

Real-time Graphics

8. Geometry Optimization

Martin Samuelčík

Geometry

- PS3 – 275 million triangles per second = 4,6 million triangles per frame (60 fps)
- XBOX – 500 million triangles per second
- Still needed optimization to reduce number of triangles or how array of triangles is sent for GPU processing
- Preprocessing stage
- On fly CPU, GPU processing

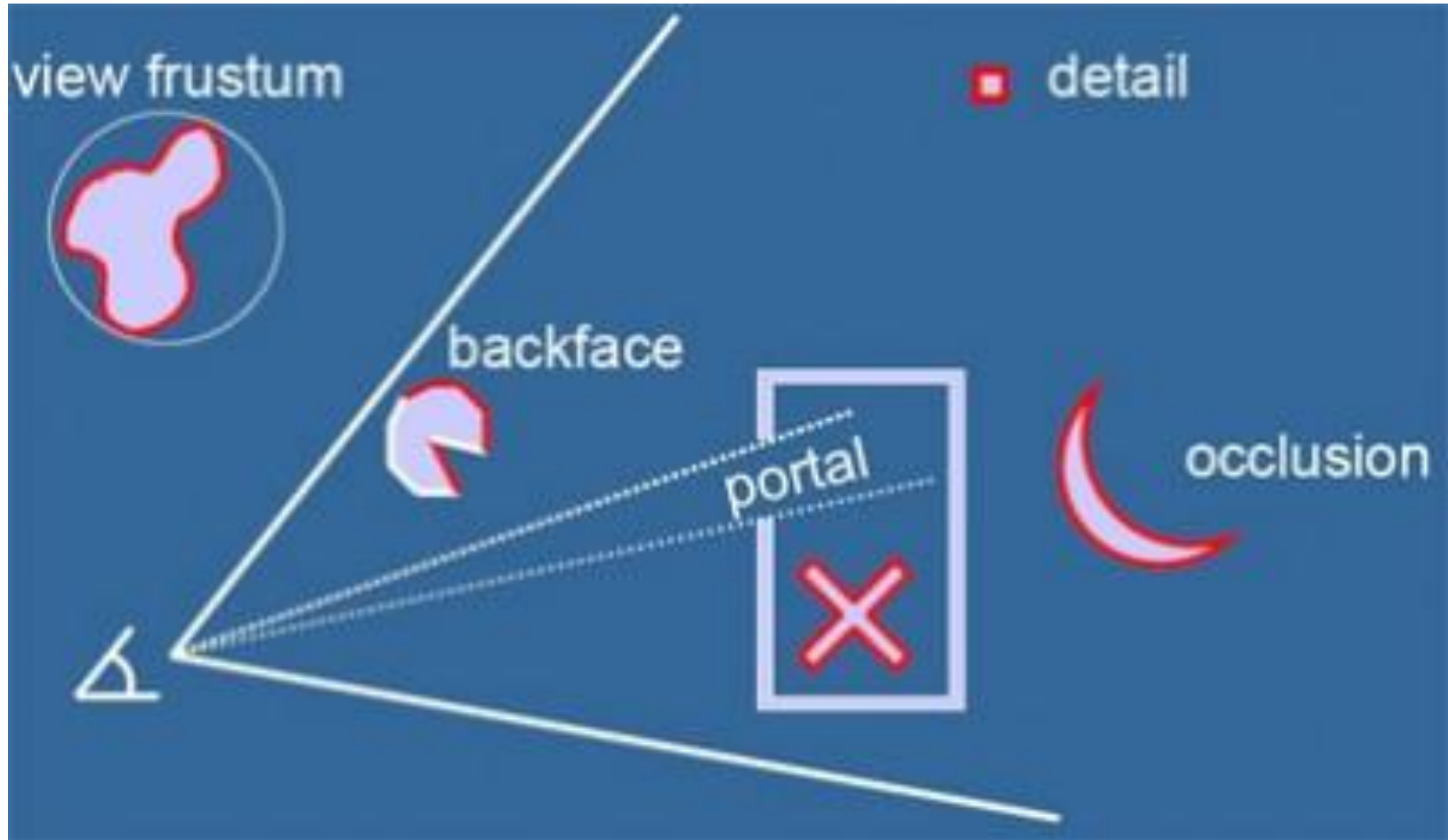


Reducing geometry

- Reduce number of triangles used for frame rendering
- Reduce number of buffer switches
- Eliminate triangles (buffers) that are not visible
- Eliminate or simplify triangles with only few pixels on screen
- Use GPU to create triangles



Culling



Back-face culling

- Front faces given by order of vertices based on winding in window coordinates
 - *void glFrontFace(GLenum mode)*
 - *mode* - GL_CCW, GL_CW
- Remove faces with given facing from rendering pipeline, usually we want remove back faces
 - *void glCullFace(GLenum mode)*
 - *mode* - GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
- Needed model with proper vertices ordering



Frustum culling

- Remove triangles from rendering that are not inside viewing frustum
- Triangle-frustum intersection
- Using simple bounding volumes for objects
 - Sphere
 - Axis aligned bounding box, Oriented bounding box
 - kDOP
- Using hierarchy of bounding volumes
- Volume-frustum intersection



Occlusion culling

- Remove triangles from rendering that are not visible from current viewpoint
- Static scene – preprocessing, visibility between cells (grid), (Potentially visible set)
- Portal culling – visibility between rooms (cells) connected with doors (portals)
- Dynamic scene – determine if parts of scene are visible or occluded



Occlusion culling

- GPU support – [GL_ARB_occlusion_query](#), [GL_ARB_occlusion_query2](#)
- Computes number of visible pixels for rendered named object (or just boolean value)
- Problems: CPU stalls (waiting for query results), many queries
- Using bounding volume hierarchies
 - Issue occlusion query for the node.
 - Stop and wait for the result of the query.
 - If the node is visible:
 - If it is an interior node:
 - Sort the children in front-to-back order.
 - Call the algorithm recursively for all children.
 - If it is a leaf node, render the objects contained in the node.



