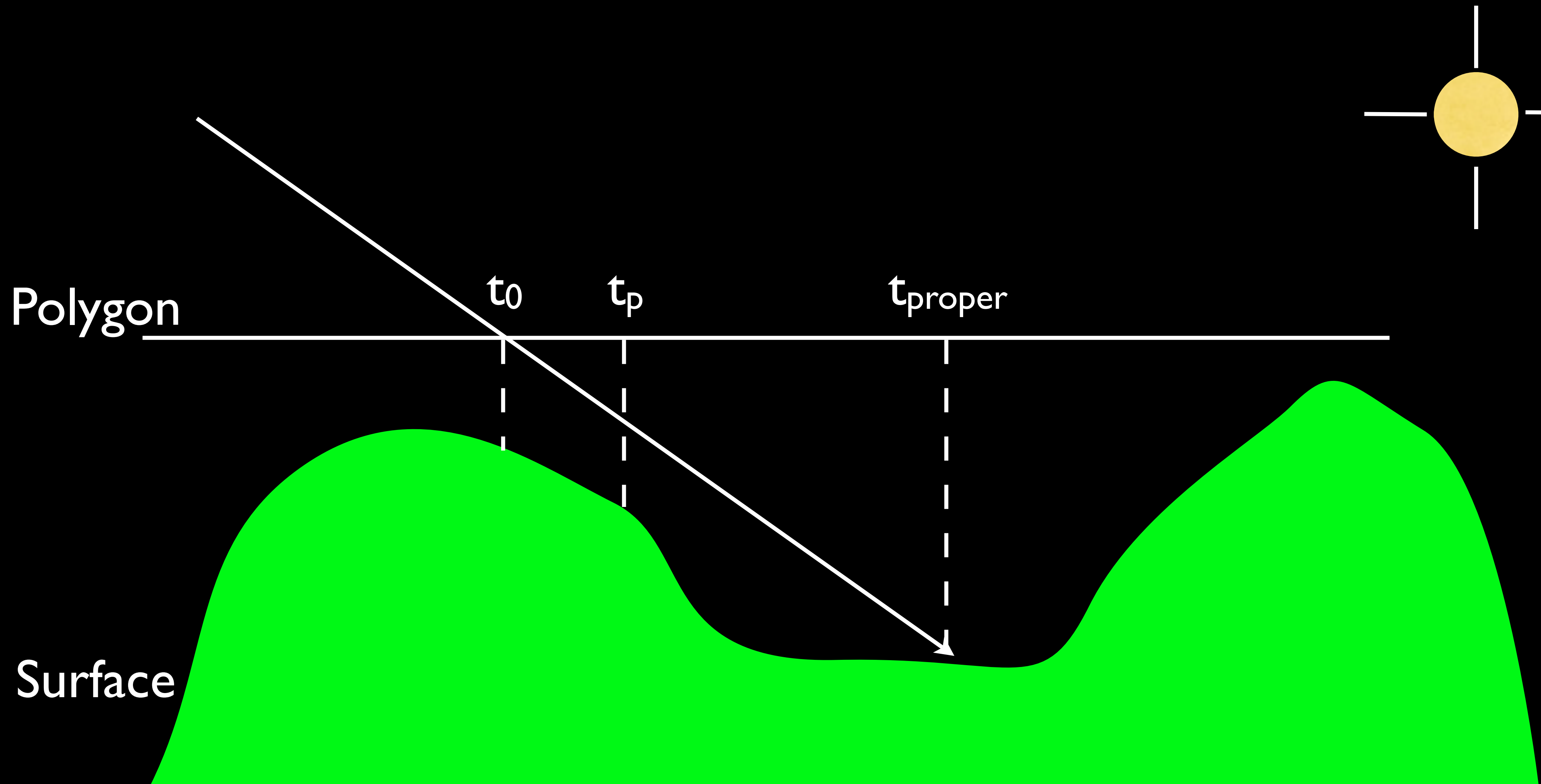
COMPSCI 715 Part 2

Lecture 3 - Parallax Occlusion Mapping

Parallax Occlusion Mapping

- An improvement over Parallax Mapping from Lecture 3
- In other words: another really cool way to trick the viewer into seeing things that aren't there
- Originally in 2004

