Rendering

Triangle Rasterization in UE5

CS 415: Game Development

Professor Eric Shaffer

In UE5 Nanite

Small triangles are software rasterized

- This means a shader program running on the GPU does the work

Big triangles are hardware rasterized

- This means fixed function circuitry on the GPU does the work

Nanite Goals

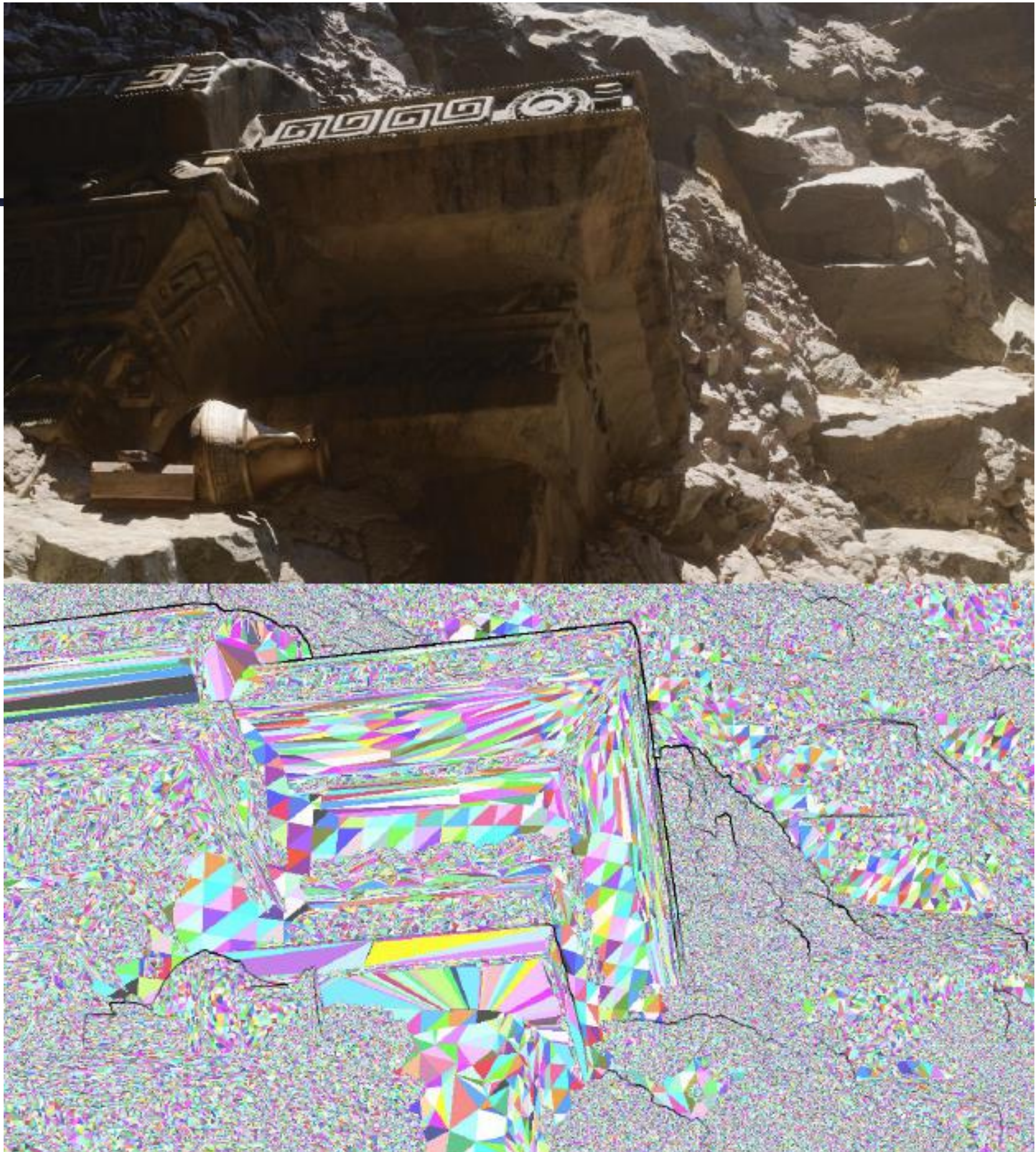
Pixel scale detail

- Use 1 triangle per pixel

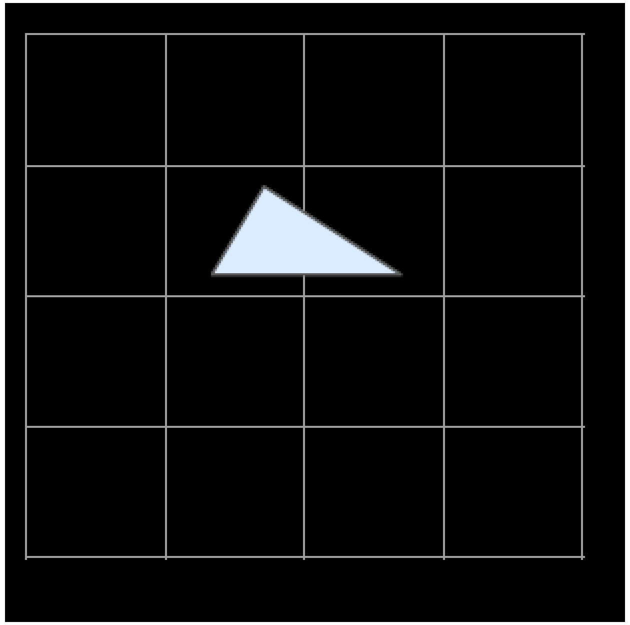
In practice

- A few large triangles
- Lots of tiny triangles

Why are some triangles still big in such a heavily optimized system?



Tiny Triangles



Terrible for typical rasterizer which is optimized for triangles covering many pixels

Typical rasterizer:

- Macro tile binning