Occlusion query

```
GLuint query;
glGenQueriesARB( 1, &.query );
for( int i = 0; i < scene_objects[i]; i++ ) {
    // find out how many fragments of object's AABB will be visible
    glBeginQueryARB( GL_SAMPLES_PASSED_ARB, query);
    glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );
    glDepthMask( GL_FALSE );
    DrawSimpleAABB(scene_objects[i]);
    glEndQueryARB( GL_SAMPLES_PASSED_ARB );
    glFlush();

    // wait for result of occlusion query
    GLint query_available;
    do {
        DoSomeIdleStuff();
        glGetQueryObjectivARB(query, GL_QUERY_RESULT_AVAILABLE_ARB, &query_available);
    } while (!query_available);

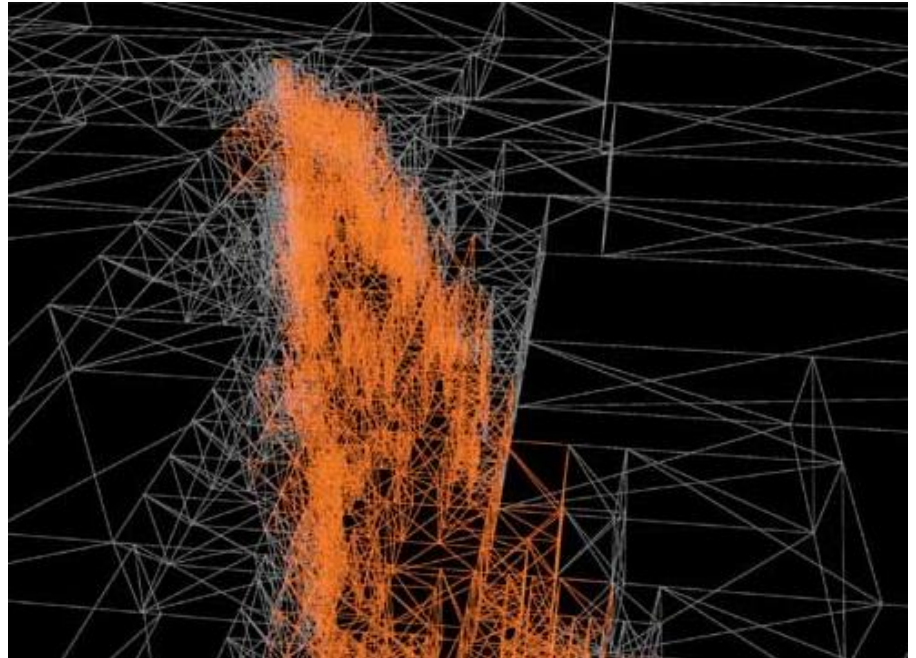
    // draw object only if its AABB is somehow visible
    GLuint fragmentcount;
    glGetQueryObjectuivARB(query, GL_QUERY_RESULT_ARB, &fragmentcount);
    glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
    glDepthMask(GL_TRUE);
    if (fragmentcount > 0)
        DrawObject(scene_objects[i]);

    // update priority of object in list of objects based on fragments rendered
    // more fragments = higher priority = object will be rendered sooner next frame
    scene_objects.SetPriority(i, fragmentcount);
}
```



Occlusion culling

<http://developer.nvidia.com/node/23>

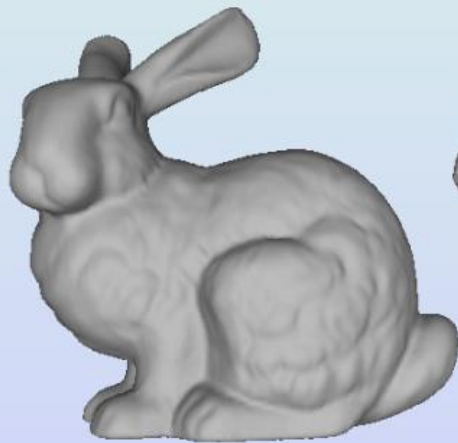


[Michael Wimmer, Jirí Bittner]