POM - Problem



POM - Basics

- Uses per-pixel raytracing to find correct hitpoint
- Gives us:
  - Parallax
  - Complex Geometry Correct
  - Occlusion and silhouettes
  - Proper Lighting
  - Adaptive Level of Detail

POM - Comparison

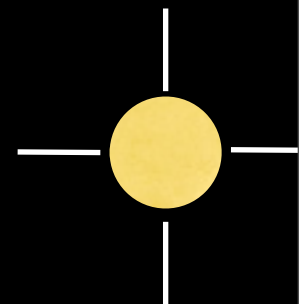
1100 polygons
14Mb memory
225fps

1500000 polygons
45Mb memory
32fps

POM - Implementation

- Use tangent space again
- Compute parallax offset vector
- Ray-cast along parallax offset vector
- Ray - Height field intersection
- Use piecewise linear curve instead of nearest neighbour (otherwise artifacts)

POM - Implementation



parallax offset vector

Polygon

Surface

POM - Implementation

- Ray casting:
  1. Fetch two samples μ apart
  2. Check if there is a possible intersection
  3. If not: continue with next sample μ along offset vector
  4. If so: compute intersection of linear segment and viewray

POM - Implementation

- Vertex Shader:

```
uniform vec3 parallaxDir;

parallaxDir =normalize(v.xy);
float parallaxLength = -sqrt(1.0 - v.z*v.z) / v.z;
float parallaxBias = -perspectiveBias;
parallaxDir = -parallaxDir * parallaxLength *
             parallaxBias * heightScale;
```

POM - Implementation

- Fragment Shader

```
float h0 = (1.0 - texture2D(colourMap,texCoord).a);
float h1;

float iter;
for (iter = 0.0; iter < MaxIterations; ++iter){
    texCoord -= parallaxDir*step;
    h1 = (1.0 - texture2D(colourMap,texCoord).a);
    if(h1 < iter*0.125)
        break;
    h0 = h1;
}
```

POM - Implementation

- Piecewise interpolation (instead of nearest neighbour:

```
for (iter = 0.0; iter < MaxIterations; ++iter){
    texCoord -= parallaxDir*step;
    h1 = (1.0 - texture2D(colourMap,texCoord).a);
    if(h1 < iter*0.125){
        float frac = clamp((iter-h0)/(h1-h0),0.0,1.0);
        texCoord -= parallaxDir*step*frac;
        break;
    }
    h0 = h1;
}
```

