gl_InvocationID].gl_Position =  
    gl_in[gl_InvocationID].gl_Position;  
}
```

Triangle Tesselation



TES for triangle Patch

```
#version 430 core
```

```
uniform mat4  mvp;
```

```
layout (triangles, equal_spacing, cw) in;
```

```
void main(void){
```

```
    gl_Position= mvp * (
        gl_TessCoord.x * gl_in[0].gl_Position +
        gl_TessCoord.y * gl_in[1].gl_Position +
        gl_TessCoord.z * gl_in[2].gl_Position    );
}
```

Isoline Tessellation

- Similar to the quad domain, the isoline domain also generates (u, v) pairs as tessellation coordinates for the tessellation evaluation shader.
- Isolines, however, use only two of the outer-tessellation levels to determine the amount of subdivision.

Shader Loader.h

- Similar to geometry shader, add create program function that takes in tessellation control shader and tessellation evaluation shaders apart from vertex and fragement shaders.
- GLuint CreateProgram(char* vertexShaderFilename, char* fragmentShaderFilename, char* TessControlShaderFilename, char* TessEvalShaderFilename);

Shader Loader .cpp

- In the newly created function add the following

```
std::string tessControl_shader_code = ReadShader(TessControlShaderFilename);
```

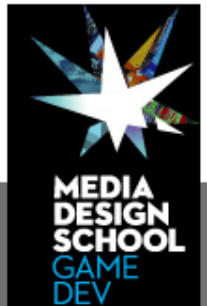
```
std::string tessEval_shader_code = ReadShader(TessEvalShaderFilename);
```

```
GLuint tessControl_shader = CreateShader(GL_TESS_CONTROL_SHADER, tessControl_shader_code, "tess  
Control shader");
```

```