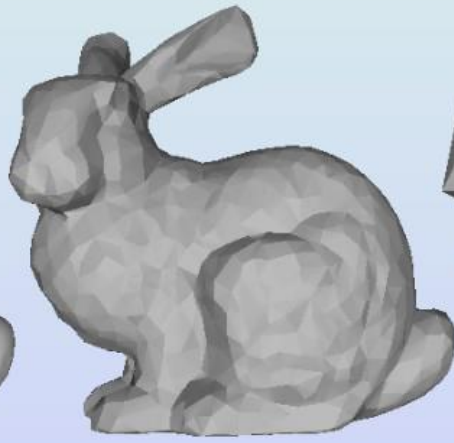


Levels of Detail

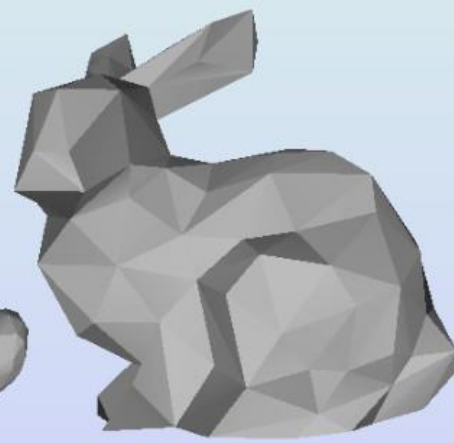
- Objects with only a few pixels on screen rendered as simplified models
- Creation of simplified models



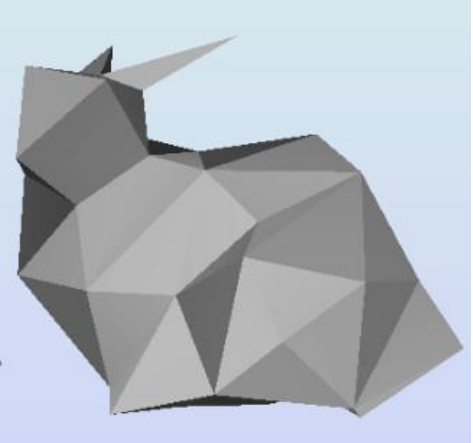
69,451 polys



2,502 polys



251 polys

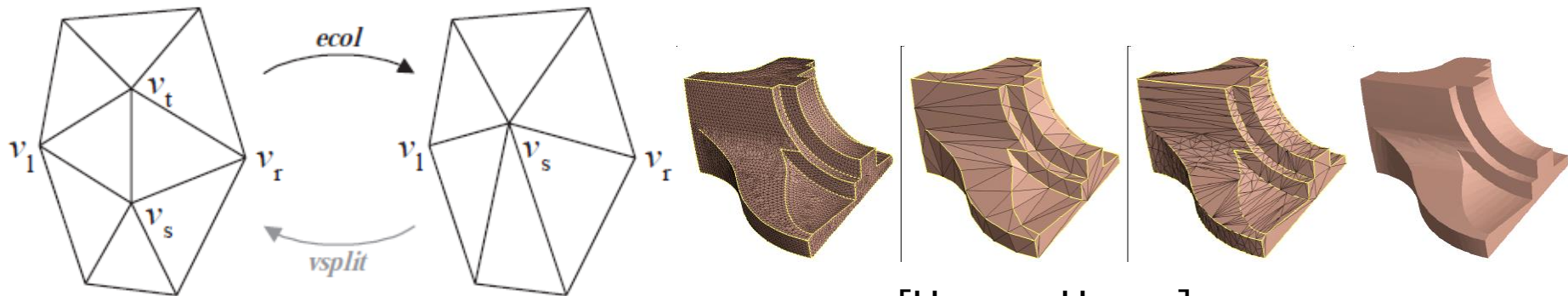


76 polys



LoD

- Progressive meshes – mesh loading
- <http://research.microsoft.com/en-us/um/people/hoppe/proj/pm/>
- From coarse mesh to finest using vertex split, created using edge collapse



[Hugues Hoppe]



LoD sources

- <http://meshlab.sourceforge.net/>
- <http://assimp.sourceforge.net/>
- http://jsomers.com/vipm_demo/meshsimp.html
- <http://lodbook.com/>



LEVEL of DETAIL FOR 3D GRAPHICS

David LUEBKE · Mani REDDY · Jonathan D. COHEN
Amitabh VARSHNEY · Benjamin WATSON · Robert HUEBNER
FOREWORD BY FREDERICK P. BROOKS, JR.

