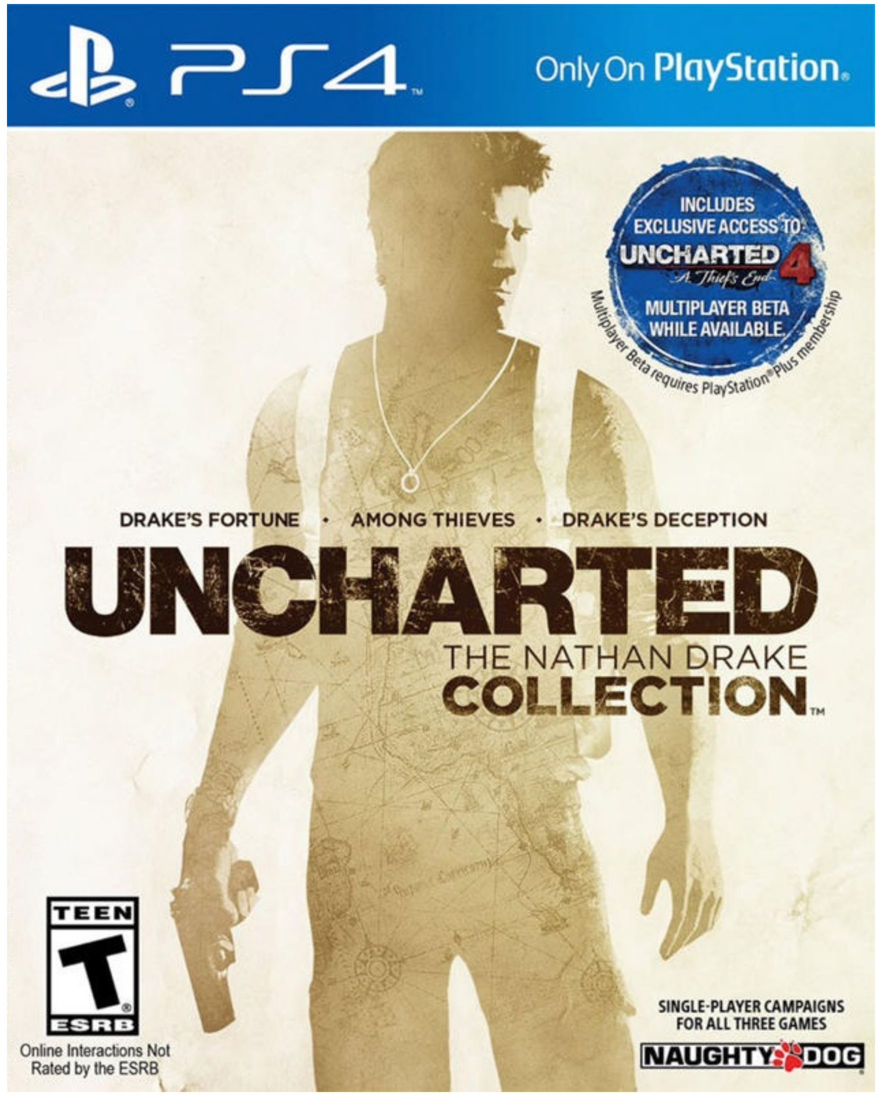
Lifting the Fog:

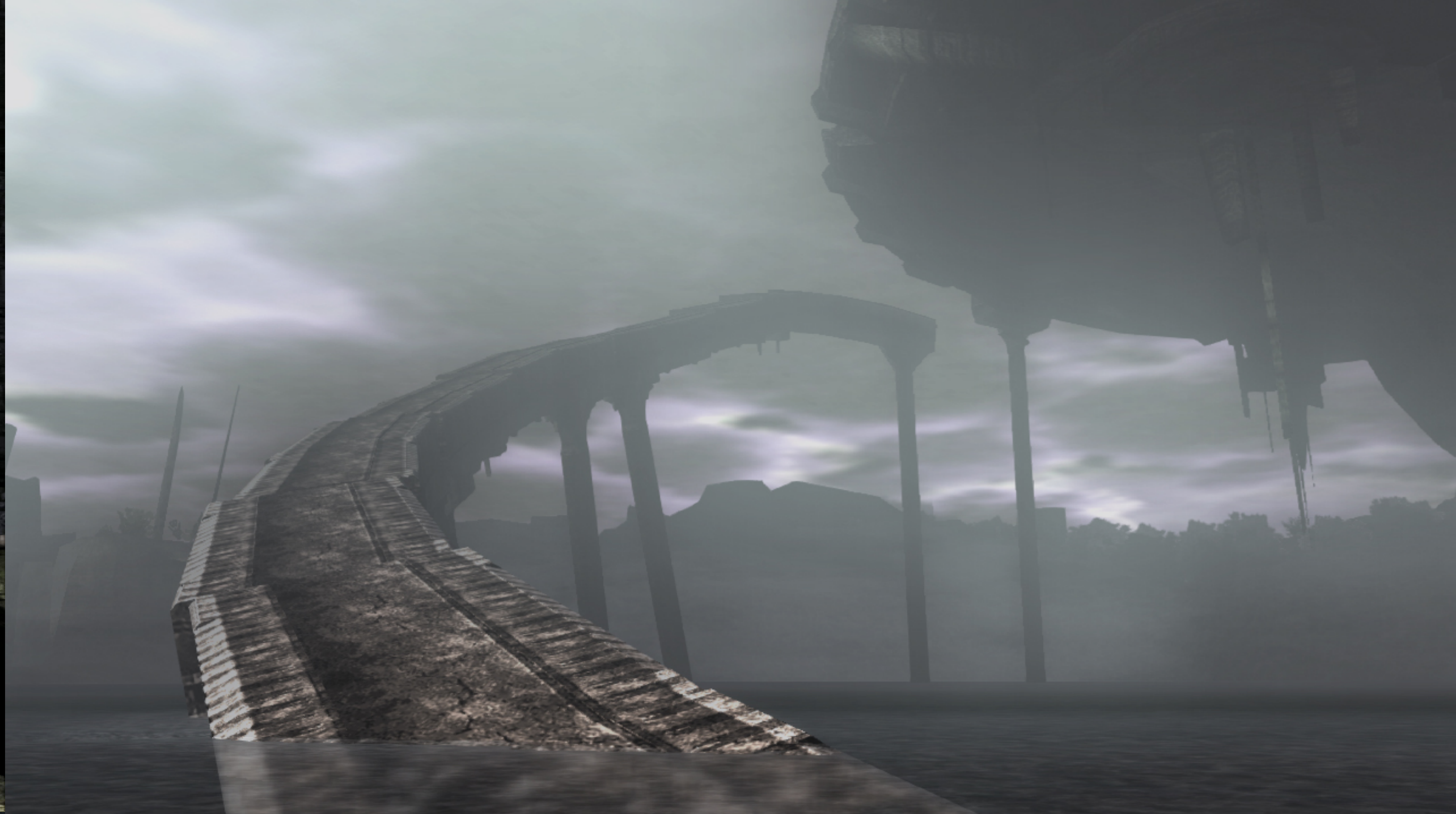
Geometry & Lighting in Demon's Souls

Bruce Woodard
Senior Programmer, Bluepoint Games

About Bluepoint






Tutorial
L
R
7 Crescent Moon Grass
10

Today’s Talk

- Part 1: Geometry
  - Compute-based tessellation
- Part 2: Lighting
  - Global illumination
  - Screen-space shadowing

Part 1: Geometry

Tessellation

Hardware Tessellation

- Too slow
  - Poor performance with high and low factors
  - Bottlenecks in shader pipeline
  - Pass redundancy

Compute Tessellation

- Goals
  - Multi-pass reuse w/ reasonable footprint
  - Scheduling flexibility
  - Optimal rendering of non-tessellated triangles

Shadow of the Colossus Approach

- Full attribute caching
  - Position, UV, normal, tangent, etc.
  - 44+ bytes (floats)
  - Just interpolated base triangle data!

- SubD vertex
  - Base triangle index
  - Barycentric coordinates
  - 4 bytes

SubD Vertex Rendering

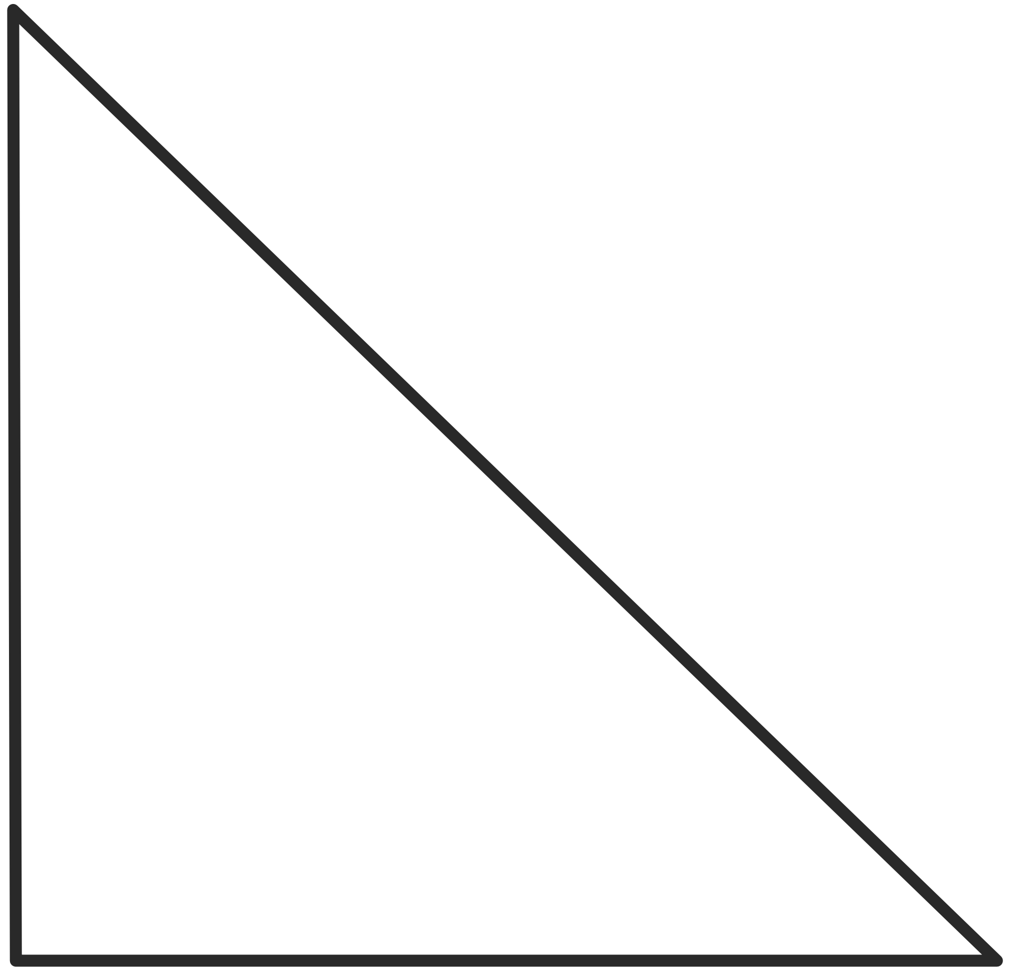
Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SV_VertexID

SubD Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, … }



SubD Vertex Rendering

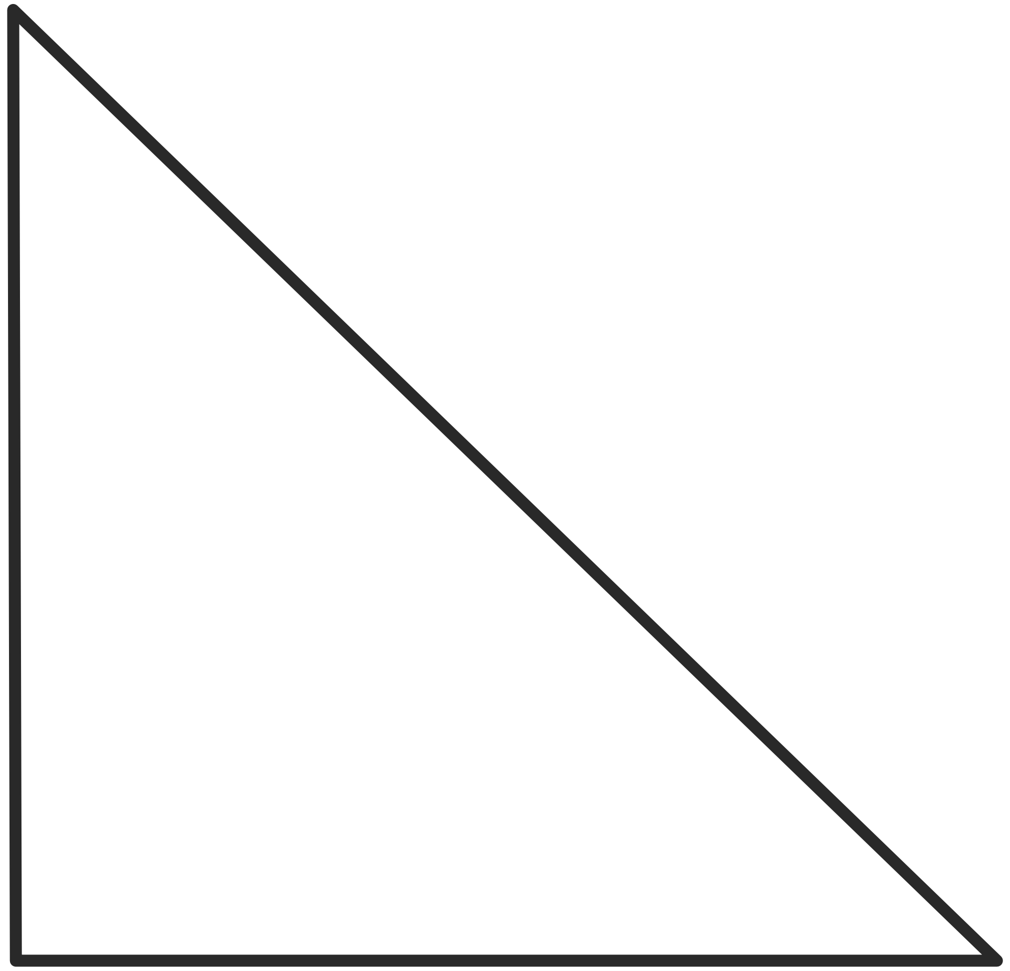
Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SubD Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, … }