GLuint tessEval_shader = CreateShader(GL_TESS_EVALUATION_SHADER, tessEval_shader_code, "tess  
Evaluation shader");
```

```
glAttachShader(program, tessControl_shader);
```

```
glAttachShader(program, tessEval_shader);
```



Usage

- Main.cpp

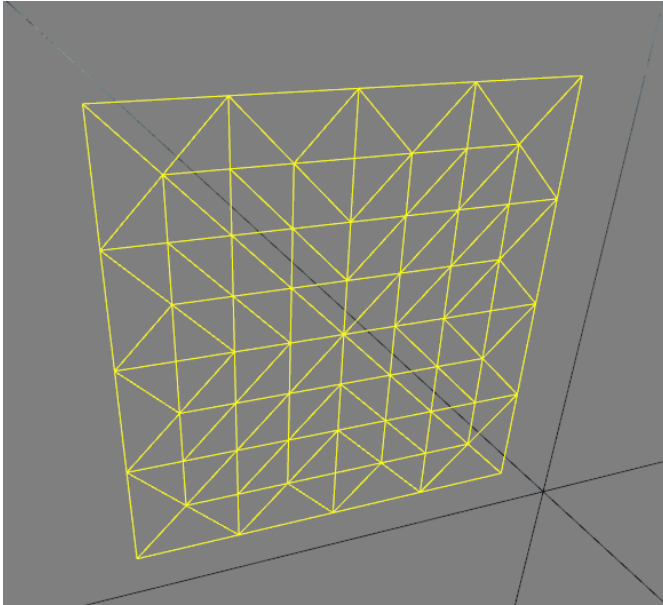
```
#include "TessModel.h"
TessModel* tessModel;
```
- Init

```
GLuint tessProgram =
shader.CreateProgram("Assets/shaders/tessTriModel.vs",
"Assets/shaders/tessTriModel.fs",
"Assets/shaders/tessQuadModel.tcs",
"Assets/shaders/tessQuadModel.tes");

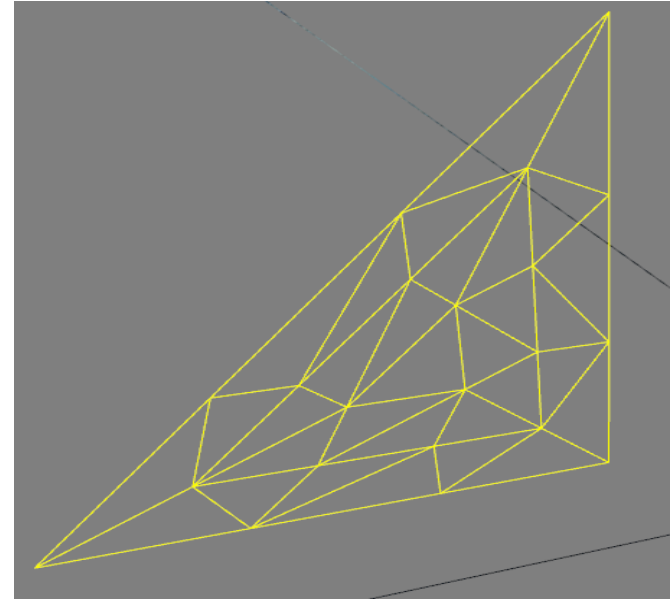
tessModel = new TessModel(tessProgram, camera);
tessModel->setPosition(glm::vec3(6.0f, -2.0f, 0.0f));
```
- Render

```
tessModel->render();
```

Output