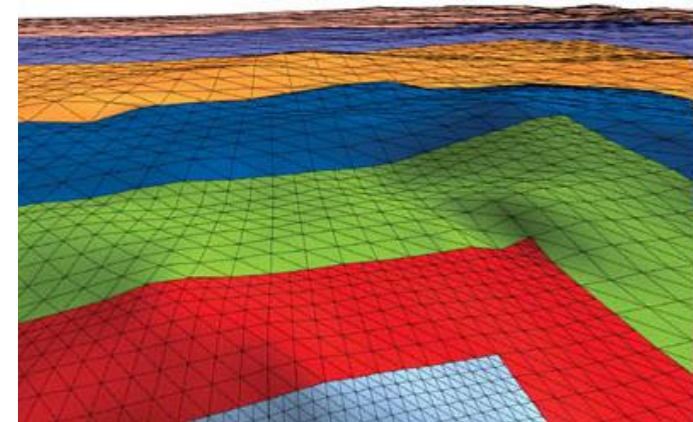
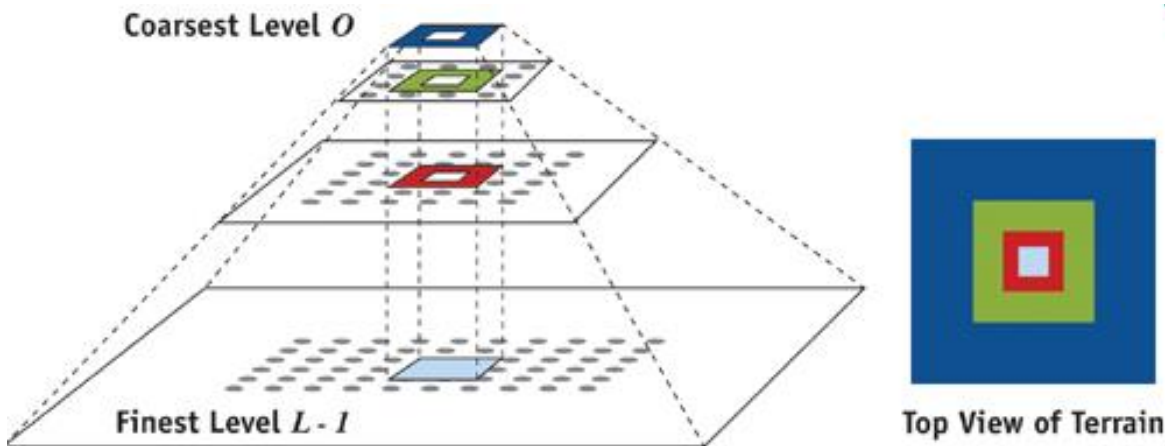


Real-time Graphics

Martin Samuelčík

Terrain LoD

- Terrain – 2D elevation (height) map
- Geometry clipmaps
- Using prefiltered mipmapped height maps



Terrain LOD

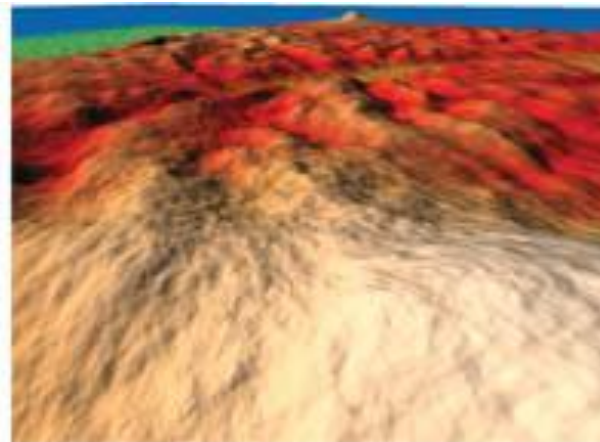
- Blending heights on levels boundary

$$\alpha_x = \text{clamp} \left(\left(x - v_x - \left(\frac{n-1}{2} - w - 1 \right) \right) w, 0, 1 \right),$$

- GPU, shaders implementation
- <http://developer.nvidia.com/node/19>



(a)



(b)



Geometry shader

- Using GPU for creating triangles

```
#version 120
#extension GL_EXT_geometry_shader4 : enable
void main(void)
{
    int i;
    //Pass-thru!
    for(i=0; i< gl_VerticesIn; i++)
    {
        gl_Position = gl_PositionIn[i];
        EmitVertex();
    }
    EndPrimitive();

    //New piece of geometry! We just swizzle the x and y terms
    for(i=0; i< gl_VerticesIn; i++)
    {
        gl_Position = gl_PositionIn[i];
        gl_Position.xy = gl_Position.yx;
        EmitVertex();
    }
    EndPrimitive();
}
```



Geometry shader

- Adjacency arguments for glBegin
 - GL_LINES_ADJACENCY_EXT
 - GL_LINE_STRIP_ADJACENCY_EXT
 - GL_TRIANGLES_ADJACENCY_EXT
 - GL_TRIANGLE_STRIP_ADJECENCY_EXT

If a Vertex Shader Writes Variables as:	then the Geometry Shader will Read Them as:	and will Write Them as:
gl_Position	gl_PositionIn[<input type="checkbox"/>]	gl_Position
gl_Normal	gl_NormalIn[<input type="checkbox"/>]	gl_Normal
gl_TexCoord[]	gl_TexCoordIn[] [<input type="checkbox"/>]	gl_TexCoord[]
gl_FrontColor	gl_FrontColorIn[<input type="checkbox"/>]	gl_FrontColor
gl_BackColor	gl_BackColorIn[<input type="checkbox"/>]	gl_BackColor
gl_PointSize	gl_PointSizeIn[<input type="checkbox"/>]	gl_PointSize
gl_Layer	gl_LayerIn[<input type="checkbox"/>]	gl_Layer
gl_PrimitiveID	gl_PrimitiveIDIn[<input type="checkbox"/>]	gl_PrimitiveID

In the Geometry Shader, the dimensions indicated by are given by the variable *gl_VerticesIn*, although you will already know this by the type of geometry you are inputting

→	1 GL_POINTS 2 GL_LINES 4 GL_LINES_ADJACENCY_EXT 3 GL_TRIANGLES 6 GL_TRIANGLES_ADJACENCY_EXT
---	---

```
glProgramParameteriEXT( program, GL_GEOMETRY_INPUT_TYPE_EXT, inputGeometryType );
glProgramParameteriEXT( program, GL_GEOMETRY_OUTPUT_TYPE_EXT, outputGeometryType );
glProgramParameteriEXT( program, GL_GEOMETRY_VERTICES_OUT_EXT, 101 );
```