Base Triangle



SubD Vertex Rendering

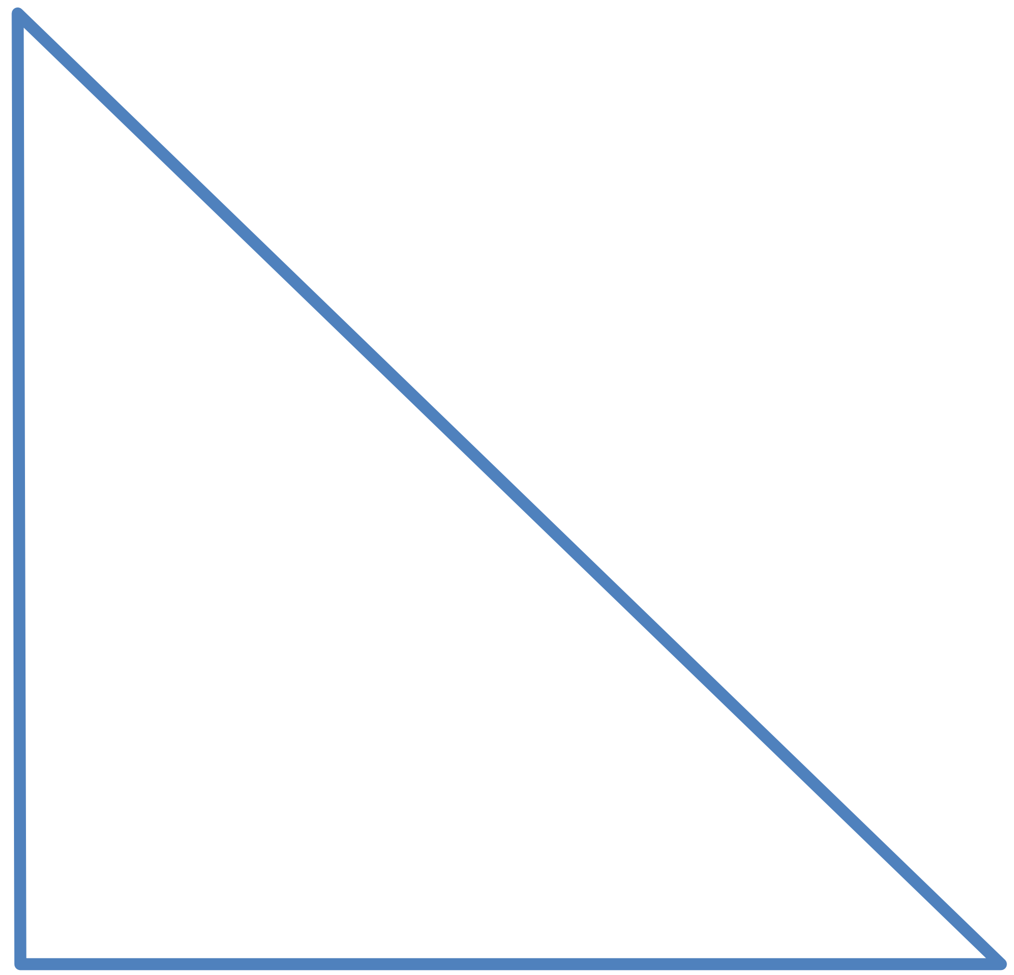
Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SubD Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, … }

Base Triangle



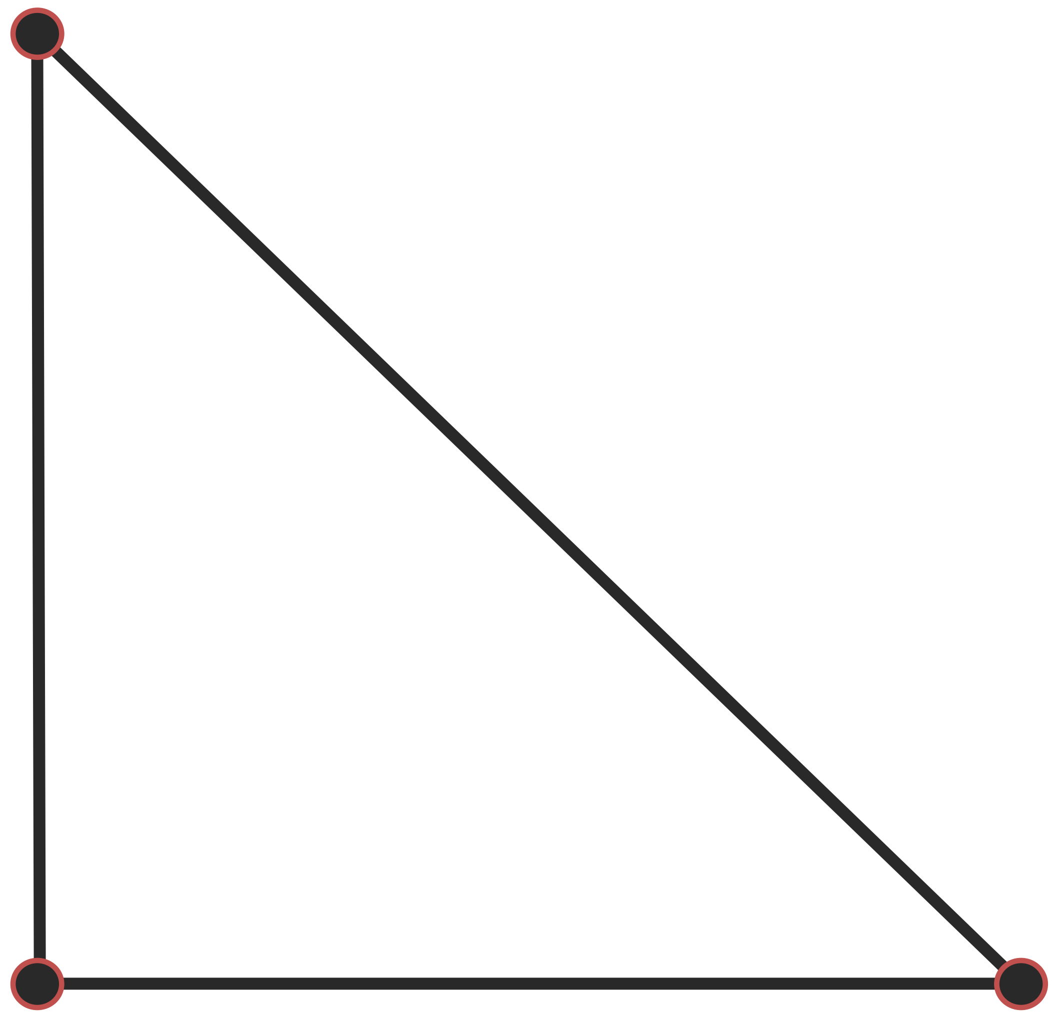
SubD Vertex Rendering

Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SubD Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, … }



SubD Vertex Rendering

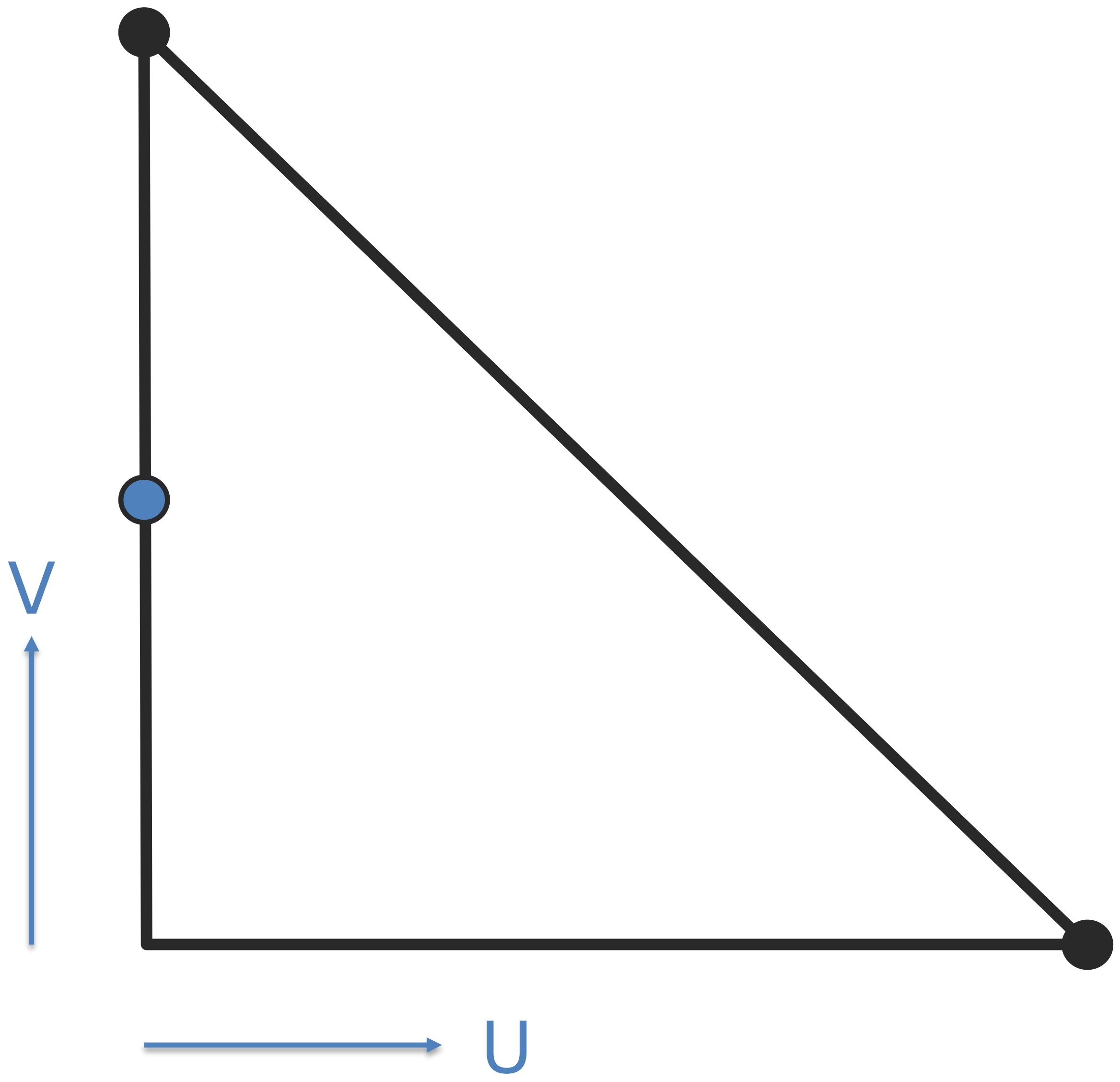
Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SubD Triangle Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, … }