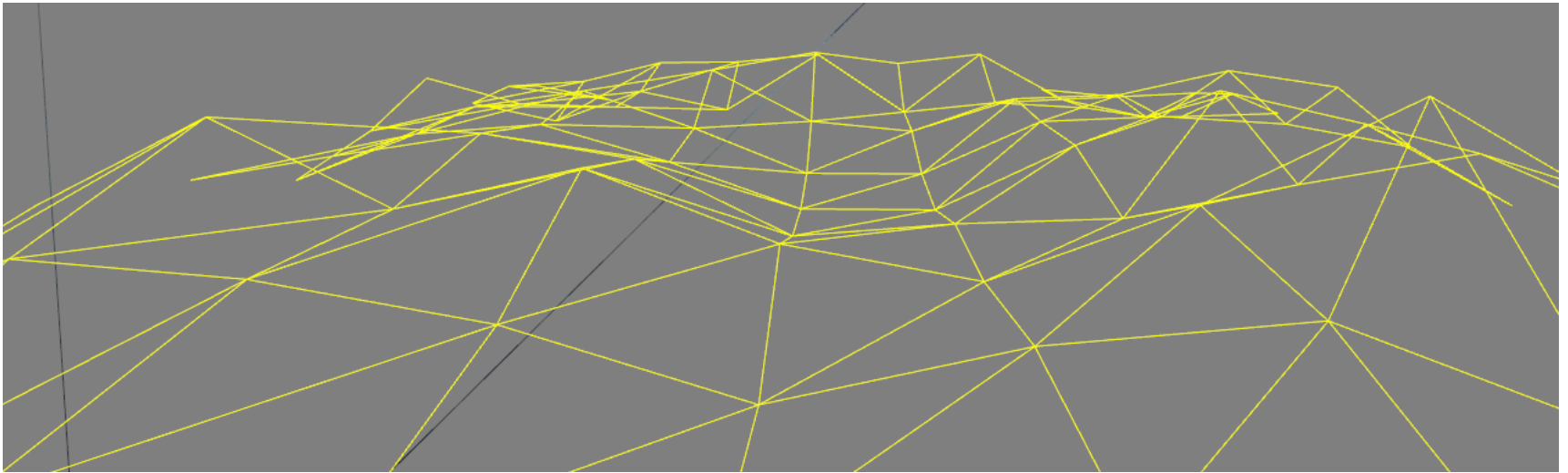
Quad Patch



Triangle Patch

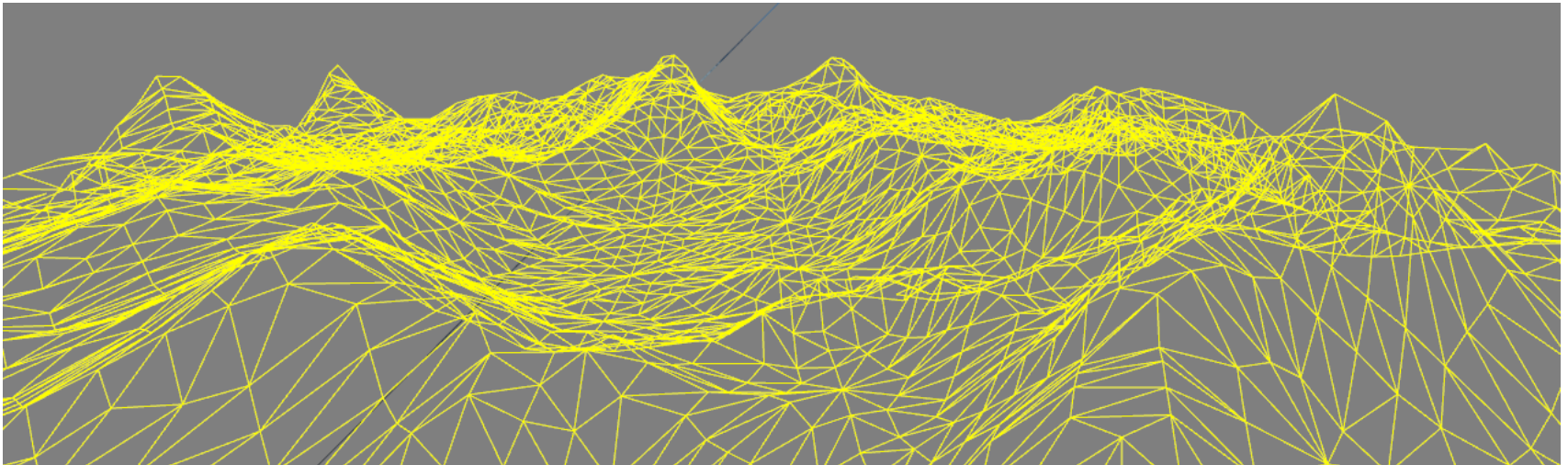
Tessellated Terrain

- Inner and Outer values = 1



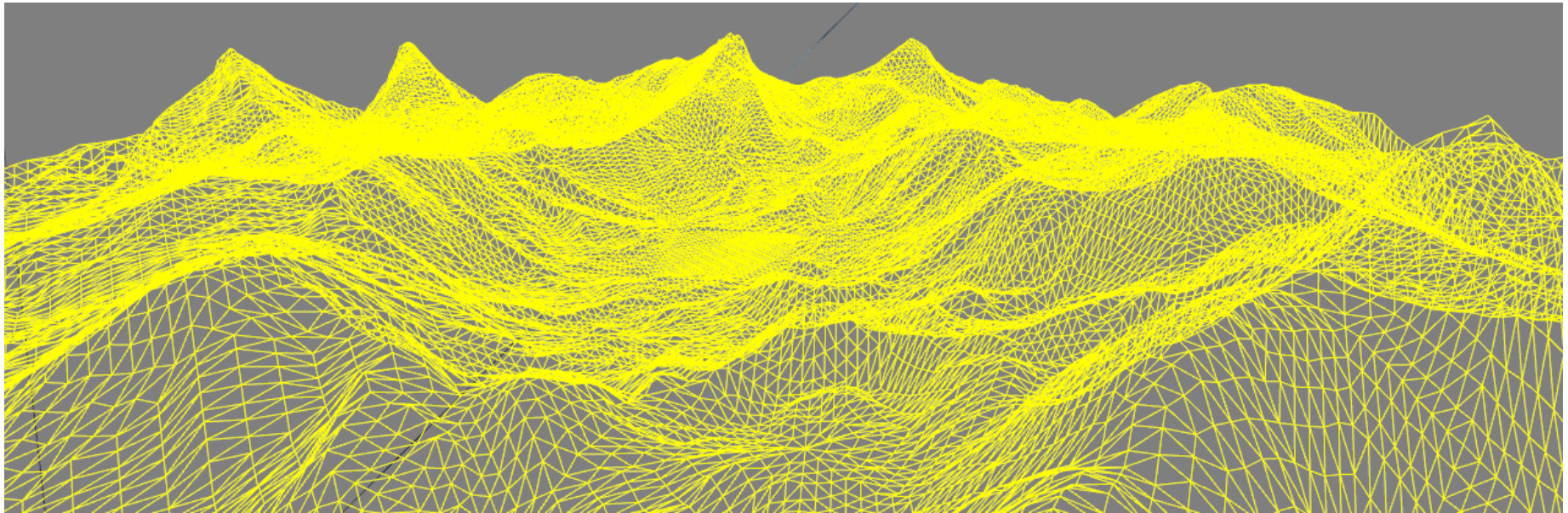
Tessellated Terrain

- Inner and Outer values = 4



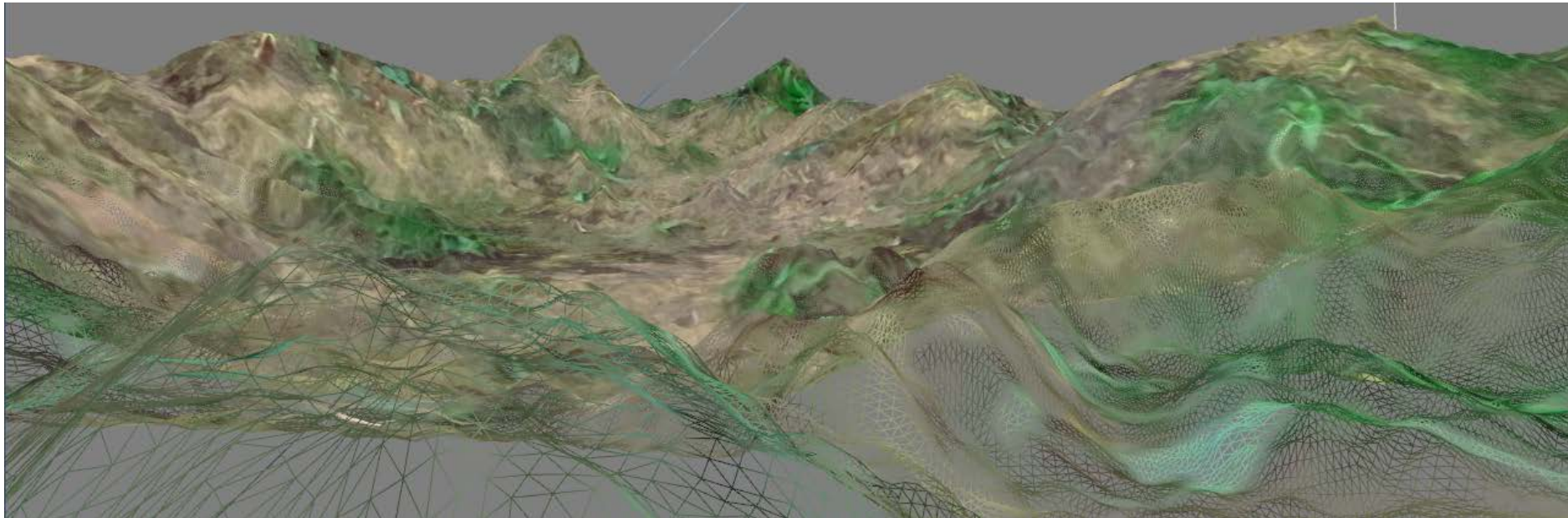
Tessellated Terrain

- Inner and Outer values = 16



Tessellated Terrain

- Textured



Level of Detail

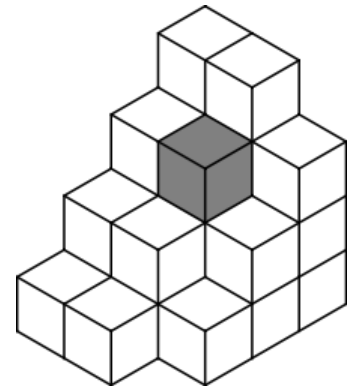
- Level of detail (LoD) in computer graphics refers to the process of simplifying the different graphical aspects of a three-dimensional (3D) object that is being rendered at a distance.
- As one get further away from an object , it takes up less of our vision (and less dots of resolution on your monitor).