Tessellation shaders

- `GL_ARB_tessellation_shader`, OpenGL 4.0
- Tessellation control shader transforms per-vertex data and sets per-patch tess. levels
- Tessellation evaluation shader computes position and attributes of new generated vertices

```
// VERTEX SHADER
in vec4 position;
uniform sampler2D terrain;

void main(void)
{
    vec2 texcoord = position.xy;
    float height = texture(terrain, texcoord).a;
    vec4 displaced = vec4(position.x, position.y, height, 1.0);
    gl_Position = displaced;
}
```



Tessellation shaders

```
// TESSELLATION CONTROL SHADER
// The main function is called for each vertex in the patch.
// gl_InvocationID identifies which vertex is being processed.
layout(vertices = 4) out;
uniform vec2 screen_size;
uniform mat4 mvp;
uniform float lod_factor;

bool offscreen(vec4 vertex)
{
    if(vertex.z < -0.5) return true;
    return any(lessThan(vertex.xy, vec2(-1.7)) ||
              greaterThan(vertex.xy, vec2(1.7)));
}

vec4 project(vec4 vertex)
{
    vec4 result = mvp * vertex;
    result /= result.w;
    return result;
}

vec2 screen_space(vec4 vertex)
{
    return (clamp(vertex.xy, -1.3, 1.3)+1) * (screen_size*0.5);
}

float level(vec2 v0, vec2 v1)
{
    return clamp(distance(v0, v1)/lod_factor, 1, 64);
}
```

```
void main()
{
    if(gl_InvocationID == 0)
    {
        vec4 v0 = project(gl_in[0].gl_Position);
        vec4 v1 = project(gl_in[1].gl_Position);
        vec4 v2 = project(gl_in[2].gl_Position);
        vec4 v3 = project(gl_in[3].gl_Position);

        if(all(bvec4(offscreen(v0), offscreen(v1), offscreen(v2), offscreen(v3))))
        {
            gl_TessLevelInner[0] = gl_TessLevelInner[1] = 0;
            gl_TessLevelOuter[0] = gl_TessLevelOuter[1] = 0;
            gl_TessLevelOuter[2] = gl_TessLevelOuter[3] = 0;
        }
        else
        {
            vec2 ss0 = screen_space(v0);
            vec2 ss1 = screen_space(v1);
            vec2 ss2 = screen_space(v2);
            vec2 ss3 = screen_space(v3);
            float e0 = level(ss1, ss2);
            float e1 = level(ss0, ss1);
            float e2 = level(ss3, ss0);
            float e3 = level(ss2, ss3);
            gl_TessLevelInner[0] = mix(e1, e2, 0.5);
            gl_TessLevelInner[1] = mix(e0, e3, 0.5);
            gl_TessLevelOuter[0] = e0;
            gl_TessLevelOuter[1] = e1;
            gl_TessLevelOuter[2] = e2;
            gl_TessLevelOuter[3] = e3;
        }
    }
    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
}
```