Barycentric UV



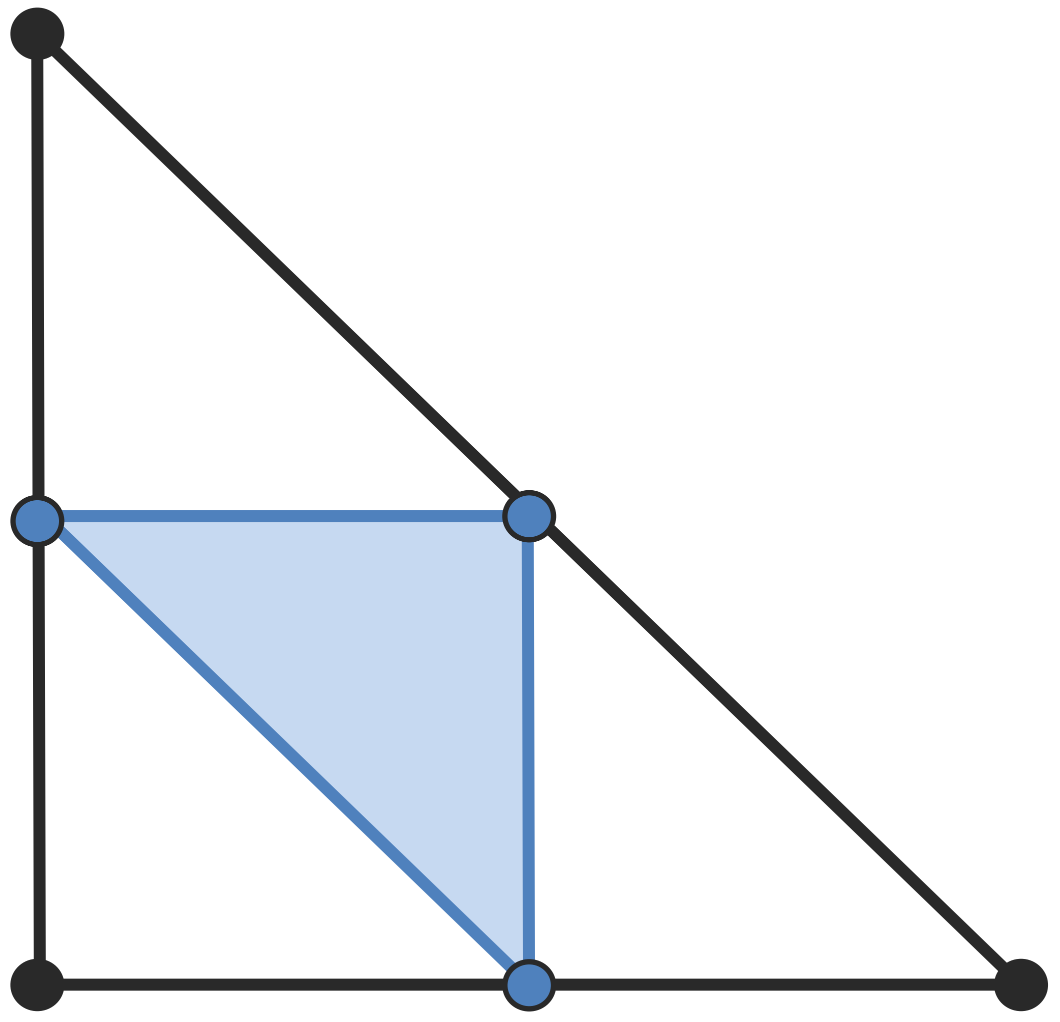
SubD Vertex Rendering

Base Verts = { {0,1}, {0,0}, {1,0} }

Base Indices = { {0, 1, 2} }

SubD Triangle Indices = { …, {0, 2, 3}, … }

SubD Vertices = { {0, 0, 0.5}, {0, 0, 0}, {0, 0.5, 0} }



Compute

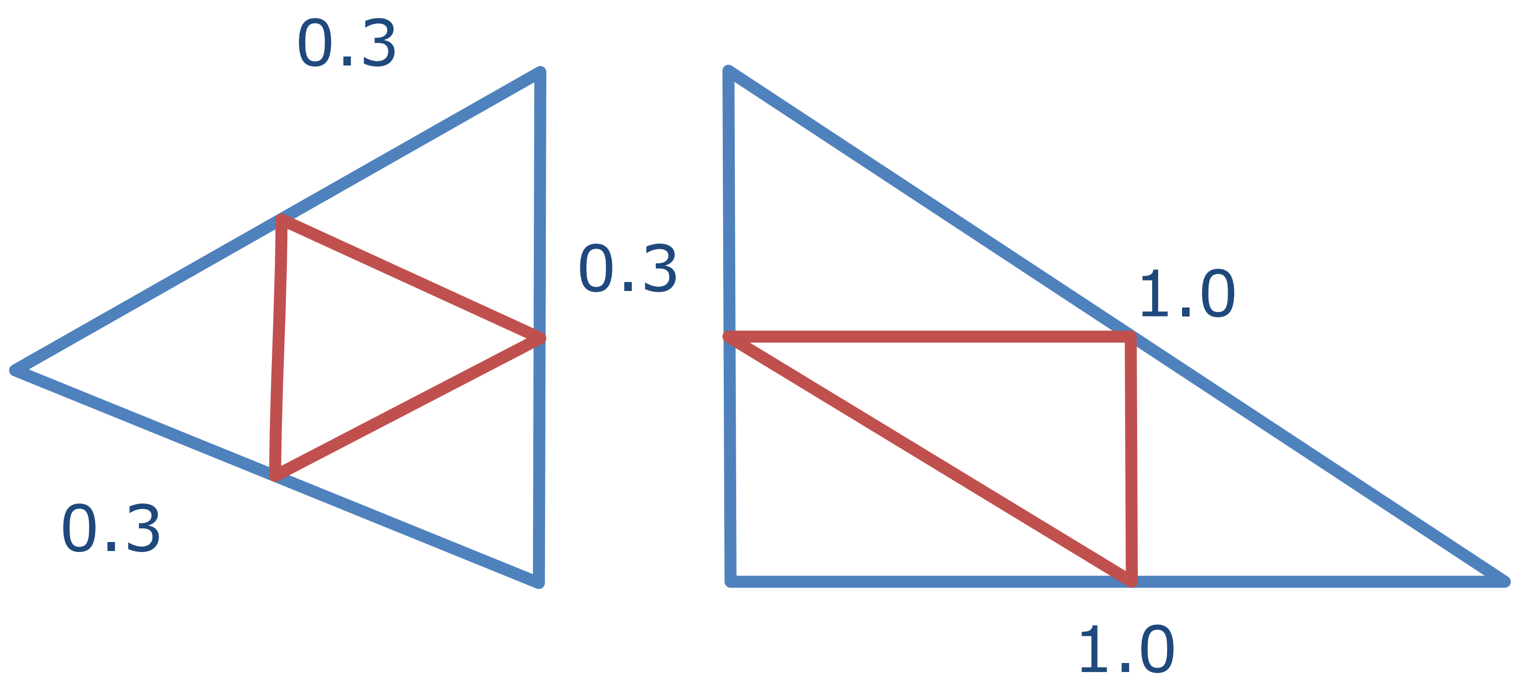
- One thread per base triangle
- Cull to frustum
- Pick tessellation level of detail
  - Edge LOD selected by closest vert distance
  - Triangle LOD selected by highest edge LOD

Vertex/Index Generation

- Atomic allocate verts & indices
  - Counters stored in IndirectDraw
  - Vertex/Index memory reserved for worst case
- Topology fixed per LOD
- Indices reused within tessellated triangle

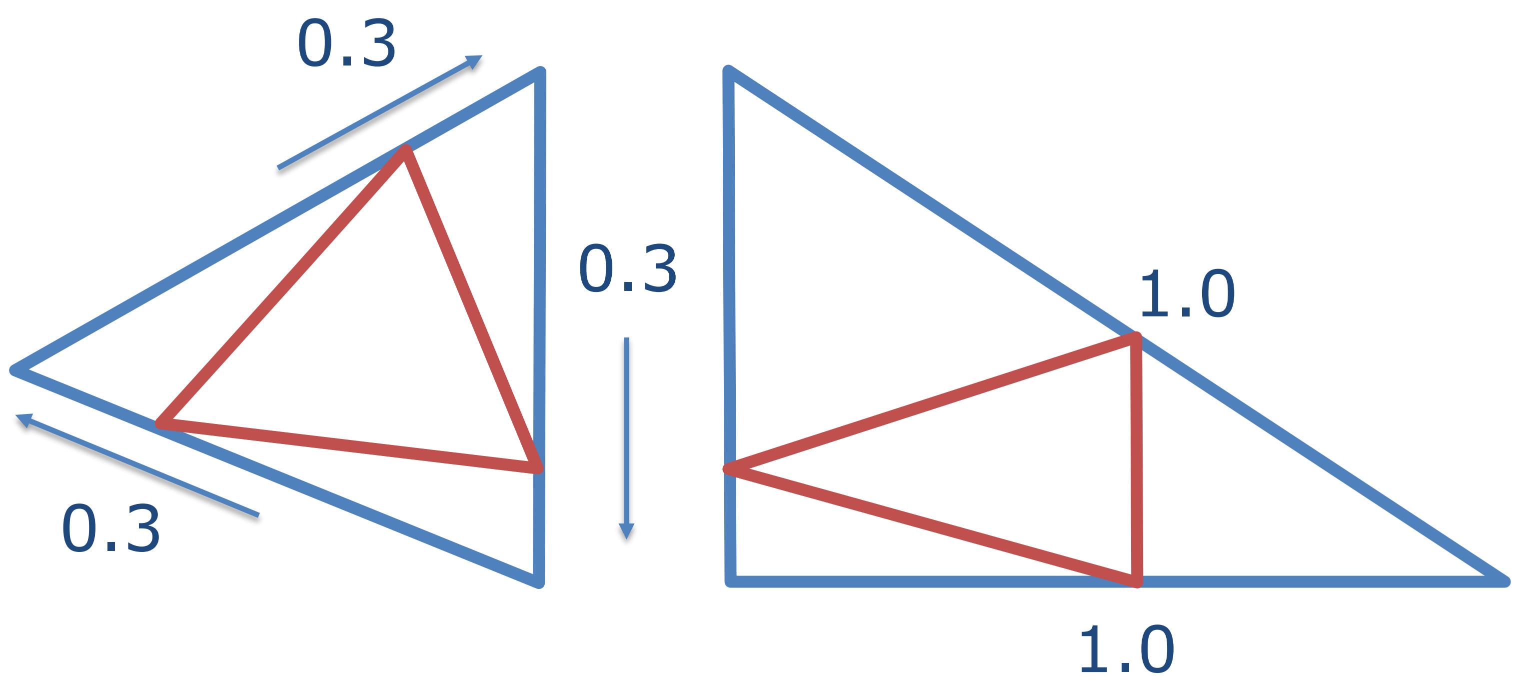
Morphing

- Triangle LOD = max of ceil(edgeLOD)



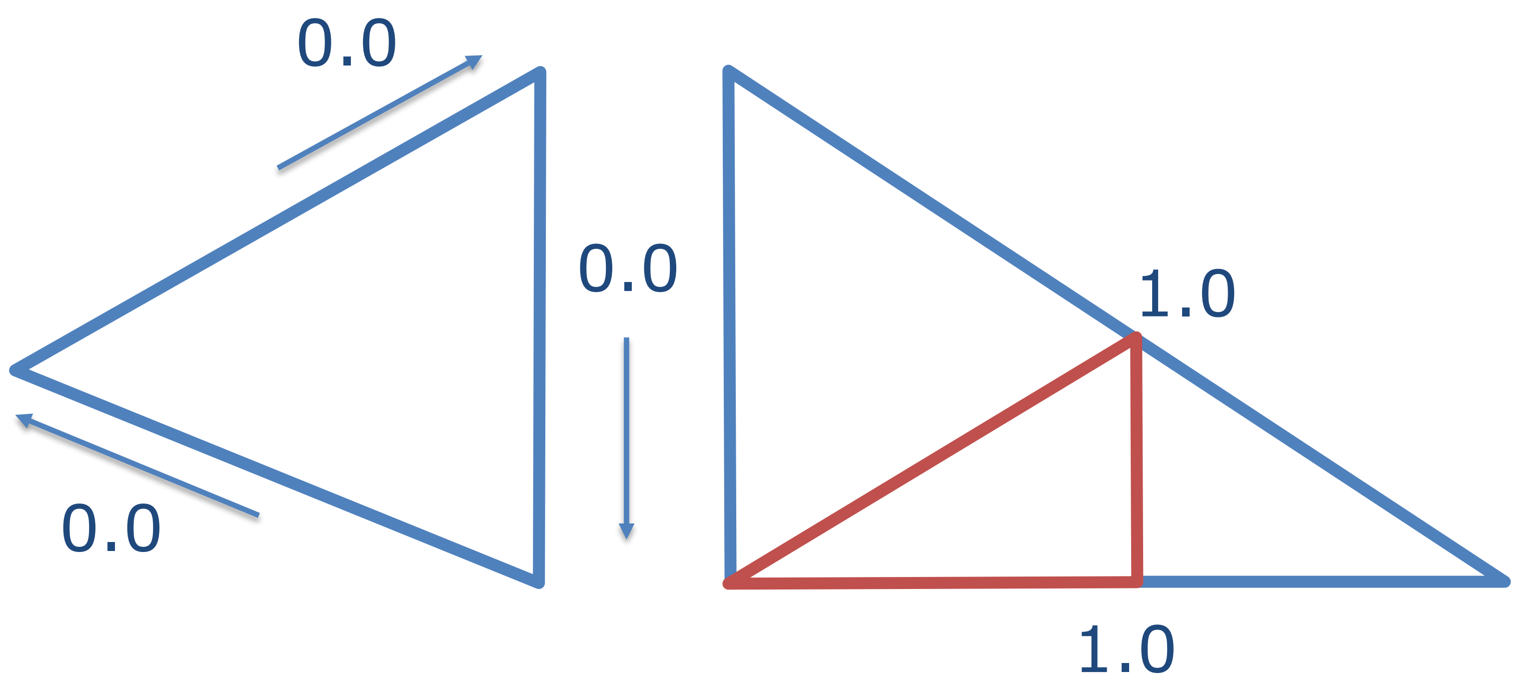
Morphing

- Morph barycentric coordinates using edge LOD



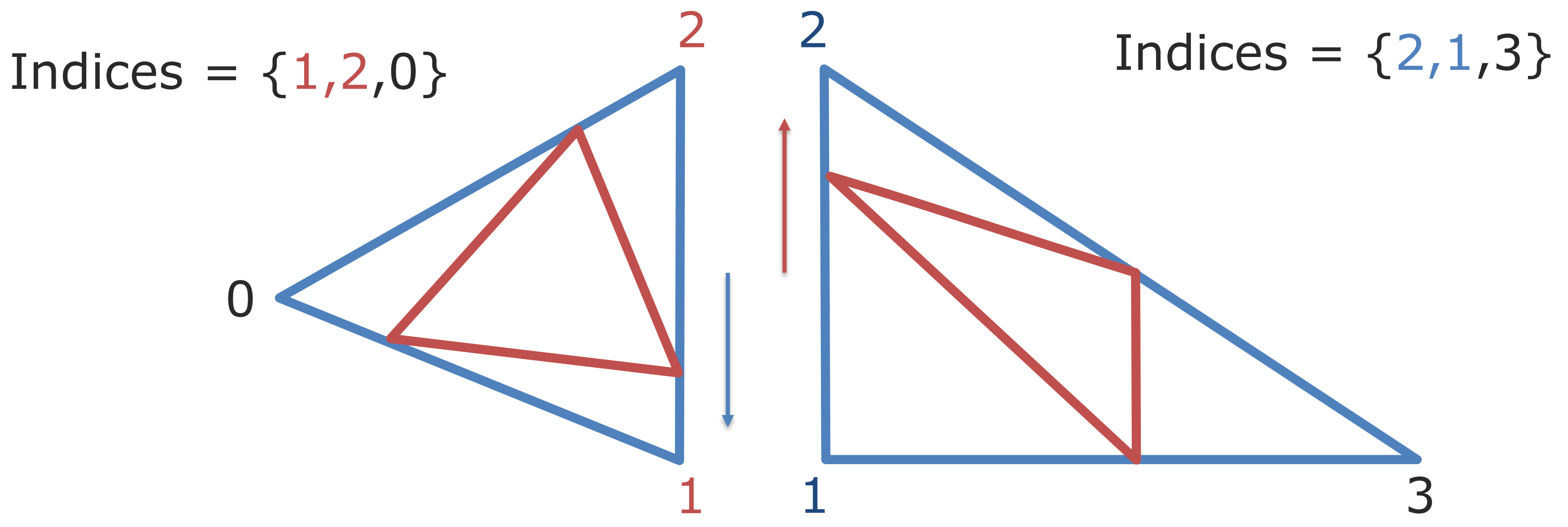
Morphing

- Morph barycentric coordinates using edge LOD



Morphing

- Choose morph direction using index order



Limitations

- Limited LOD levels (3)
  - LOD0: 3 verts, 3 indices
  - LOD3: 45 verts, 192 indices
  - LOD6: 2k verts, 12k indices --- NOPE!
- Morphing limited to 1 LOD difference
  - Range based LOD bad for varied triangle sizes
- Redundant displacement per pass

Ship It

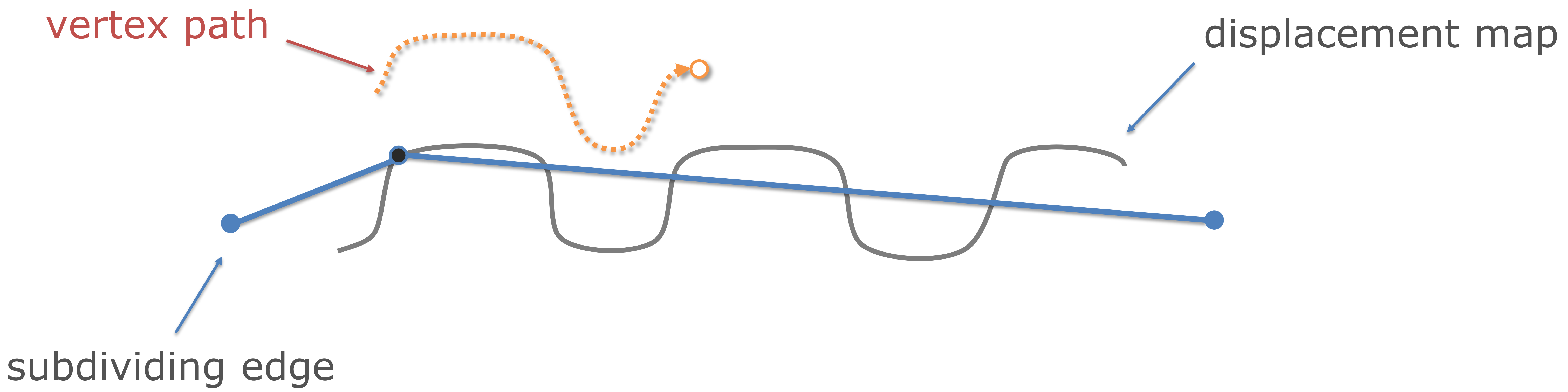
- Worked well enough for intended usage
  - Water
  - Bonus: Sand Arena