- Therefore, we can drop its *level of detail* , saving memory and computation power
- The purpose of implementing level of detail into a program is so the speed of rendering can be increased

LOD

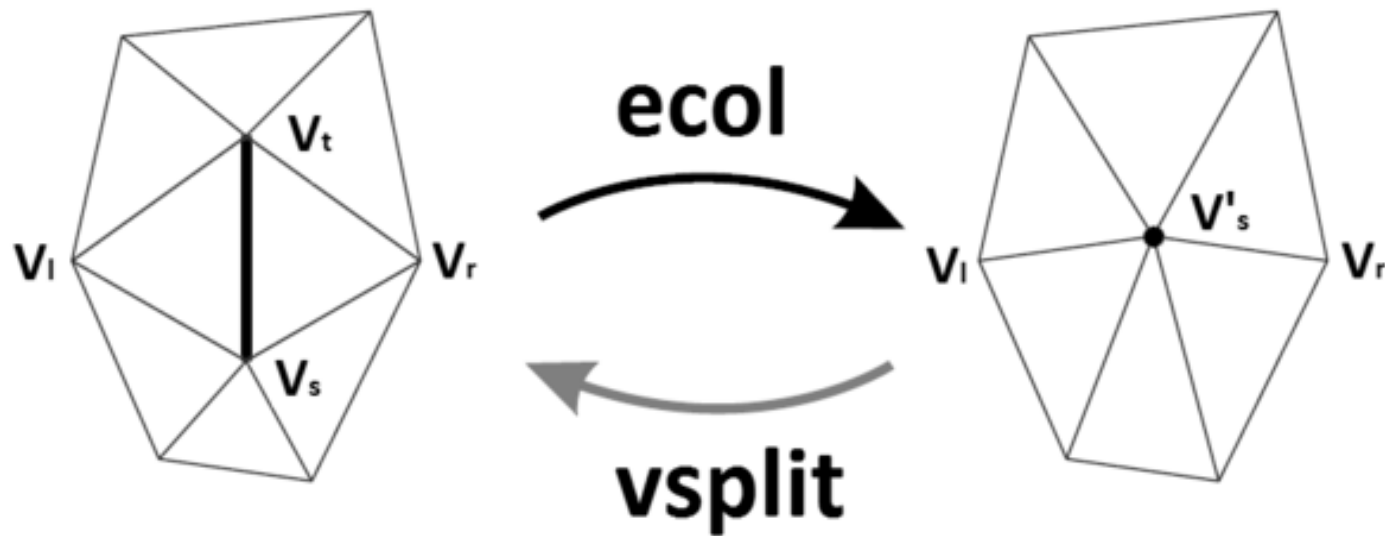
- LOD can be integrated into the program in 2 ways :
 - Discrete
 - Continuous
 - Trans-Voxel Method
 - Progressive Mesh
 - Edge Collapse
 - Vertex Split
 - ROAM (Real Time Optimally Adapting Meshes)