POM - Silhouettes

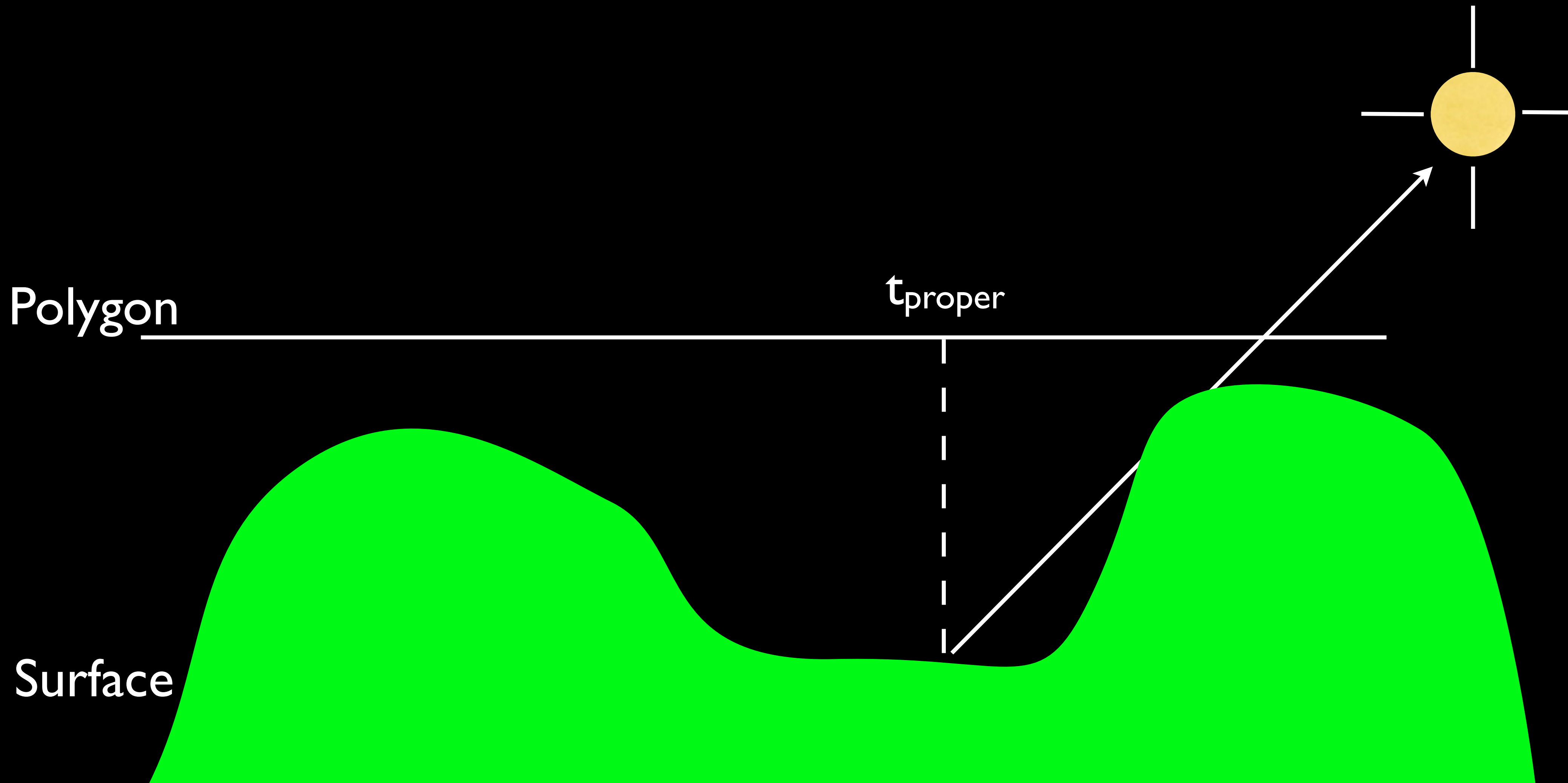
- Now with accurate offsets we can get silhouettes
- Simply discard all fragments that would have a texture coordinate outside our bounds

```
if(texCoord.x < 0.0 || texCoord.y < 0.0 ||
   texCoord.x > 1.0 || texCoord.y > 1.0)
      discard;
```

POM - Shadowing

- Need to find out if the point can see the light source
- Same problem in the reverse direction
  - So raycast again