Demon's Souls

New Goals

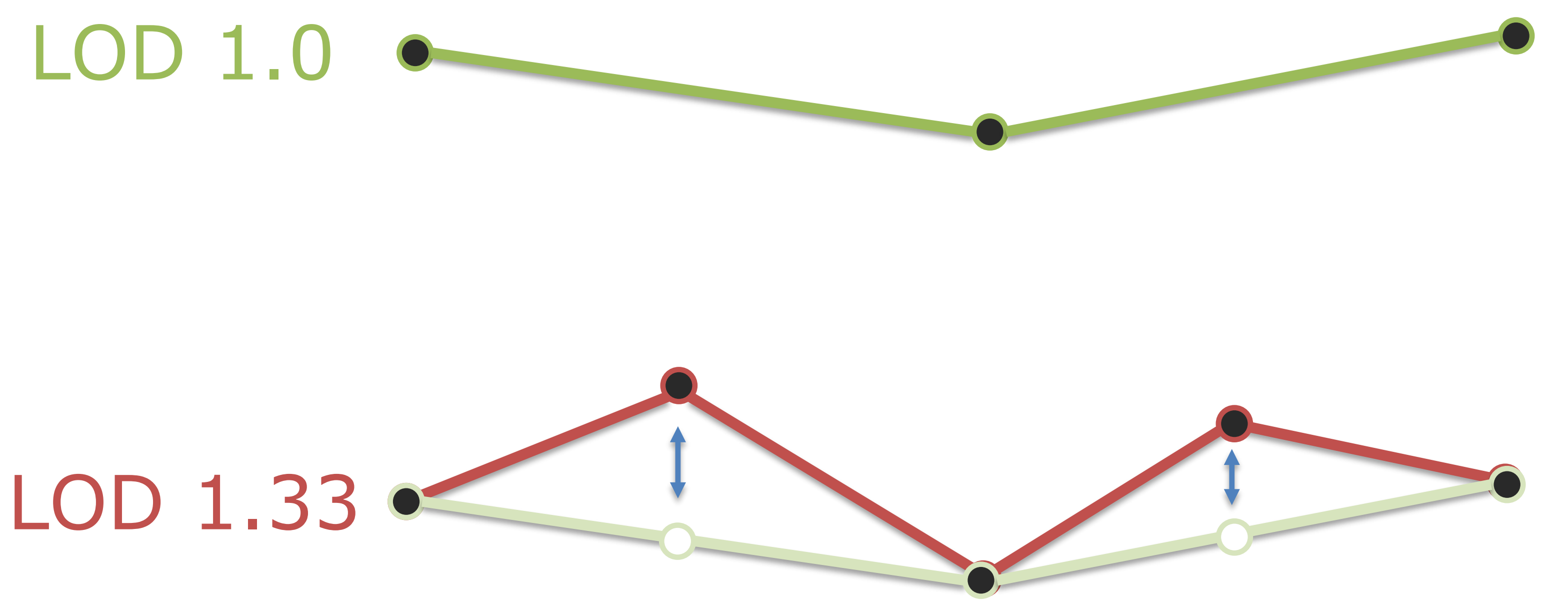
- *Much* Higher Density
  - 7 LODS (1 tri -> 16k tris)
  - 2.5m edge -> 2cm edge
- Faster
  - Distribute Compute Workload
  - Cache Layered Material Displacements
- Reduce Visible Morphing Artifacts

Edge Morphing

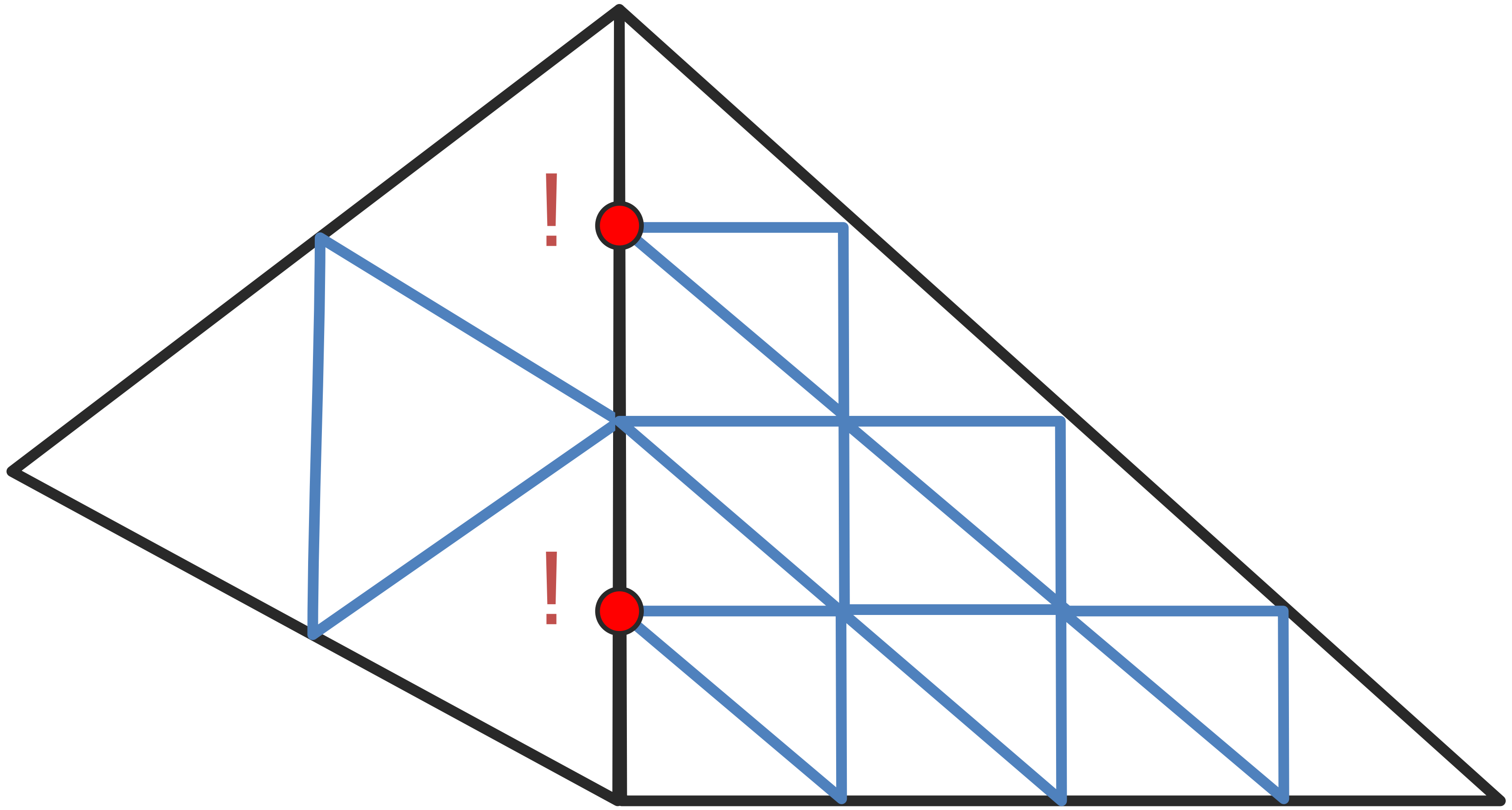


Extrusion Morphing

- Pop LODs
- Morph from neighbor midpoint



T-Junctions




Meh

T-Junctions

- Overblown
  - Morph boundary verts to match neighbor edge
  - Just a few pixel sized cracks
  - Invisible after antialiasing
- Does mean holes in Z tiles / classifier tiles!
- But seriously… <12 pixels a frame

SubD Vertex 2.0

- Expanded SubD Vertex
  - 3D Displacement (+/- 1 meter): +6 Bytes
  - Larger Base Triangle Index: +2 Bytes
  - Total: 4 Bytes -> 12 Bytes

Compute Passes

- Cull & Tally
  - Cull triangles (inc. occlusion & loose backface)
  - Compute LODs
  - Tally space & work requirements
- Allocate & Prep Indirect
  - Reserve contiguous space from global pools
  - Prep indirect compute / draw
- Setup Work
  - Generate worker thread assignments (triangle & sub-part)
- Tessellate
  - Generate indices & SubD vertices with material displacements
- Morph
  - Blend interior verts toward triangle LOD-1
  - Blend edge verts toward edge LODs

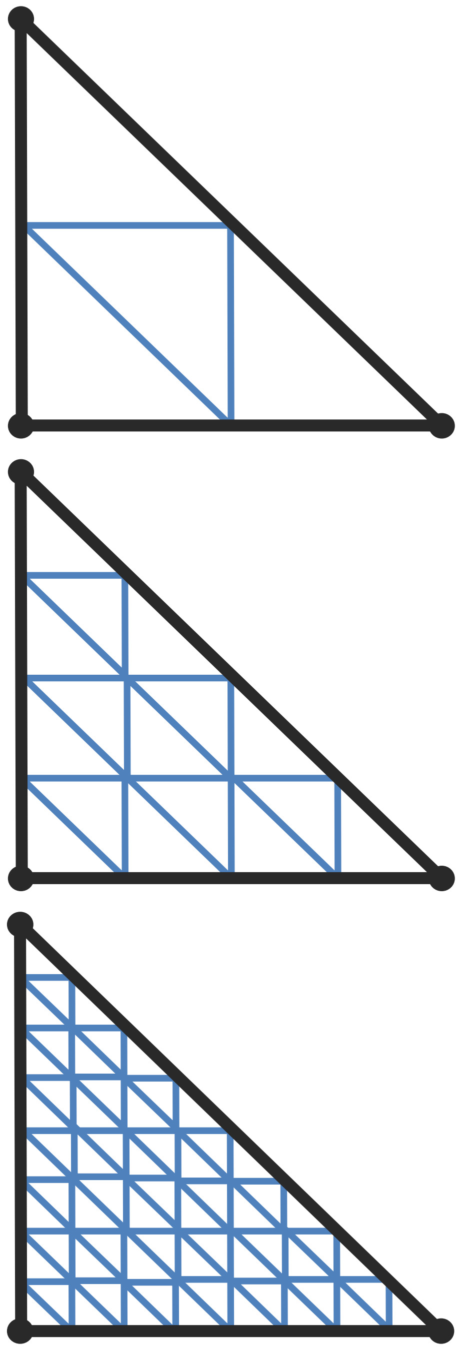
Parallelization

- Gridded triangle
  - Derive properties form edge LODs…

```
void InitTriangleProperties(float3 edgeLods)
{
    mTriLod = min(uint(ceil(max3(edgeLods.x, edgeLods.y, edgeLods.z))), kMaxLod);

    mNumEdges = 1 << mTriLod;
    mNumEdgeVerts = mNumEdges + 1;

    mNumVerts = (mNumEdgeVerts * mNumEdgeVerts + mNumEdgeVerts) >> 1;
    mIndexCount = 3 << (mTriLod * 2);
}
```

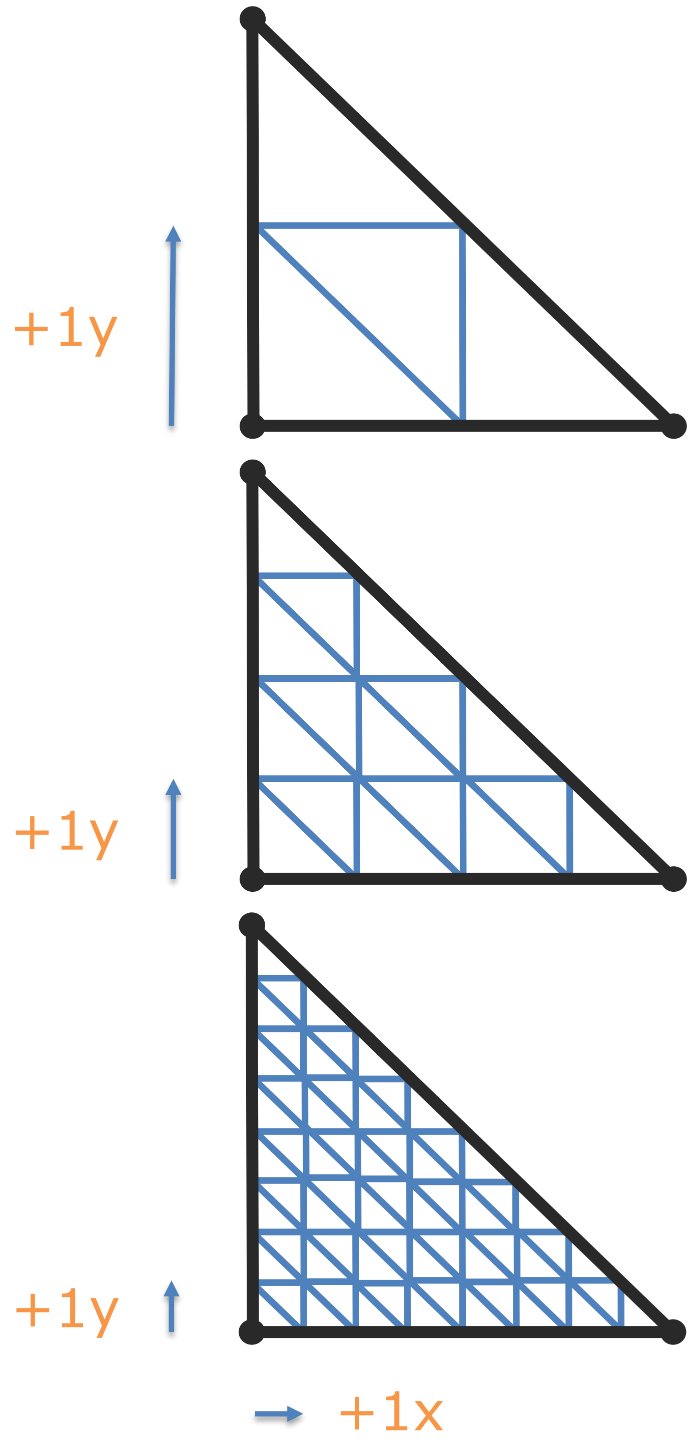


Parallelization

- Gridded triangle
  - Map 2D grid to 1D output…
    - Write location for verts & tris
    - Read locations for morphing

```
uint GetVertexIndex(uint x, uint y)
{
    return x + mNumEdgeVerts * y - ((y*y - y) >> 1);
}

uint GetTriangleIndex(uint xTri, uint y)
{
    // note: there are 2 xTris per x (except on diagonal)
    return xTri + (mNumEdges * 2 – 1) * y - ((y*y - y) >> 1);
}
```