Tessellation shaders

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

```
// TESSELATION EVALUATION SHADER
// The Evaluation main function is called once for each vertex
// of the tessellated output.
// The coordinate is given as UV vector relative to the positions
// of the patches control points.
layout(quads, fractional_odd_spacing, ccw) in;
out vec2 texcoord;
out float depth;
uniform sampler2D terrain;
uniform mat4 mvp;

void main()
{
    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

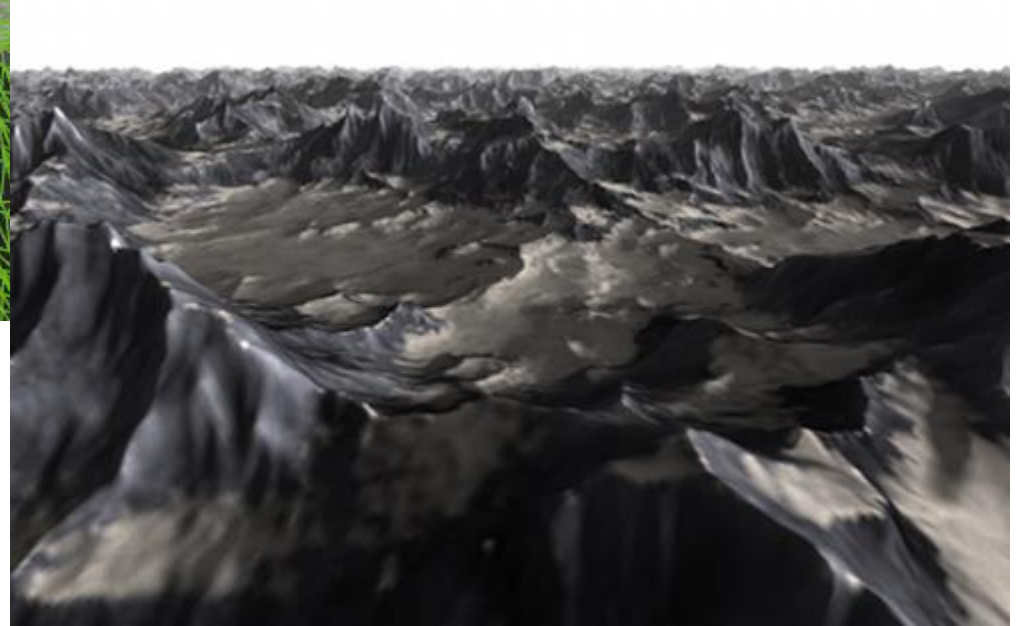
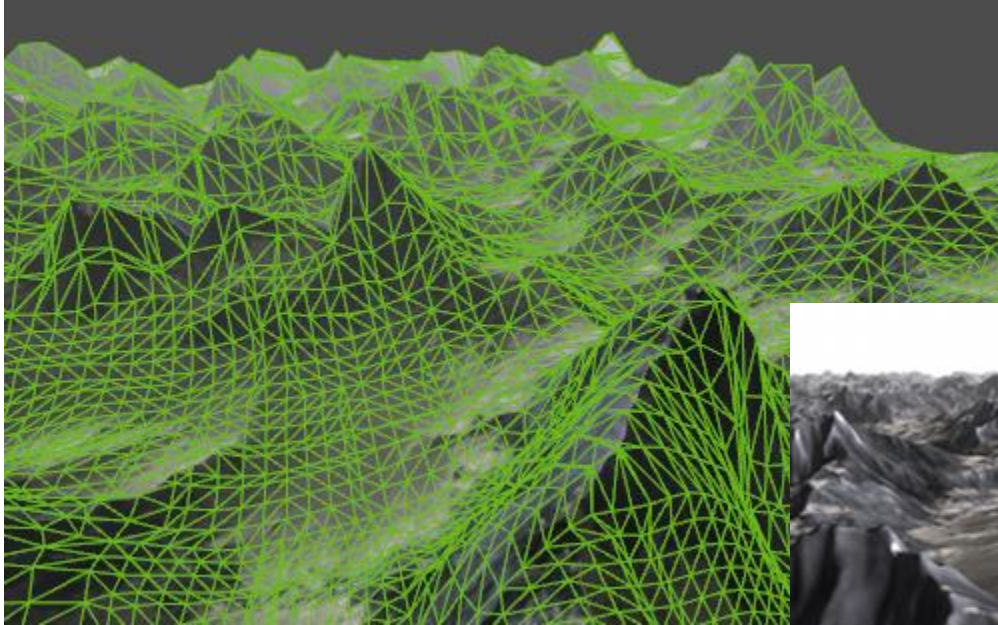
    vec4 a = mix(gl_in[1].gl_Position, gl_in[0].gl_Position, u);
    vec4 b = mix(gl_in[2].gl_Position, gl_in[3].gl_Position, u);
    vec4 position = mix(a, b, v);
    texcoord = position.xy;
    float height = texture(terrain, texcoord).a;
    gl_Position = mvp * vec4(texcoord, height, 1.0);
    depth = gl_Position.z;
}
```

```
// FRAGMENT SHADER
in vec2 texcoord;
in float depth;
out vec4 fragment;
uniform sampler2D diffuse;
uniform sampler2D terrain;
uniform sampler2D noise_tile;
vec3 incident = normalize(vec3(1.0, 0.2, 0.5));
vec4 light = vec4(1.0, 0.95, 0.9, 1.0) * 1.1;

void main()
{
    vec3 normal = normalize(texture(terrain, texcoord).xyz);
    vec4 color = texture(diffuse, texcoord);
    float noise_factor = texture(noise_tile, texcoord*32).r+0.1;
    float dot_surface_incident = max(0, dot(normal, incident));
    color = color * light * noise_factor * (max(0.1, dot_surface_incident)+0.05)*1.5;
    fragment = mix(color, color*0.5+vec4(0.5, 0.5, 0.5, 1.0), depth*2.0);
}
```