- Micro tile 4x4
- Output 2x2 pixel quads
- Highly parallel in pixels not triangles

Modern GPUs setup 4 tris/clock max - not great for massive triangle counts

- Outputting SV_PrimitiveID makes it even worse (this is ID of which triangle generated the pixel)
    - …needed for visibility operations in Nanite

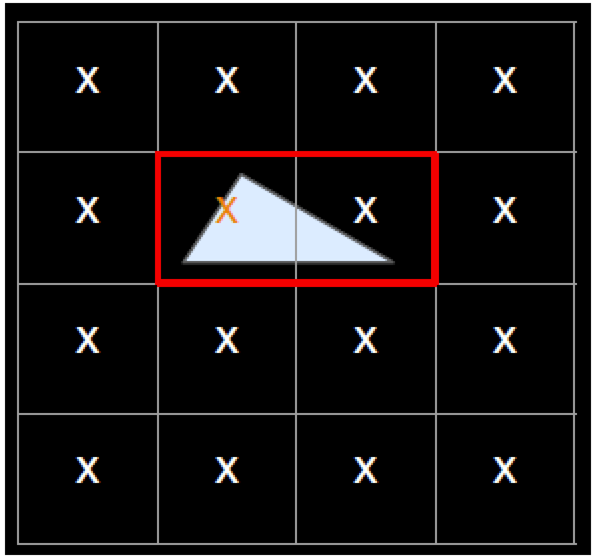
UE5 Nanite Micropoly Software Rasterizer

1 thread per vertex

- Transform position

1 thread per triangle

- Fetch indices
- Fetch transformed positions
- Calculate edge equations and depth gradient
- Calculate screen bounding rect
- For all pixels in rect
  - If inside all edges then write pixel



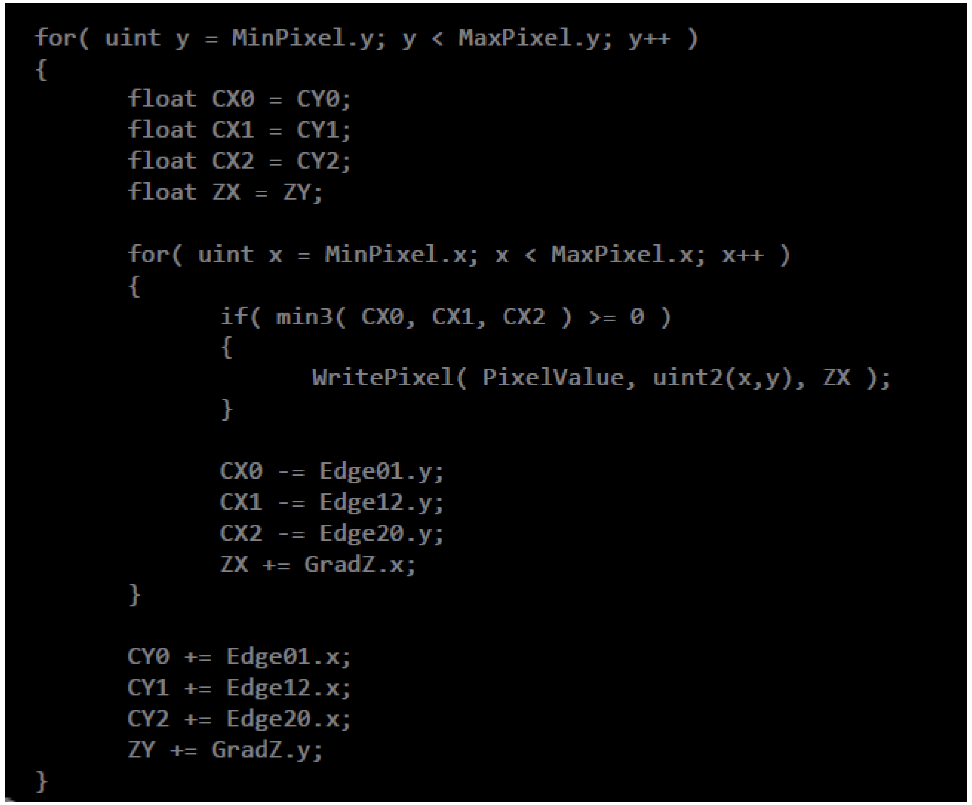
UE5 Nanite Micropoly Software Rasterizer