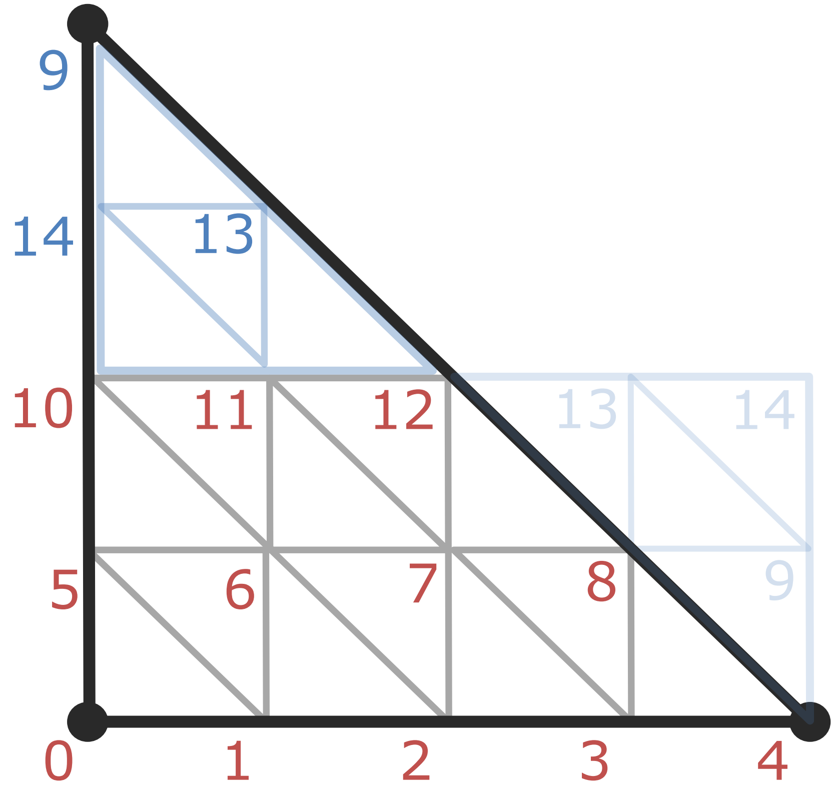


Parallelization

- Gridded triangle
  - Map 1D indices to 2D grid coords
    - Worker thread ID to target vert / tri

```
uint2 GetUV(uint index)
{
    uint w = mNumEdgeVerts;
    uint y = index / w;
    uint x = index - w * y;

    // if we've crossed the diagonal, flip to the upper half of the triangle
    return x < w-y ? uint2(x, y) : uint2(w-x-1, w-y);
}
```



Usage

- Ground, walls, water, rubble
  - Some animated materials
  - PN smoothing sometimes
  - Displacement discontinuities locked with vertex color
  - Ramped out after 15 meters
- Props, plants
  - Used early on, but backed off
  - Played it safe for perf, drew a line in the sand for art
- Characters
  - Didn't use, but works well
  - Read base verts from skinning output buffer

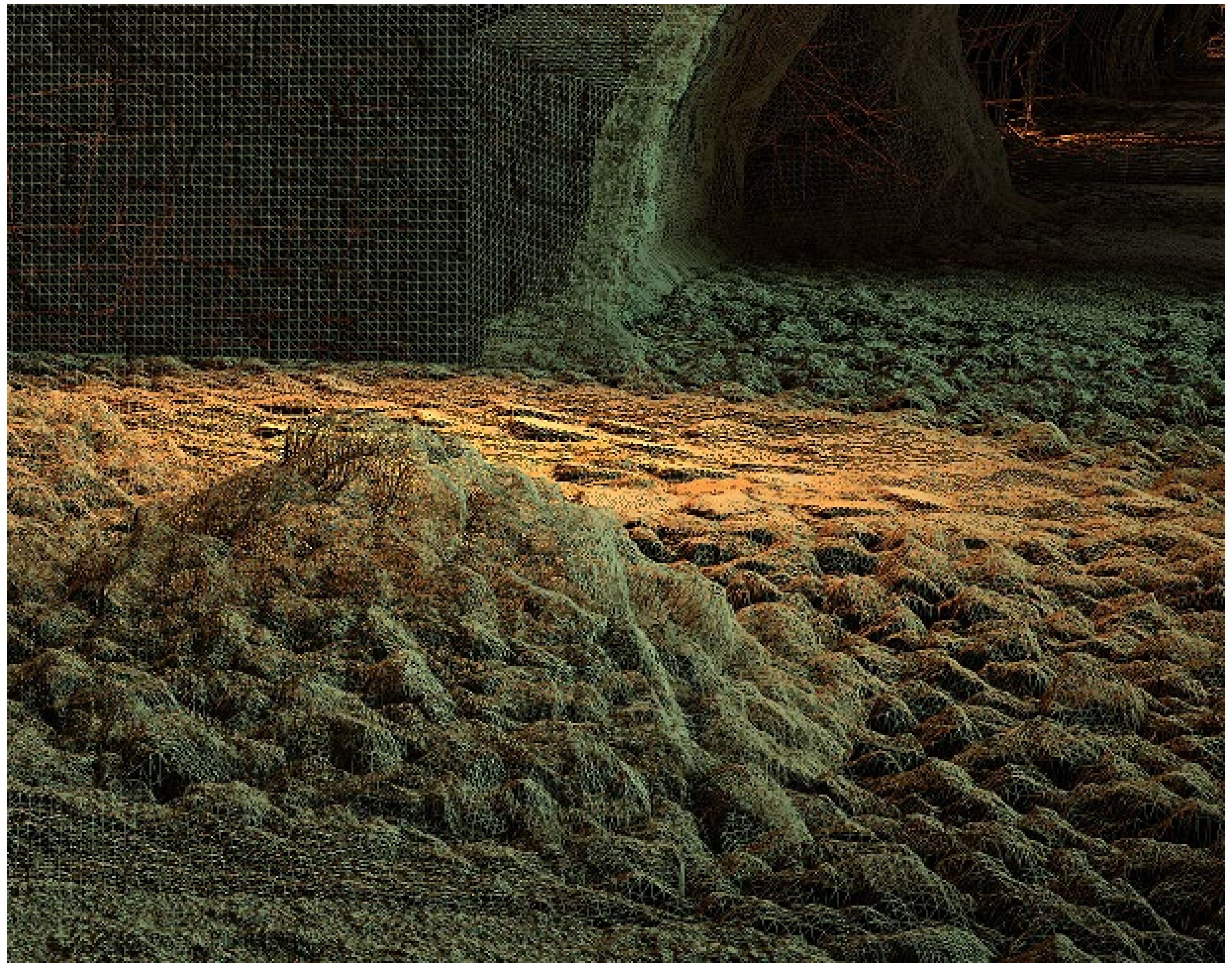
Usage

- Shadows / GI
    - Rendered without tessellation
    - Forced min displacement on base verts

- Collision
    - GPU raycasts for IK
    - Raycast finds barycentric coord on base mesh
    - Coord used to sample material displacement
    - Bonus: wetness, layer surface types/amounts for FX

Perf (PS5)

- Compute
  - 150µs for 1440p mode; 600k tris
  - 300µs for 4k mode; 1.3M tris
  - Depends on material complexity



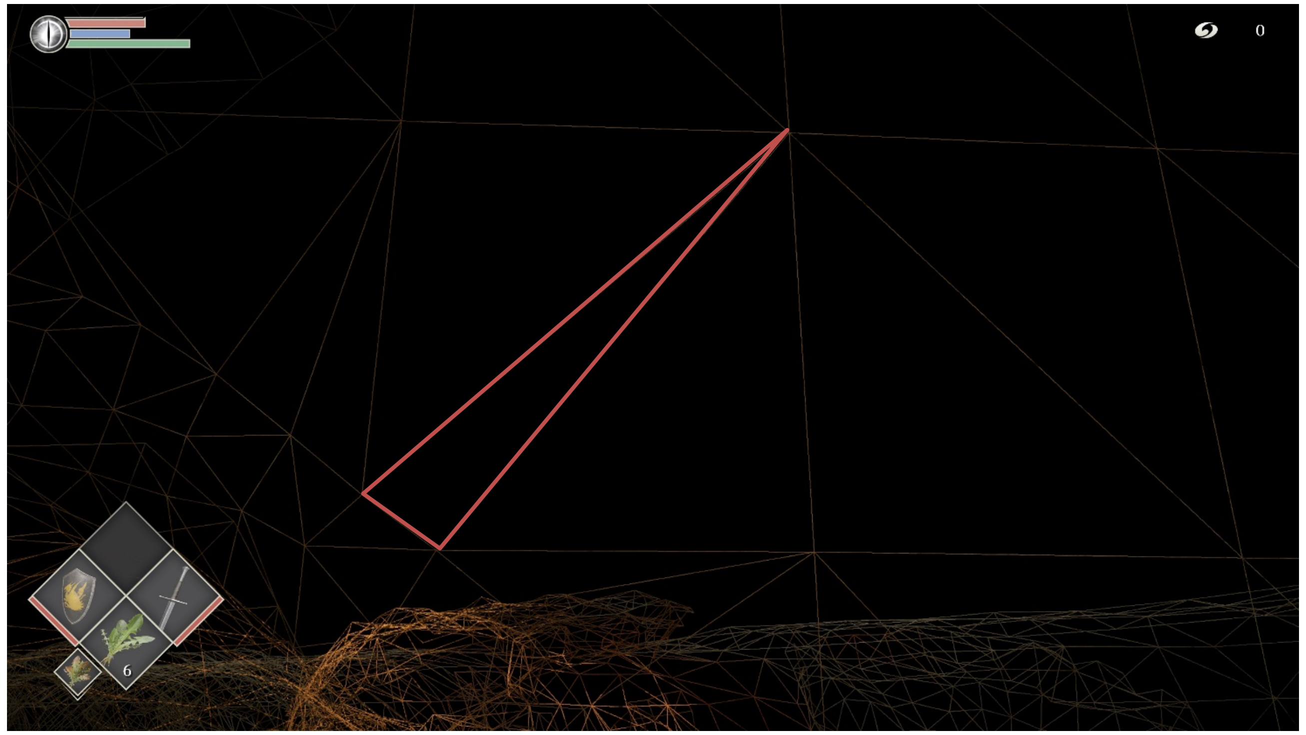
- Render (depth + gbuffer)
  - +200µs 1440 mode; 600k tris
  - +700µs 4k mode; 1.3M tris
  - Depends on triangle coverage & material complexity

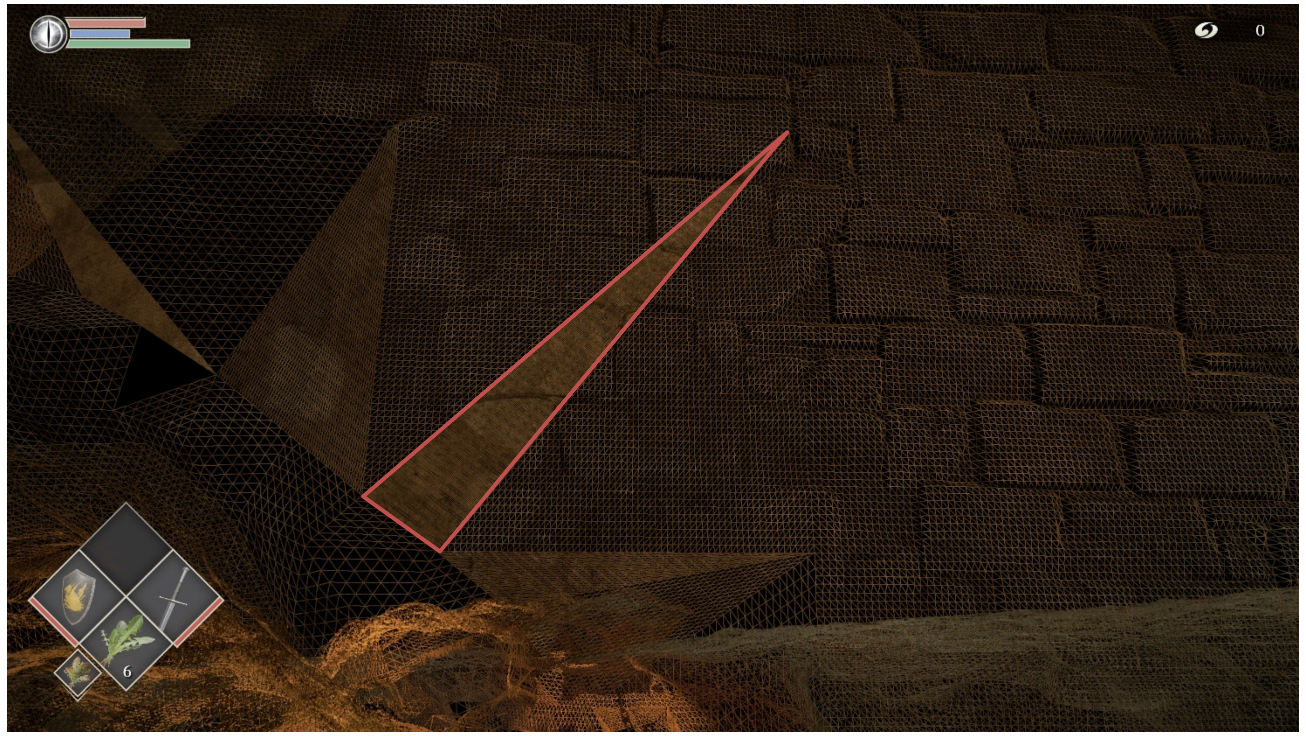
Memory

- 144 MiB of global buffers
  - 60MiB Subd Verts
  - 84MiB Indices
  - Typically < 30% usage @ 4k
- Very heavily padded at last minute
  - 4k testing was fairly neglected
  - Unnoticed slivers were causing overflow / dropout

Limitations

- Slivers
  - LOD is based on longest edge
  - Short edges get over-tessellated
  - Addressed manually
  - Tools to visualize





Future Work

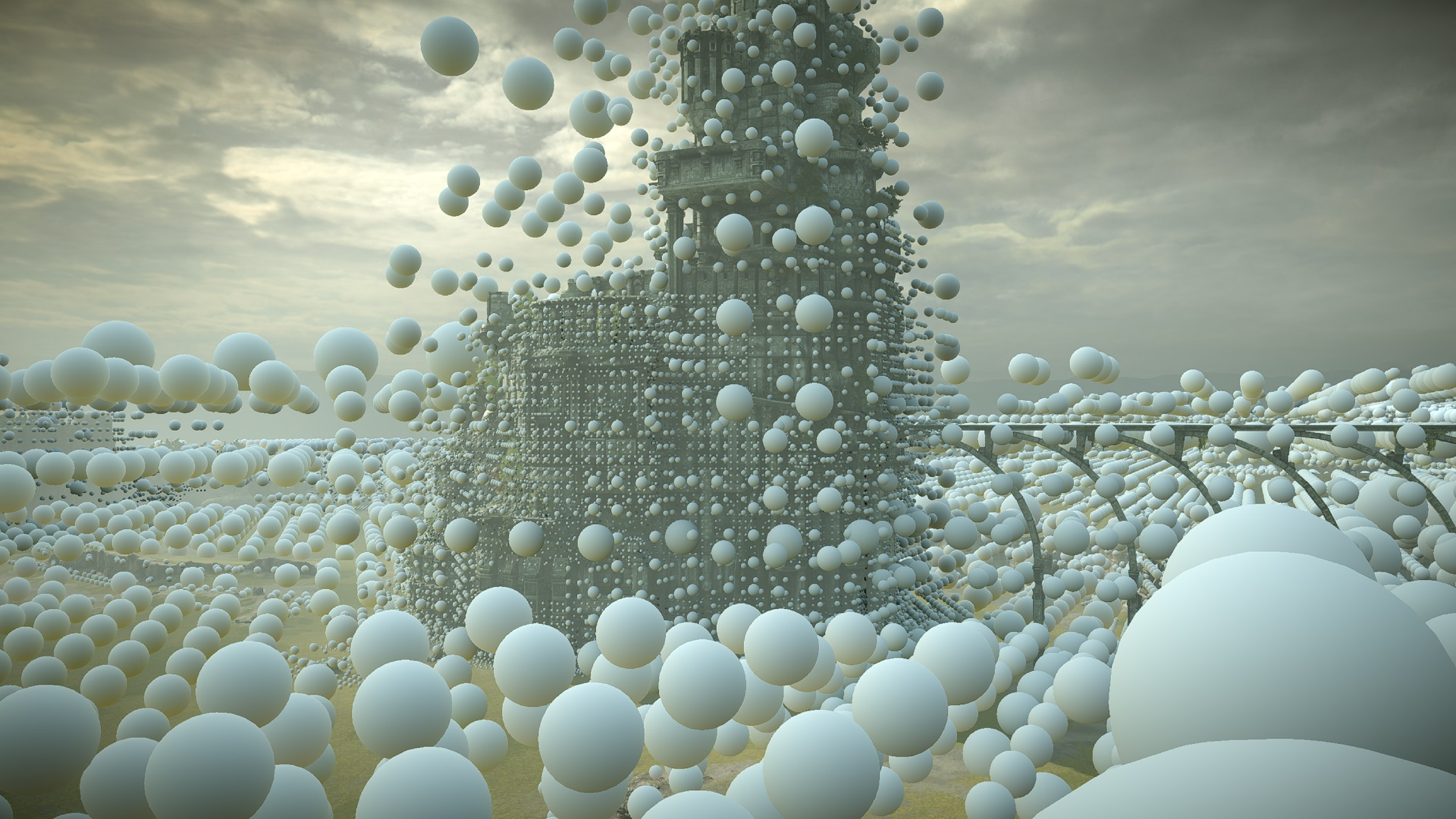
- Offline sliver identification & fixup
- Push attribute interpolation to pixel shader
  - Only interpolate position in VS
  - PS already does attribute interpolation
  - Adjust PS interpolation for subd tri
- Leverage mesh shaders for compression, fine culling
- Optimize displacement stage occupancy