Tessellation shaders



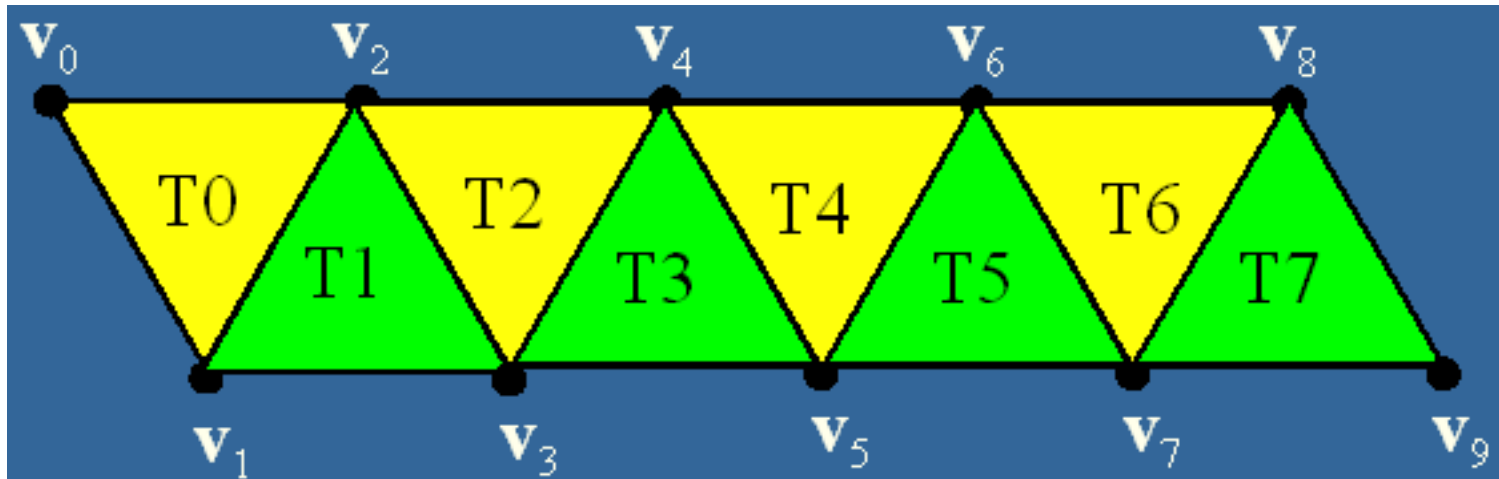
Optimizing geometry

- Reduce number of API calls and state changes!
- Define triangles with fewer amount of data
- Create as large vertex arrays as possible
- Use small number of textures
- Use geometry instancing
- Use VBO and its enhancements
- https://www.opengl.org/wiki/Vertex_Rendering



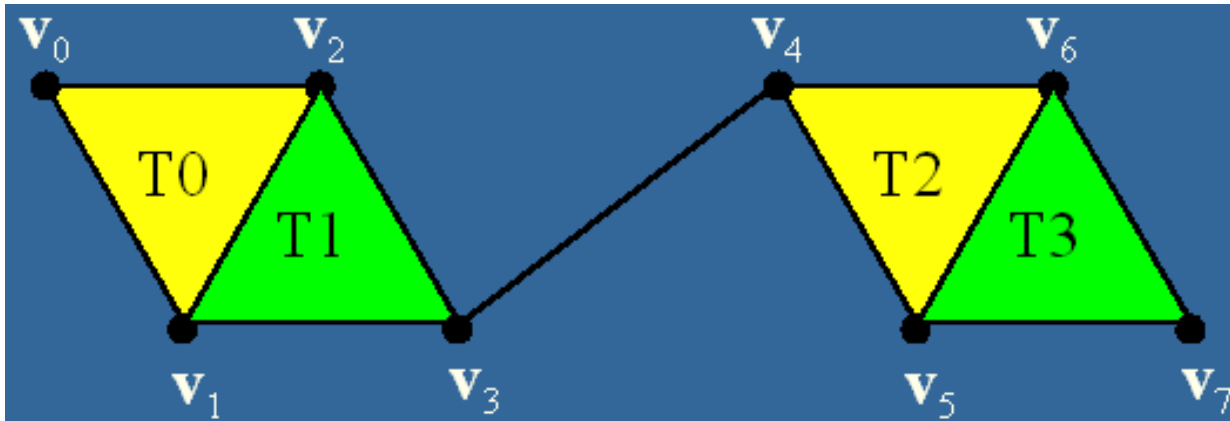
Polygon strips

- Reducing number of data for representing triangles
- Can increase number of buffers, API calls
- OpenGL maintain orientation



Polygon strips

- Non-connected strips – swaps 0,1,2,3,3,4,4,5,6,7



- Or use primitive restart - [GL_NV_primitive_restart](#)
- Creation
 - Manually, own code
 - NVTriStripper



Geometry instancing

- Scene with many same objects (trees, people)
- Display lists
- VBOs
- Dynamic update of VBO data
- GPU geometry instancing
 - `GL_EXT_draw_instanced`
 - Draw VBO using `glDrawArraysInstancedEXT(GLenum mode, GLint start, GLsizei count, GLsizei primcount)`,
`glDrawElementsInstancedEXT(GLenum mode, GLsizei count, GLenum type, const GLvoid *indices, GLsizei primcount)`
 - In vert. shader, use `gl_InstanceID`, to get which instance is processed
 - Based on instance ID, use different transformation, attributes, ...