POM - Shadowing



POM - Shadowing Impl

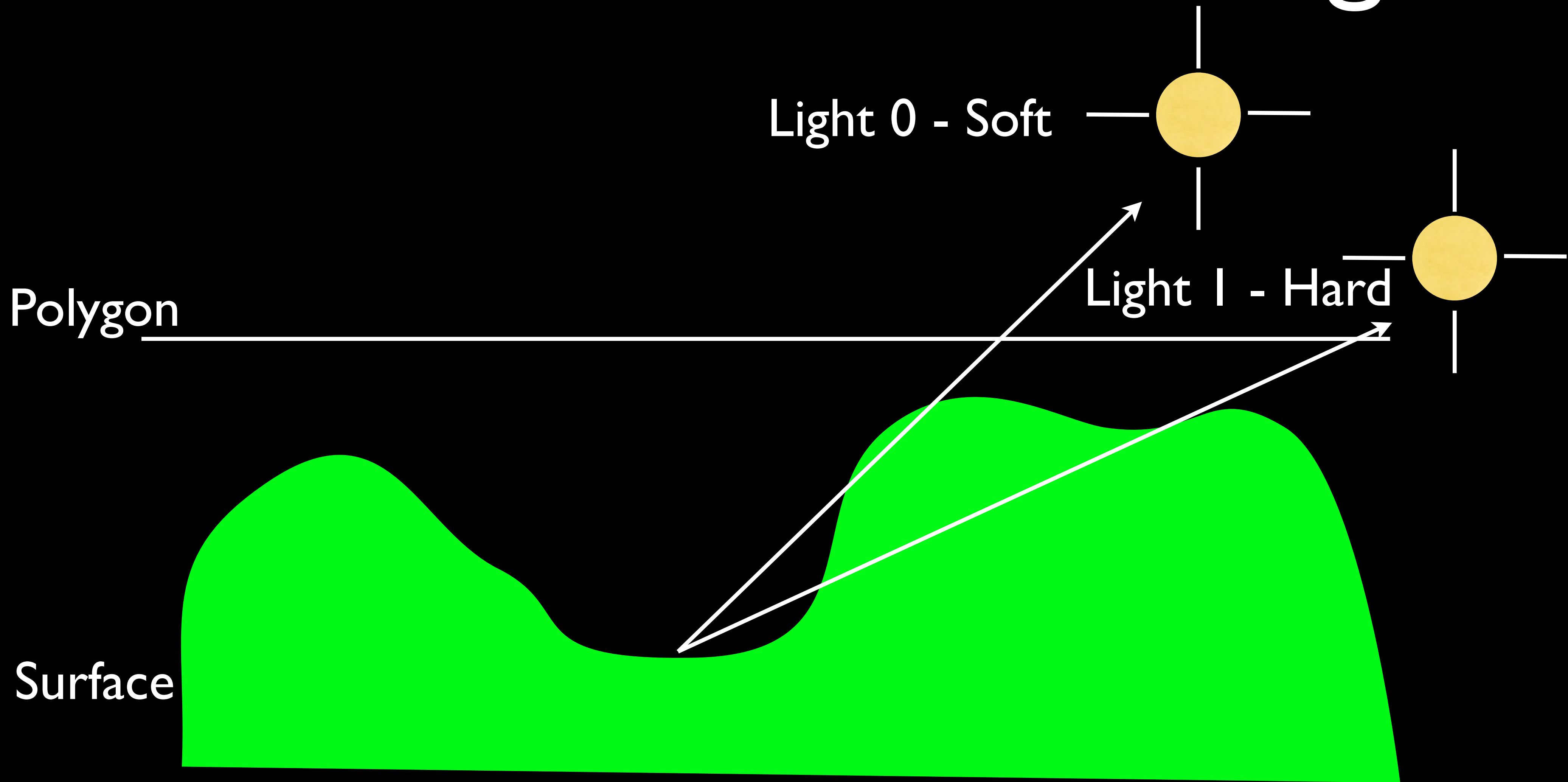
- Vertex shader:
  - Create shadowDir same as parallaxDir
- Fragment shader:

```
float shadow = 1.0;
vec2 lTexoord = texCoord;
for (iter = h0; iter > 0.0; iter-=iterStep){
  lTexoord -= shadowDir.yx*step;
  h0 = (1.0 - texture2D(colourMap, lTexoord).a);
  if(h0 > iter && h1 > iter){
    shadow = 0.2;
    break;
  }
  h1 = h0;
}
```

POM - Soft Shadowing

- What about penumbras?
- POM can emulate these by using an approximation:
  - Don’t stop raytracing until we leave the surface again and see how much is in the way

POM - Soft Shadowing



POM - Soft Shadowing Impl

- When calculating shadows count the number of iterations through to other side
- Use this with a heuristic to decide proper shadow level

POM - Soft Shadowing Impl

```
float shadow = 1.0;
vec2 lTexoord = texCoord;
for (iter = h0; iter > 0.0; iter-=iterStep){
  lTexoord -= shadowDir.yx*step;
  h0 = (1.0 - texture2D(colourMap, lTexoord).a);
  if(h0 > iter && h1 > iter){
    shadow -= 0.2;
  }
  h1 = h0;
}
```

POM - Improvements

- Bias to get rid of noise
- Dynamic sampling to lower aliasing noise
  - Use the angle between the surface normal and the view direction
- Smarter shadowing (use normal/lightDir test)
- Fix all those little bugs!

Sources

- Tatarchuk, N (2006). Practical Parallax Occlusion Mapping for Highly Detailed Surface Rendering. GDC 2006