Part 2: Lighting

Global Illumination

Shadow of the Colossus

- Starting point
  - Sparse spherical harmonic radiance probe clusters
  - Offline transferred to static vertex data
  - Runtime transferred to dynamic meshes



Shadow of the Colossus

- Limitations
  - Slow to iterate
    - Probes slow to re-light
    - Probe or static mesh changes require re-transfer
  - Slow to retrieve probes
    - Objects use single probe for entire mesh
    - Particle systems use single probe for all particles
    - Fog unable to use at all

Demon's Souls

- Complex interiors

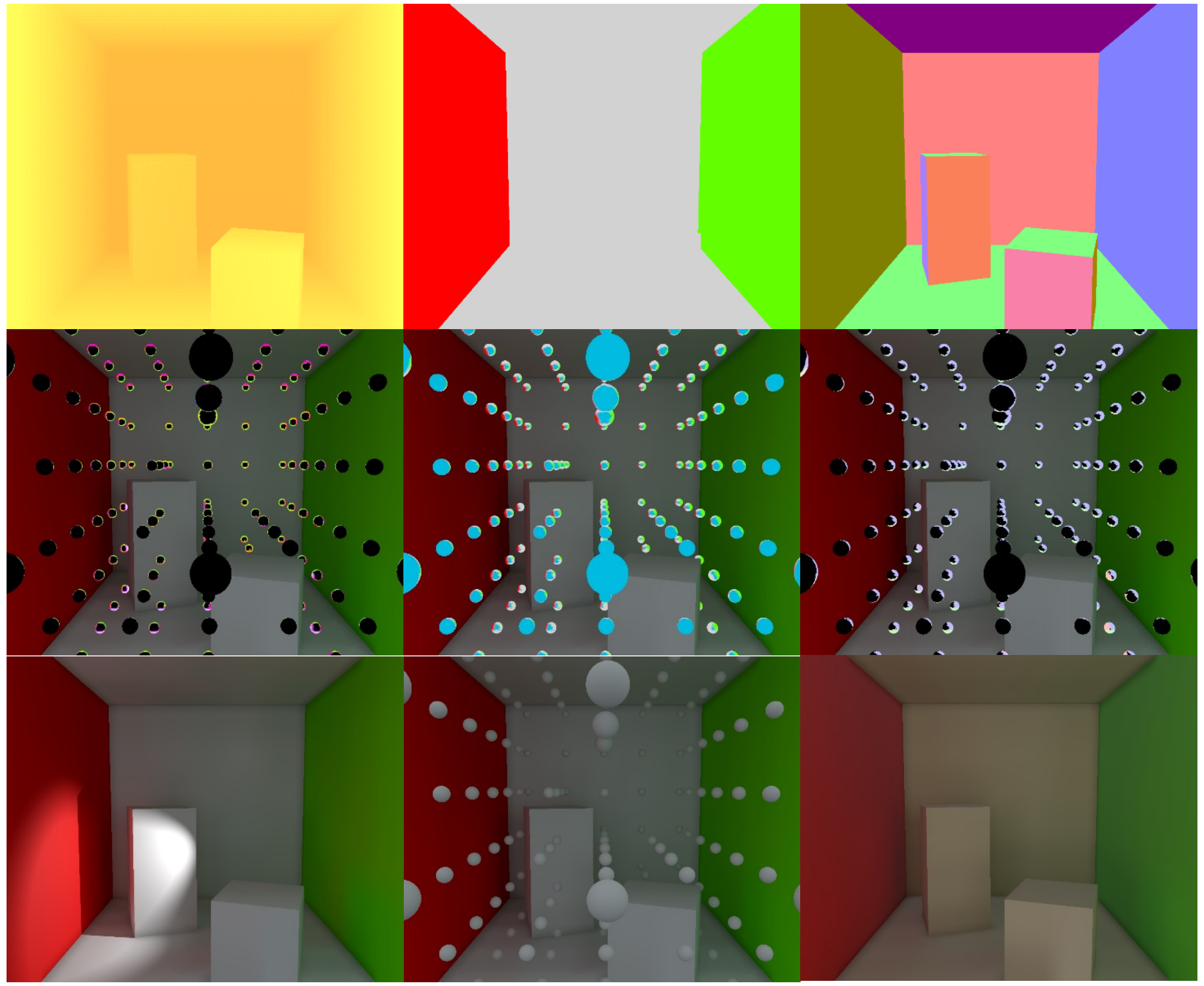



Demon’s Souls

- *Lots* of light rigging work
  - Need faster iteration!

Solution

- Inspired by [McGuire2017]
  - "Real-Time Global Illumination using Precomputed Light Field Probes"
- Basic idea
  - Add a Gbuffer to each probe
  - Render to capture indirect
  - Sample probes using weighted probe-to-sample relevance



Solution

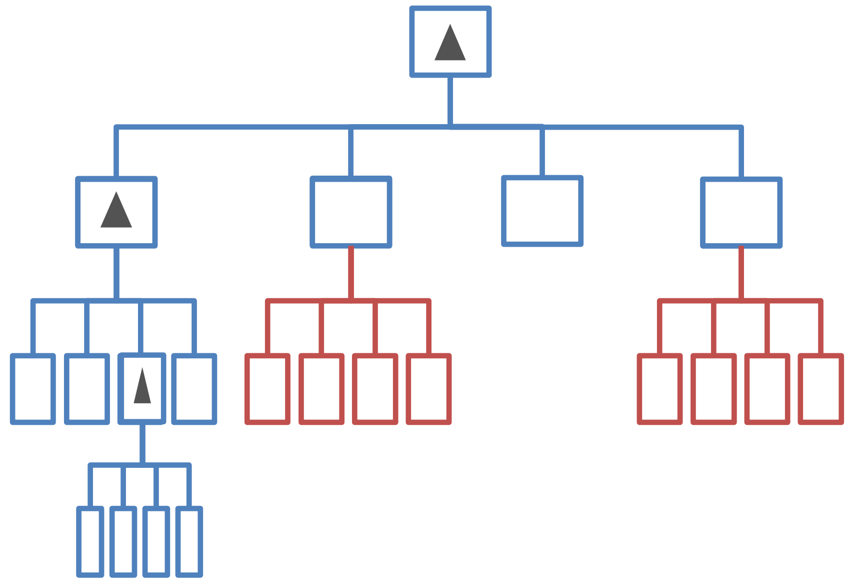
- Offline
  - Determine probe placement
  - Capture probe GBuffer
  - Capture probe lighting (cache for runtime)
- Runtime
  - Prioritize & recapture probe lighting
  - Blend updated probes
  - Sample probes

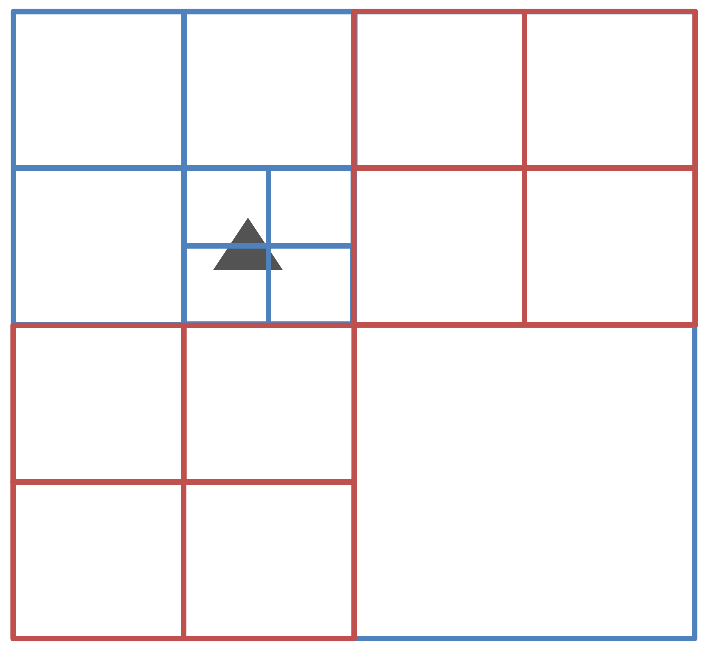
Probe Generation

- Sparse Octree
  - Cells with probes at corners
  - Bounds & spacing limit set by artists (1m+)
  - Placement $64m^3$ quantized for streaming/merging

Probe Placement

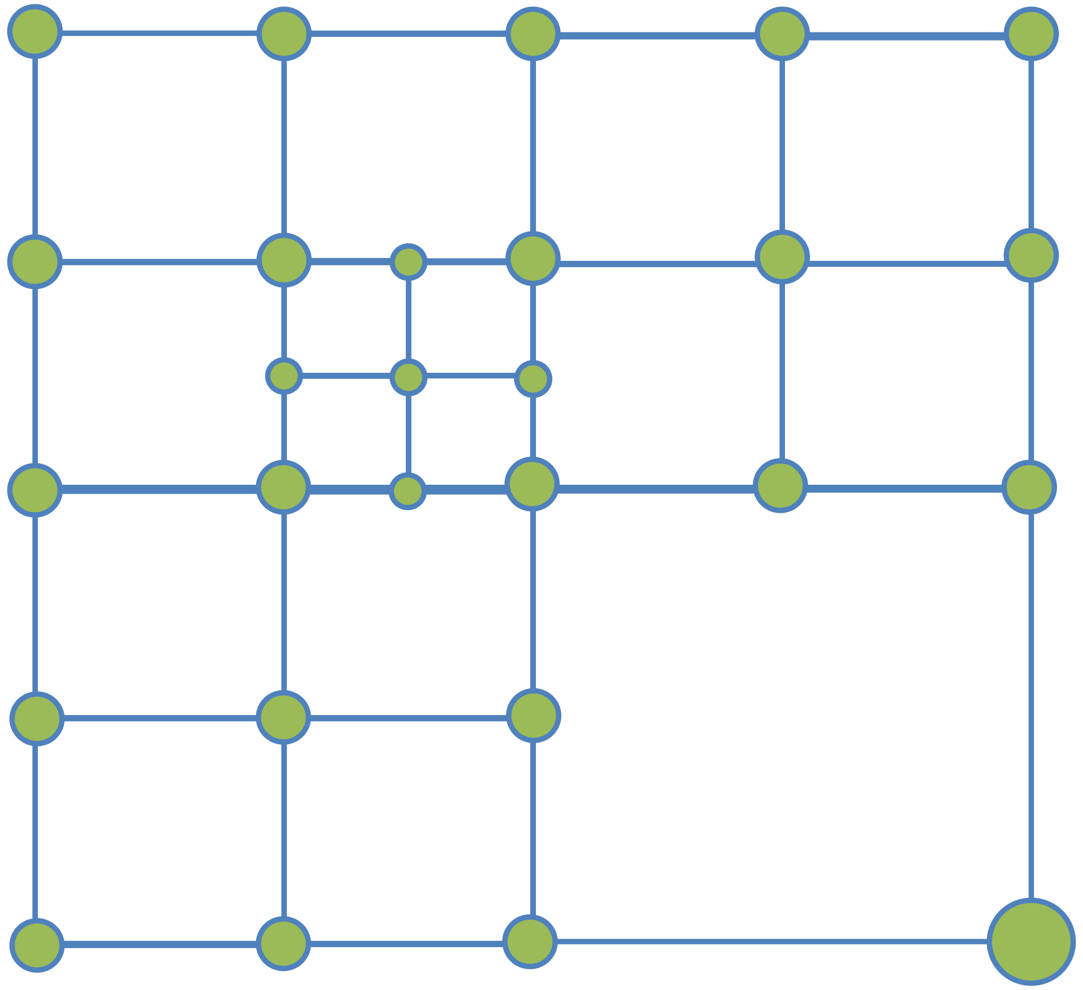
- Mark & Sweep
  - Traverse to tree leaves starting from root
  - Mark
    - Cells intersected by static geometry
    - Cells more than 1 LOD from neighbors
    - “Filler” cells until minimum resolution (optional)
  - Sweep
    - Fully split marked cells
  - Loop or stop if max depth/resolution