Thing to get from code is how little compute is used

All it does is:

- Iterates over the pixels in the bounding rectangle
- Tests if the center is inside all 3 edges

Micropoly rasterization was used in REYES..Pixar's first renderer in the 1980s…used in Star Trek II: The Wrath of Khan



```
for( uint y = MinPixel.y; y < MaxPixel.y; y++ )
{
        float CX0 = CY0;
        float CX1 = CY1;
        float CX2 = CY2;
        float ZX = ZY;

        for( uint x = MinPixel.x; x < MaxPixel.x; x++ )
        {
                if( min3( CX0, CX1, CX2 ) >= 0 )
                {
                        WritePixel( PixelValue, uint2(x,y), ZX );
                }

                CX0 -= Edge01.y;
                CX1 -= Edge12.y;
                CX2 -= Edge20.y;
                ZX += GradZ.x;
        }

        CY0 += Edge01.x;
        CY1 += Edge12.x;
        CY2 += Edge20.x;
        ZY += GradZ.y;
}
```

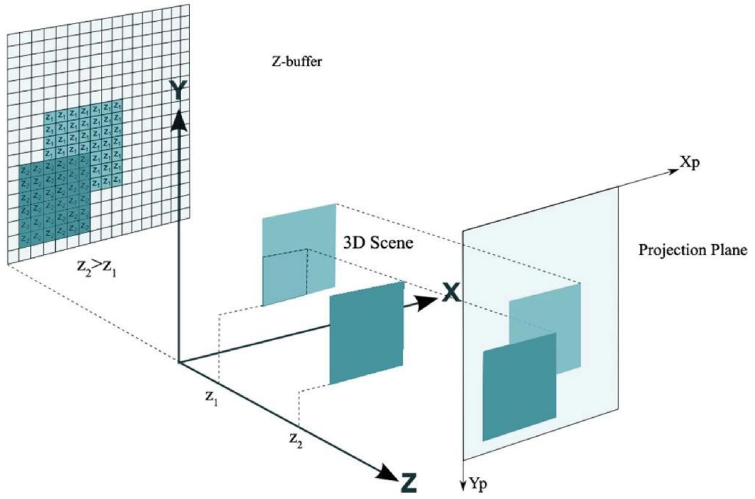
Software Depth Test

Depth test removes hidden surfaces

- Typically uses z-buffer algorithm

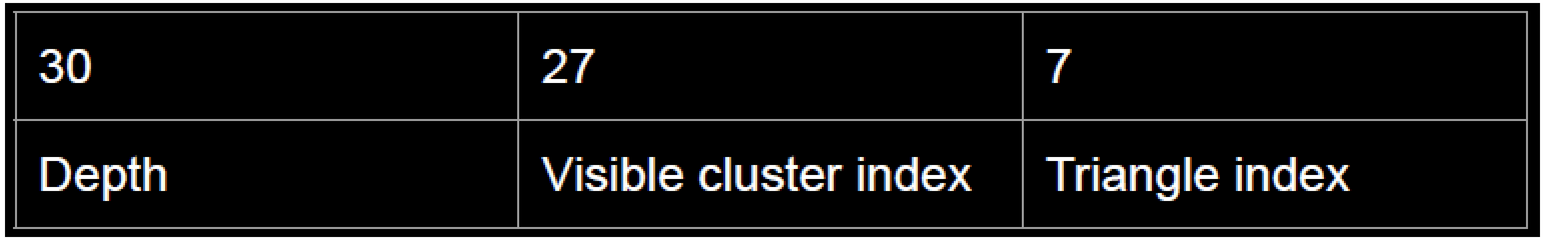
Nanite has to implement own visibility buffer