 - Tessellation

Trans- Voxel Method

- Voxel (volumetric pixel) represents a value on a regular grid in 3-d space.
- Voxel do not have position in their values
- Its position is based on relative position of other voxels
- Voxel represents a single sample of data
- Could be single or multiple pieces of data
- Uses
 - Used to create caves, overhangs, arches.
 - These cannot be created using height-maps.

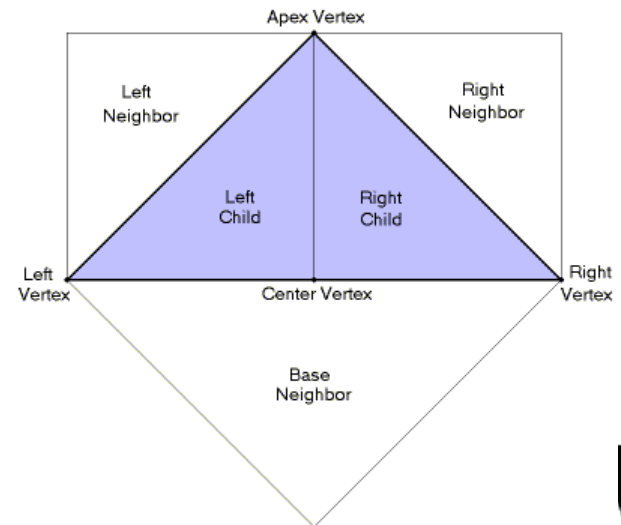


Progressive Mesh

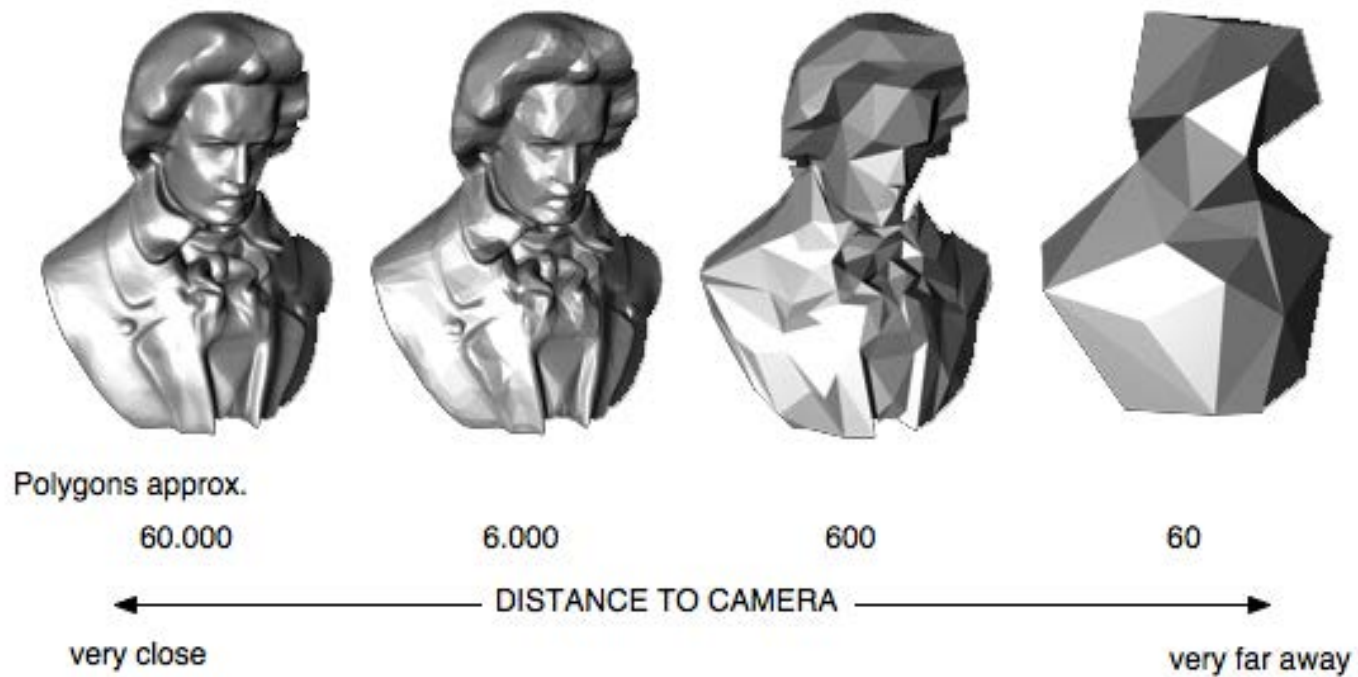


ROAM

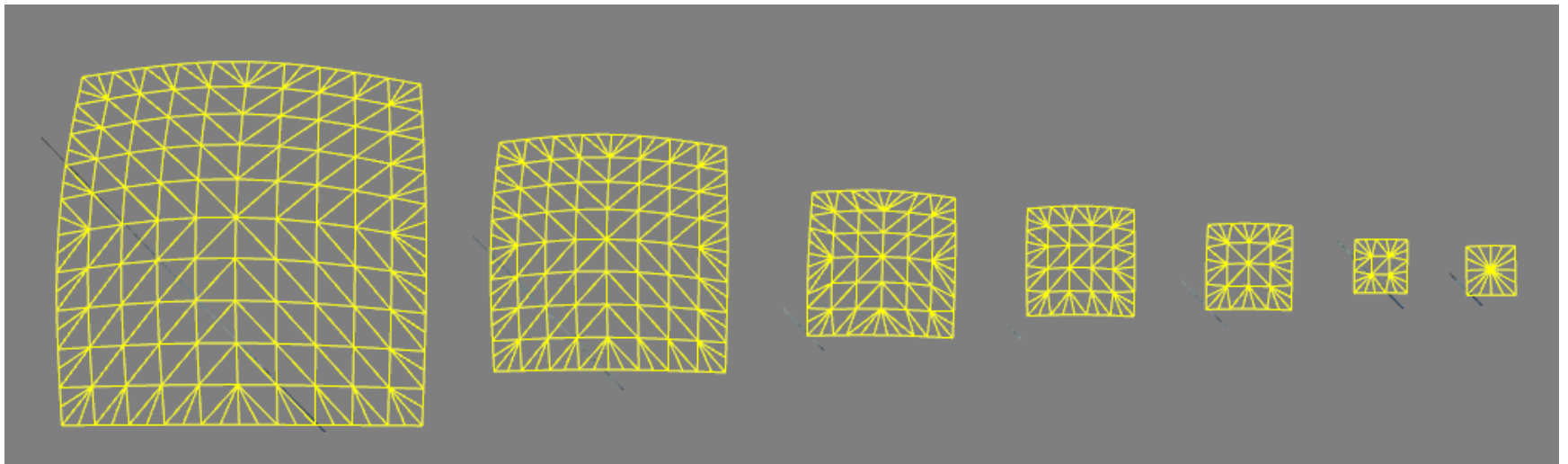
- It uses the same concept as progressive mesh
- Represents the terrain in the form a binary triangle tree
- Five relations are important for a ROAM structure
 - Left child
 - Right child
 - Left neighbour
 - Right neighbour
 - Base neighbour
- Uses tessellation and patches