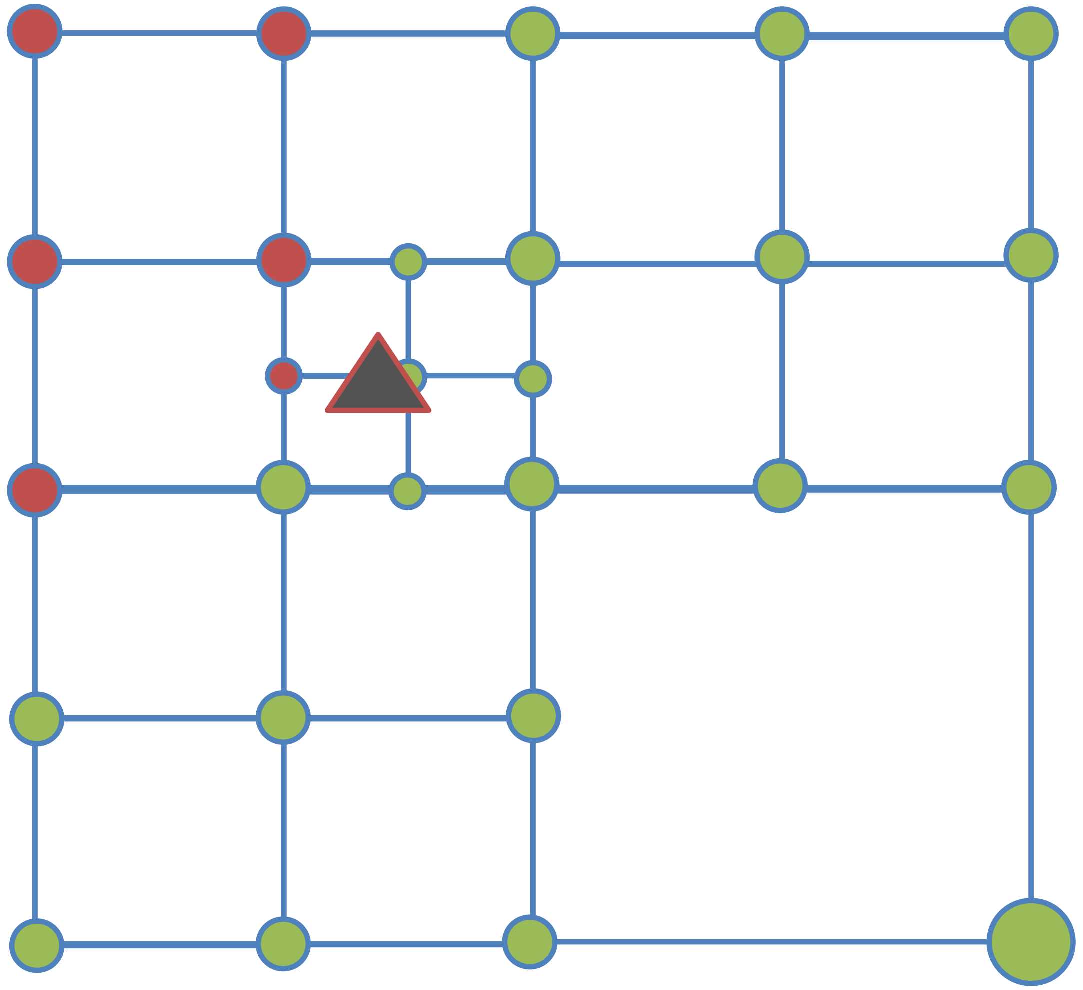
Probe Placement

- Generate probes from leaf cells
  - One probe per corner
  - Share with adjacent cells



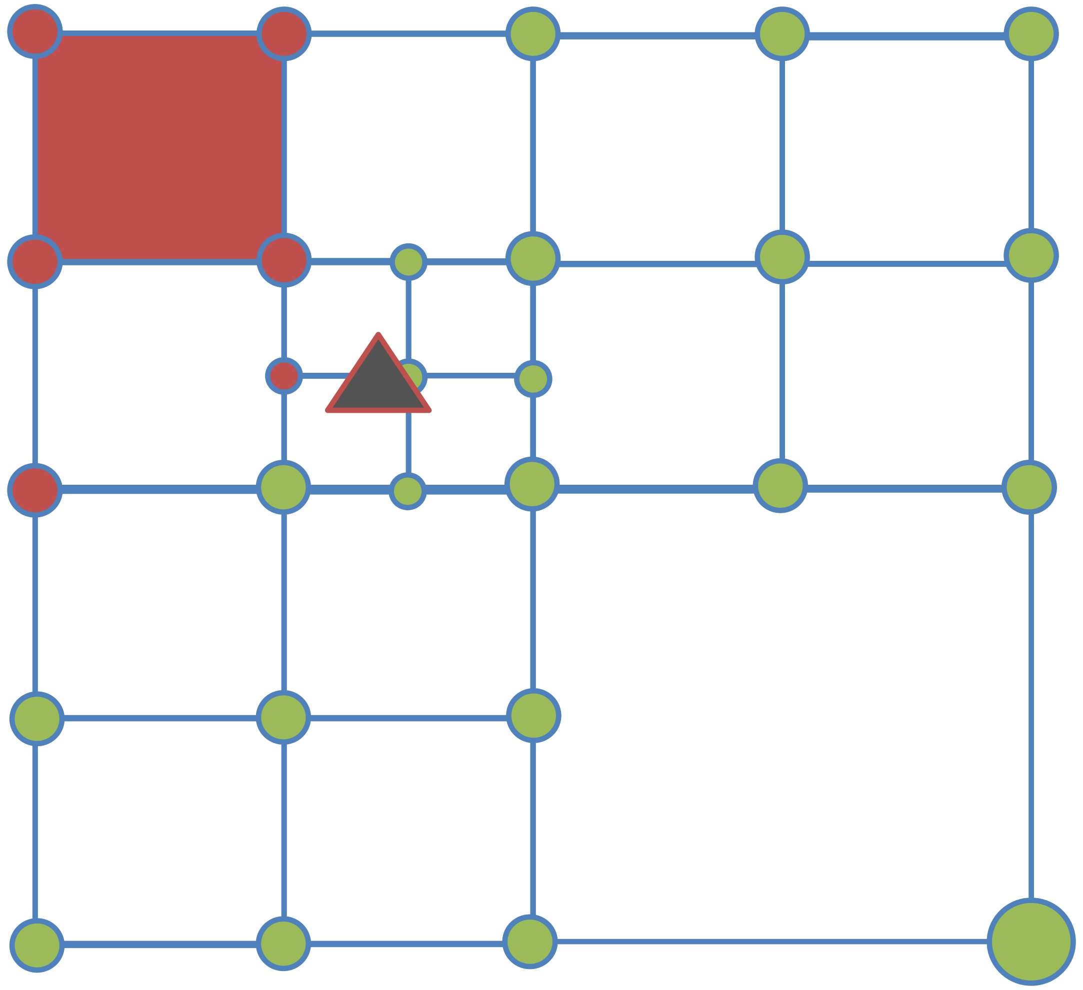
Probe Placement

- Prune probes
  - Probes that see backfacing pixels
  - Hand placed kill volumes