- Result of the rasterization strategy



Each entry in buffer corresponds to a pixel

Uses 64 bit atomics so no locking and loss of parallelism processing

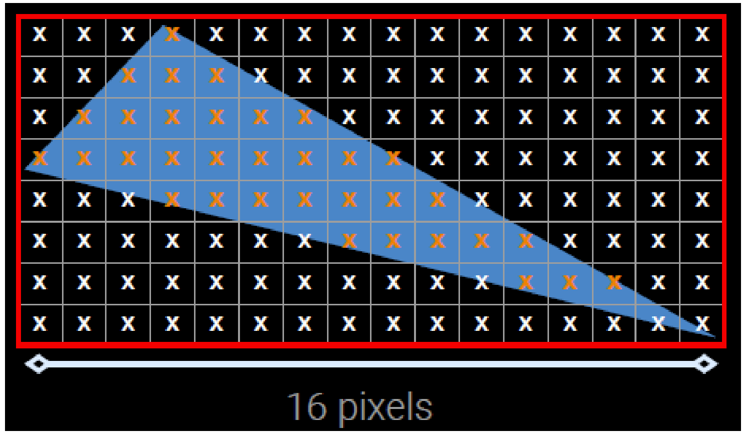


30	27	7
Depth	Visible cluster index	Triangle index

What About Big Triangles?

For each triangle cluster

- Choose software or hardware based on projected screen size
- Magic limit number is 32 pixels



Carefully following DirectX spec rasterization rules results in no cracks

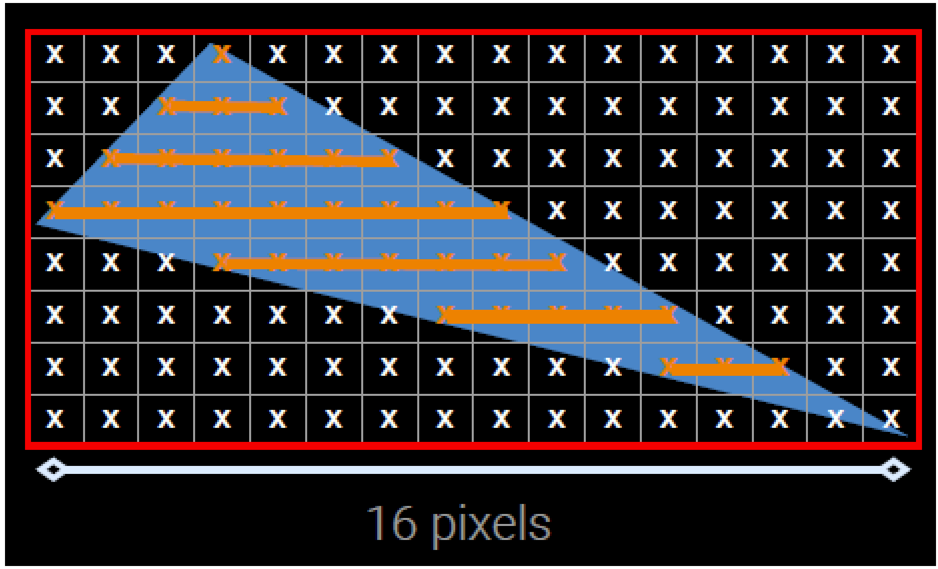
- Could have happen since HW and SW rasterization results must be combined

Visibility results also combined in the visibility buffer

What About Sort of Big Triangles?

Less than 32 pixels but greater than 4

Software rasterize but use a scanline algorithm



Don’t test each pixel against 3 edge equations

Calculate x span for each y coordinates and march

Very old school

Challenges with Instances

Artists using Nanite have chosen to use very high instance counts

An instance is a reference to a mesh and transform

Tiny instances should be merged –future work

- Hierarchical instances

For now, image based imposters (sort of like sprites) are used

An impostor is **a two-dimensional image texture that is mapped onto a rectangular card or billboard, producing the illusion of highly detailed geometry**. Impostors can be created on the fly or precomputed and stored in memory. In both cases, the impostor is accurate only for a specific viewing direction.