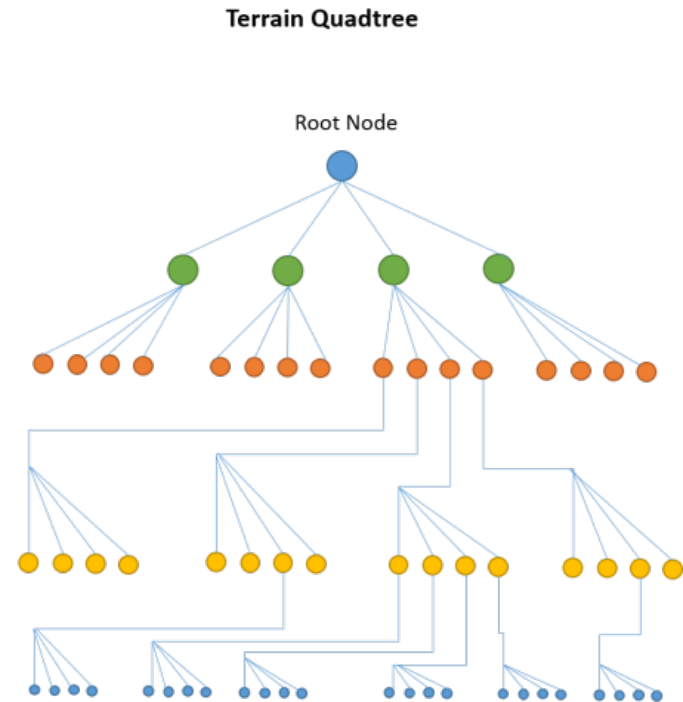
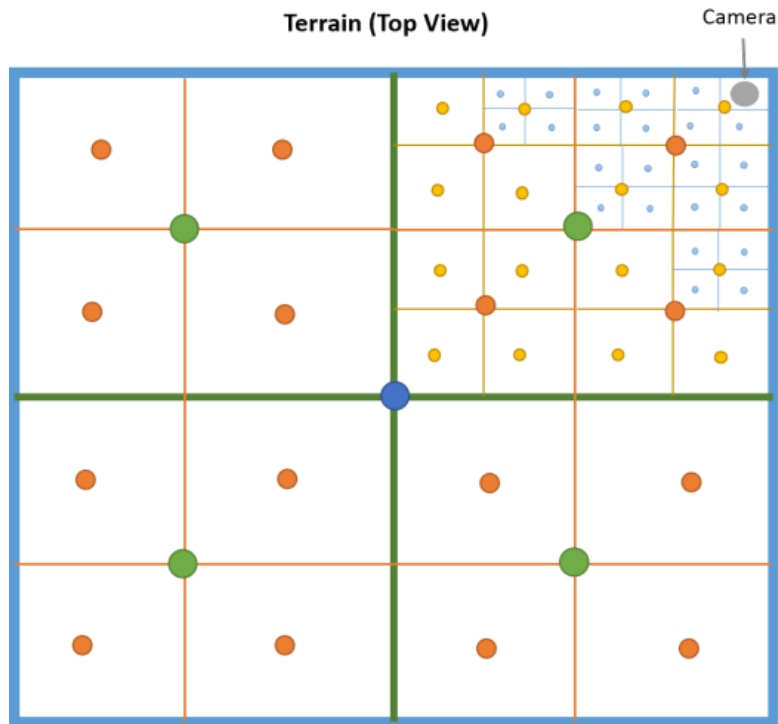
Tessellation LOD



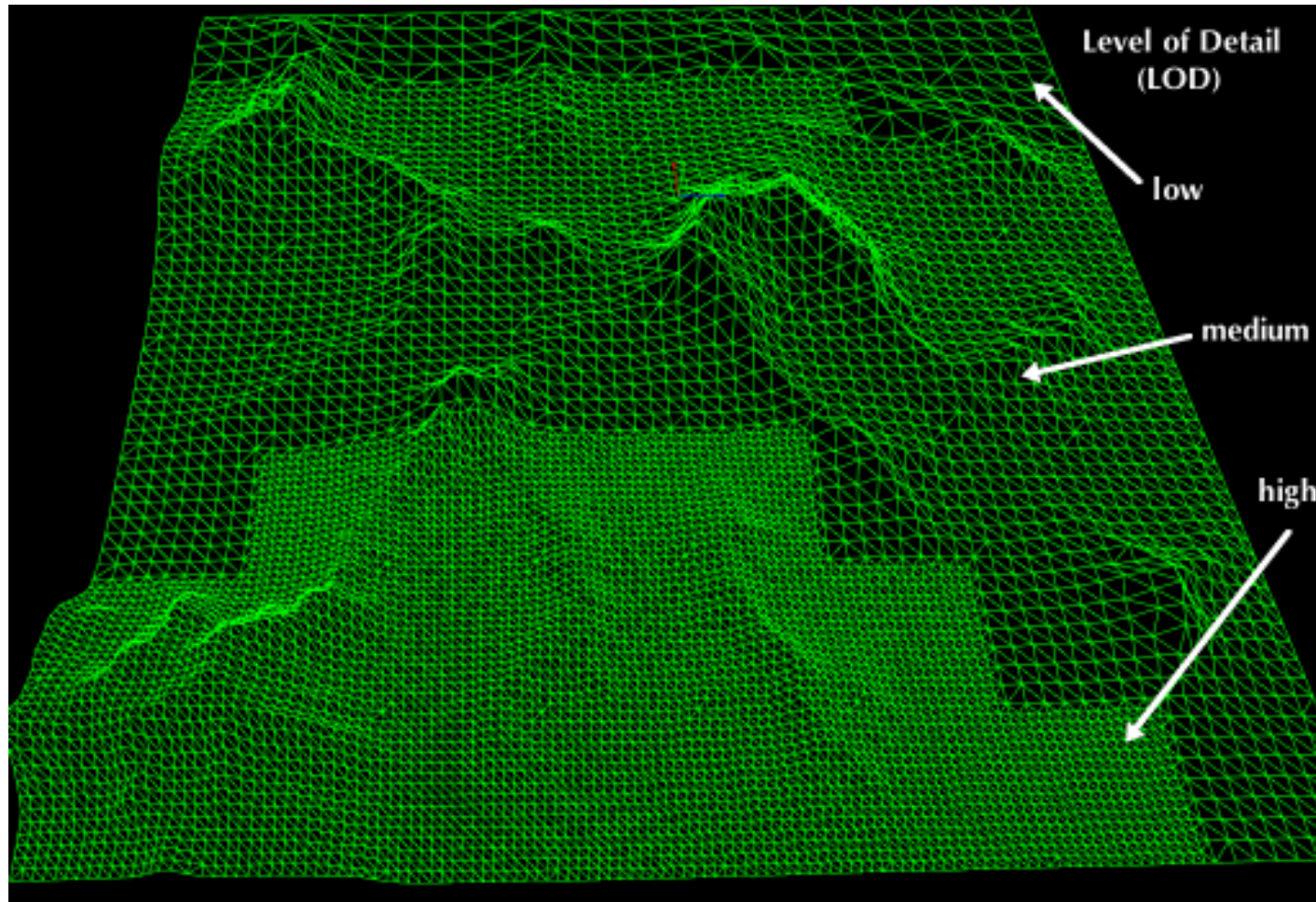
Tessellation LOD



Quad Tree



Quad Tree



Further reading

- <http://www.terathon.com/voxels/>
- http://en.wikipedia.org/wiki/Marching_cubes
- <http://www.terathon.com/lengyel/Lengyel-VoxelTerrain.pdf>
- http://www.gamasutra.com/view/feature/131596/real_time_dynamic_level_of_detail_.php