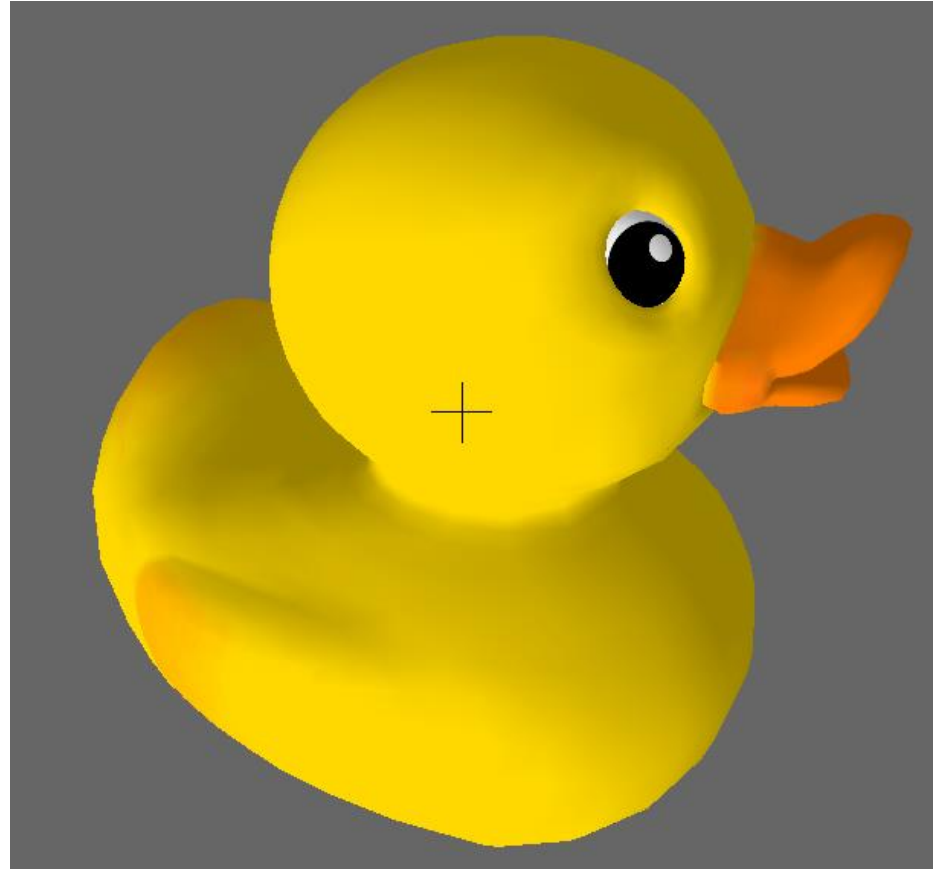


Texture atlas

- Combination of as many textures as possible in one texture
- Combination of meshes into one set of vertex arrays
- Recomputation of texture coordinates
- Reducing number of state changes (texture switches)
- Problems: No tiling, color bleeding (mipmap, filtering)



Texture atlas



Texture atlas

- <http://developer.nvidia.com/legacy-texture-tools>
- <http://unity3d.com/support/documentation/ScriptReference/Texture2D.PackTextures.html>
- <http://www.scriptspot.com/3ds-max/scripts/texture-atlas-generator>
- <http://www.openscenegraph.org/documentation/OpenSceneGraphReferenceDocs/a00479.html>
- <http://www.texturepacker.com/>



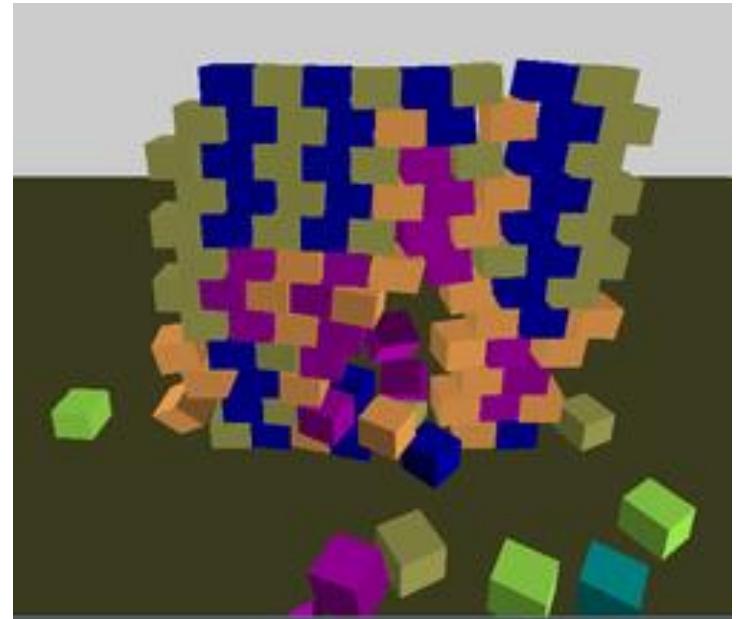
Collision detection

- Testing intersection
 - Camera-object
 - Object-object
 - Triangle-line
 - Triangle-triangle
- Using acceleration structures
 - BSP tree
 - Bounding volumes tree
 - Octree
- Physics engines



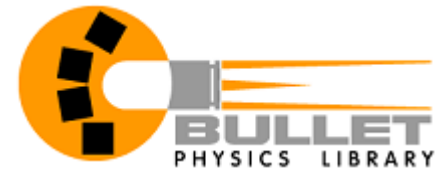
Collision detection

- Testing intersection
 - Camera-object
 - Object-object
 - Triangle-line
 - Triangle-triangle
- Using acceleration structures
 - BSP tree
 - Bounding volumes tree
 - Octree
- Physics engines



Physics engines

- C/C++ libraries for computation of collisions, simulations, movements, forces, clothes, ...
- Many platforms: PS3, Win, Linux, MAC OS X, 3DS max, ..
- **Bullet**
 - Open source, free to use
 - <http://bulletphysics.org/>
 - GPU acceleration
 - Grand Theft Auto IV, Red Dead Redemption, ..
- **Havok**
 - Proprietary/Shareware
 - <http://www.havok.com/>
 - Fallout 3, StarCraft II, Half Life 2, The Elder Scrolls IV: Oblivion, ..
- **PhysX**
 - Proprietary realtime physics engine
 - GPU and PPU acceleration
 - http://www.nvidia.com/object/physx_new.html
 - Mafia II, Batman: Arkham Asylum, Metro 2033, ..



Questions?