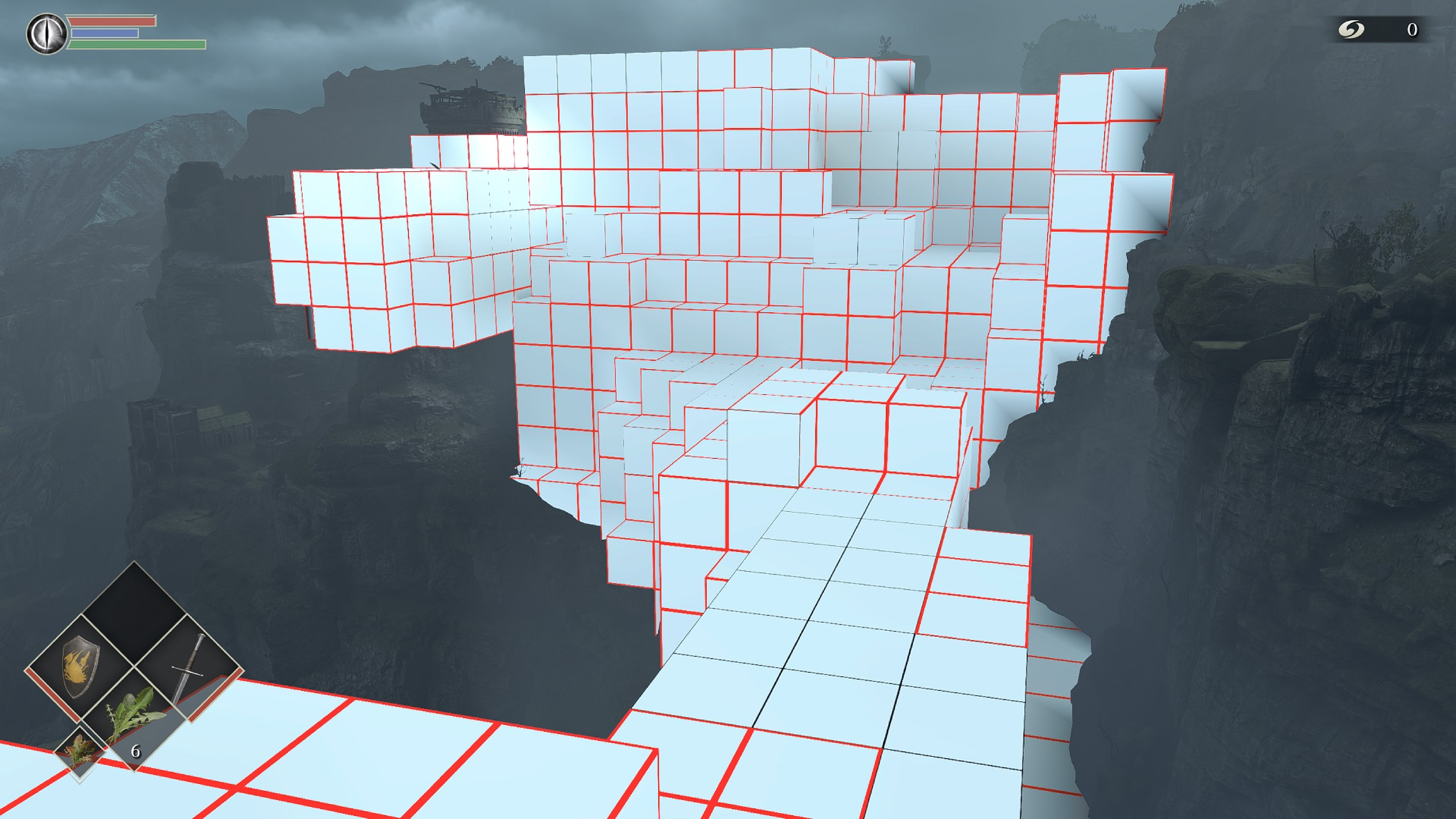


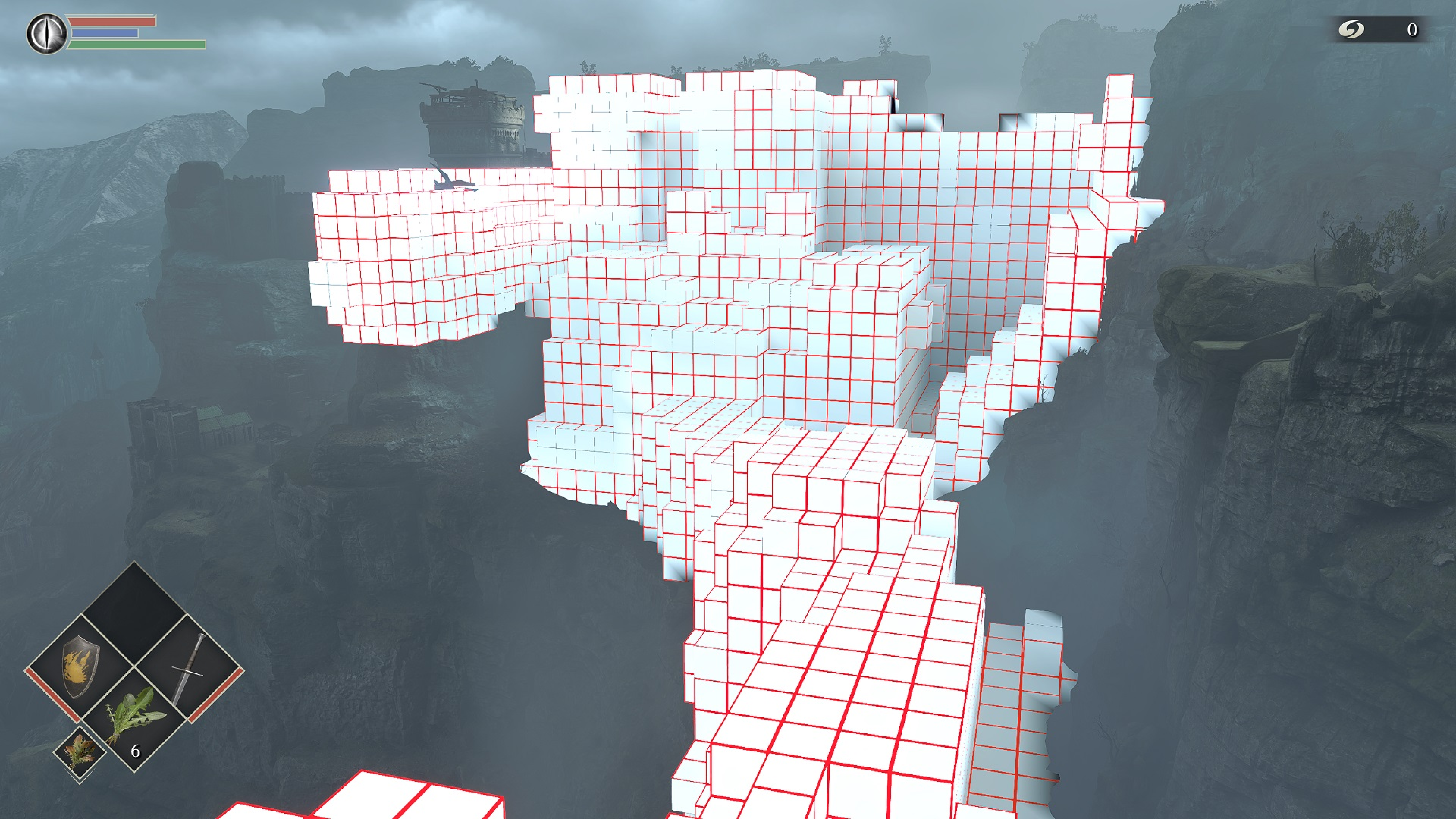
Probe Placement

- Prune cells
  - Cells with no valid probes




0
6


0
6


0
6

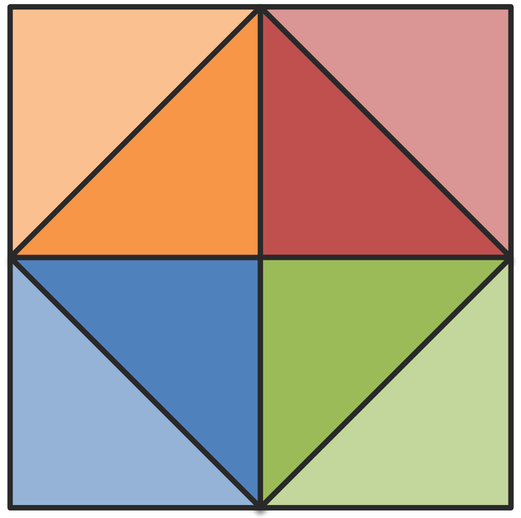

0
6


0
6


0
6

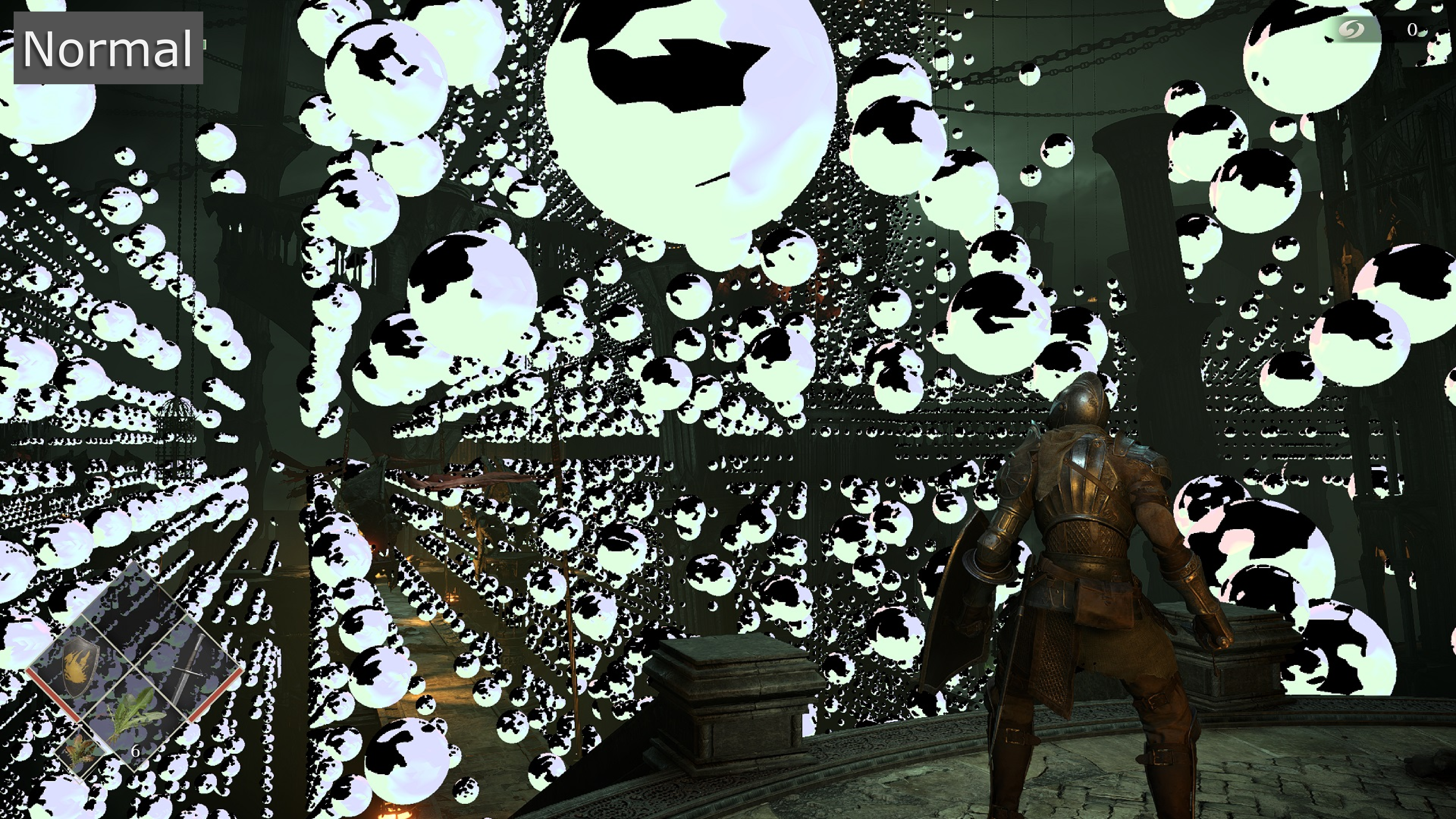
Gbuffer Capture

- Render probes to 256x256 cube faces
- Downsample to octahedral unwrapped buffers
  - 16x16 gbuffer (8B per pixel)
    - Albedo, normal
    - Sky visibility, Sun visibility
    - Radial depth (*including* water; used for lighting)
  - 24x24 depth test buffer (2B per pixel)
    - Radial depth (*excluding* water; used for probe weighting)
    - Includes a 1 texel wrapped border for 2x2 PCF

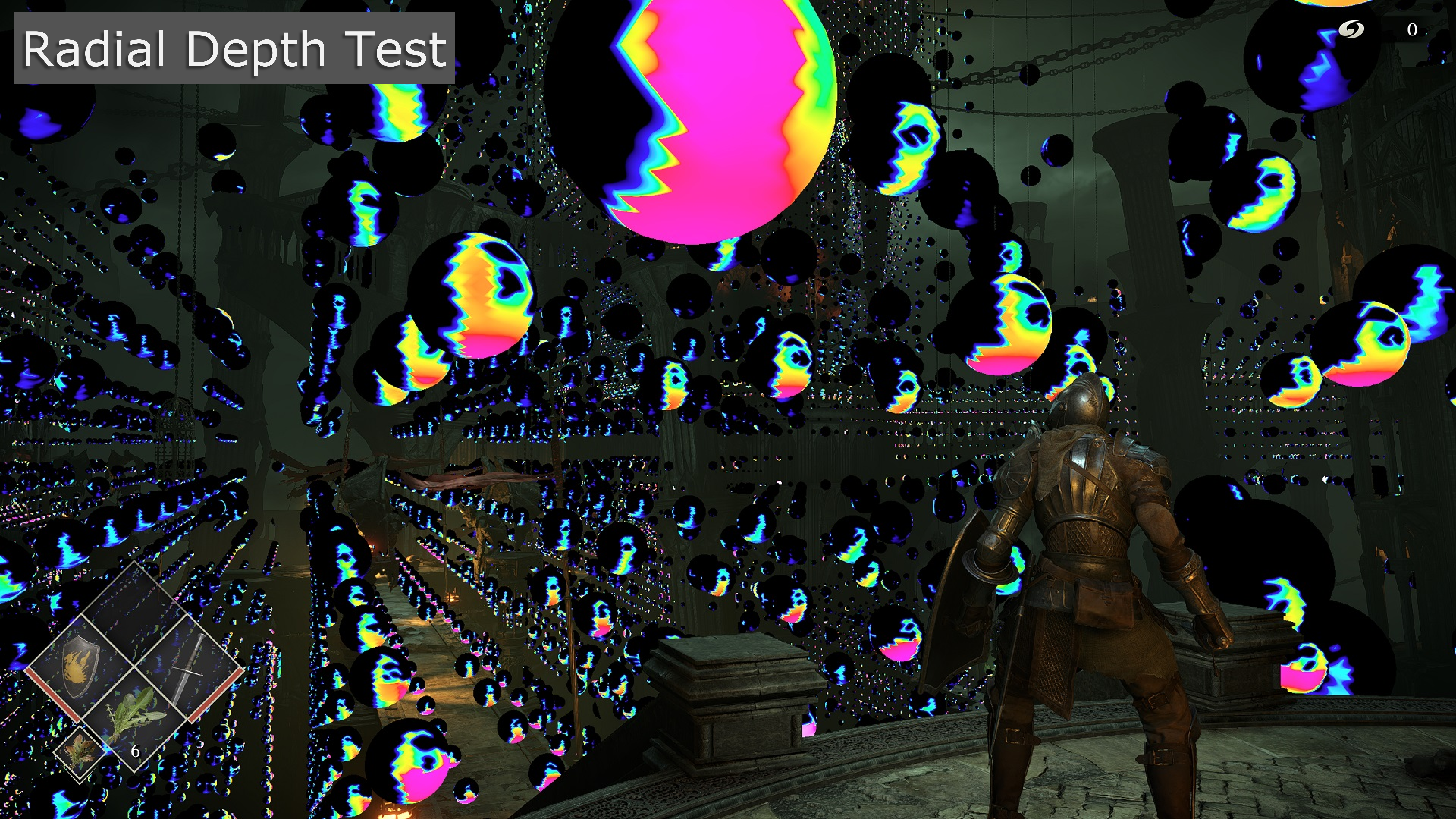



0
6


Albedo
+ sky visibility
0
6


Normal
0
6

Radial Depth Test

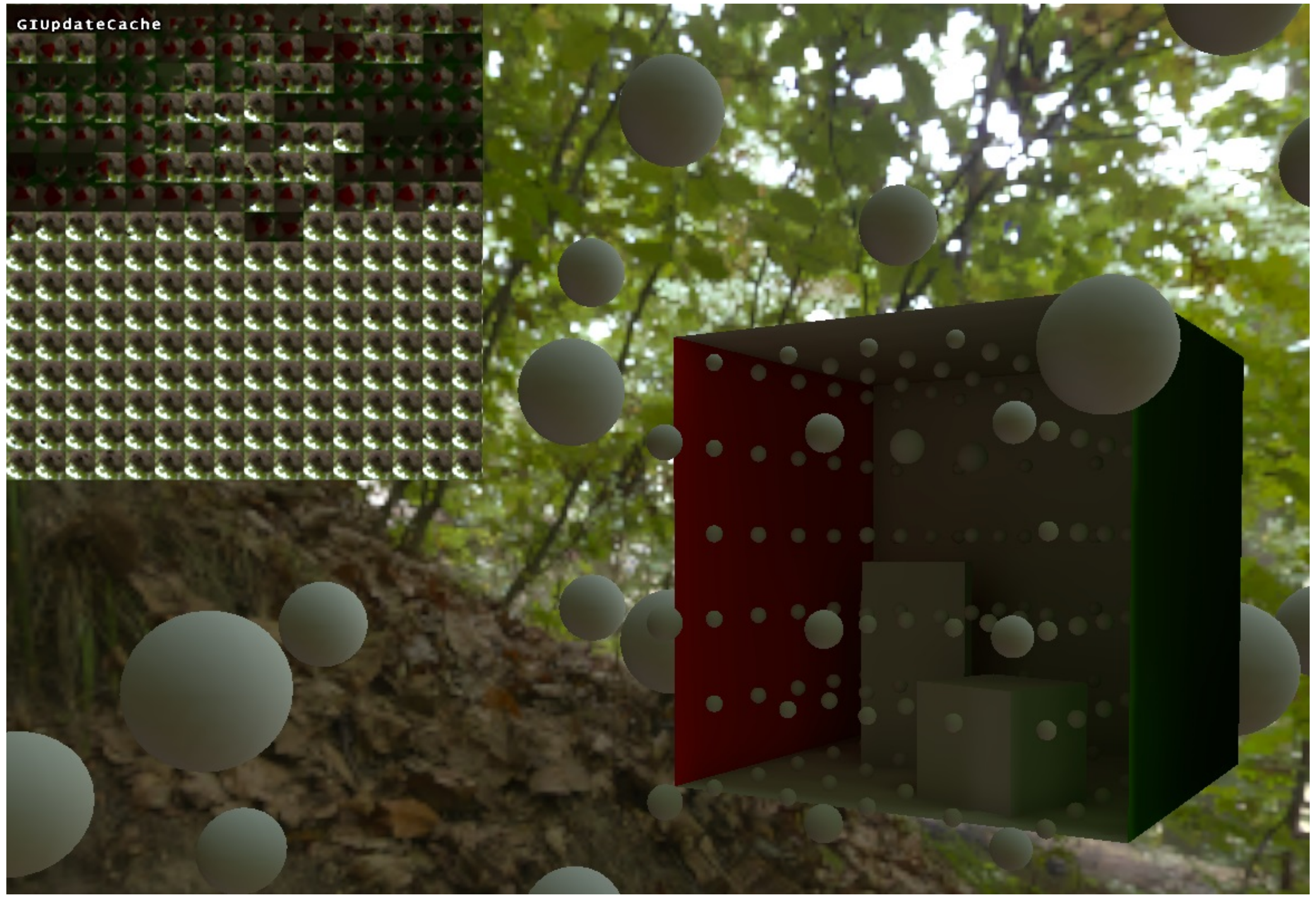


Probe Relight Prioritization

- Probes maintain an “age”
- Area-of-interest objects increase age over time
    - Cameras
    - Regions
    - Background
- Sort to find N (256) oldest probes

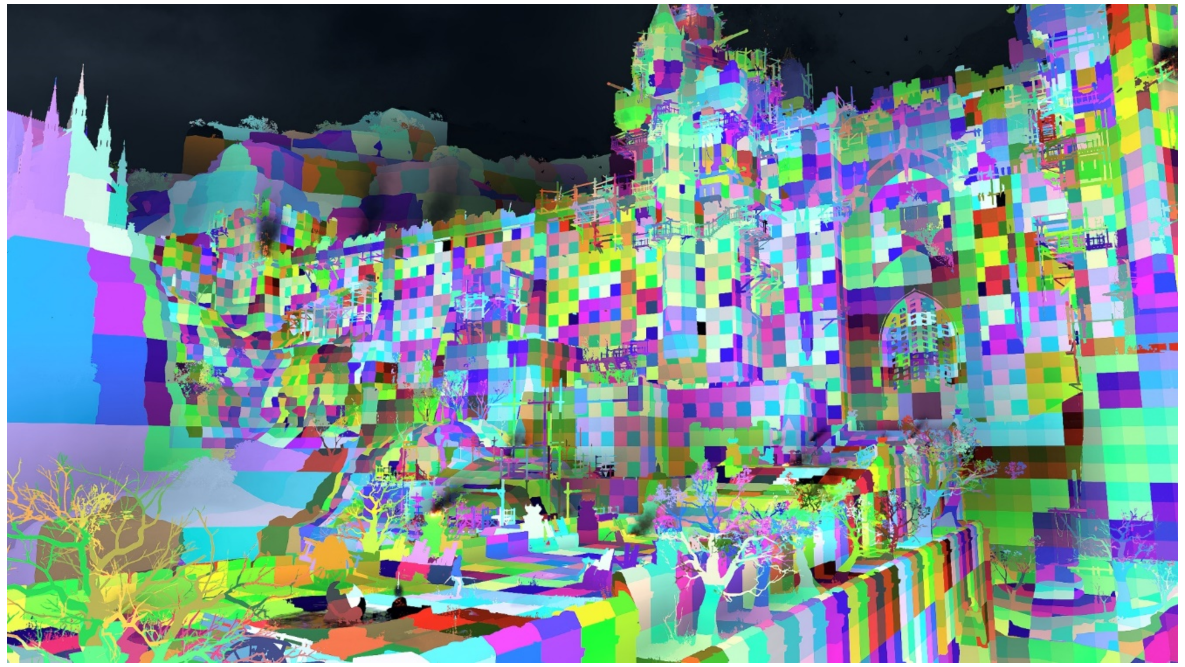
Probe Relighting

- Cull lights using AABB of probe gbuffer depths
- Render lighting to temp 16x16 octahedral unwrapped buffer
  - Indirect lighting
    - Sunlight x gbuffer sun fraction
    - Point, spot, box lights
    - GI probes (for secondary bounces)
    - Scaled by albedo * (1-skyFraction)
  - Direct lighting
    - HDR sky x skyFraction
    - *Not* scaled by albedo
- Transfer to spherical harmonics
  - Add in GI-only area lights



Acceleration

- Cell query LUTs
  - 2 level 3D texture lookup to find cell from position
  - Updated with level streaming
  - Cell chunk LUT
    - $64m^3$ region to sub-region of Cell LUT
    - Specifies resolution of sub-region in Cell LUT
  - Cell LUT
    - Chunk sub-region to cell index

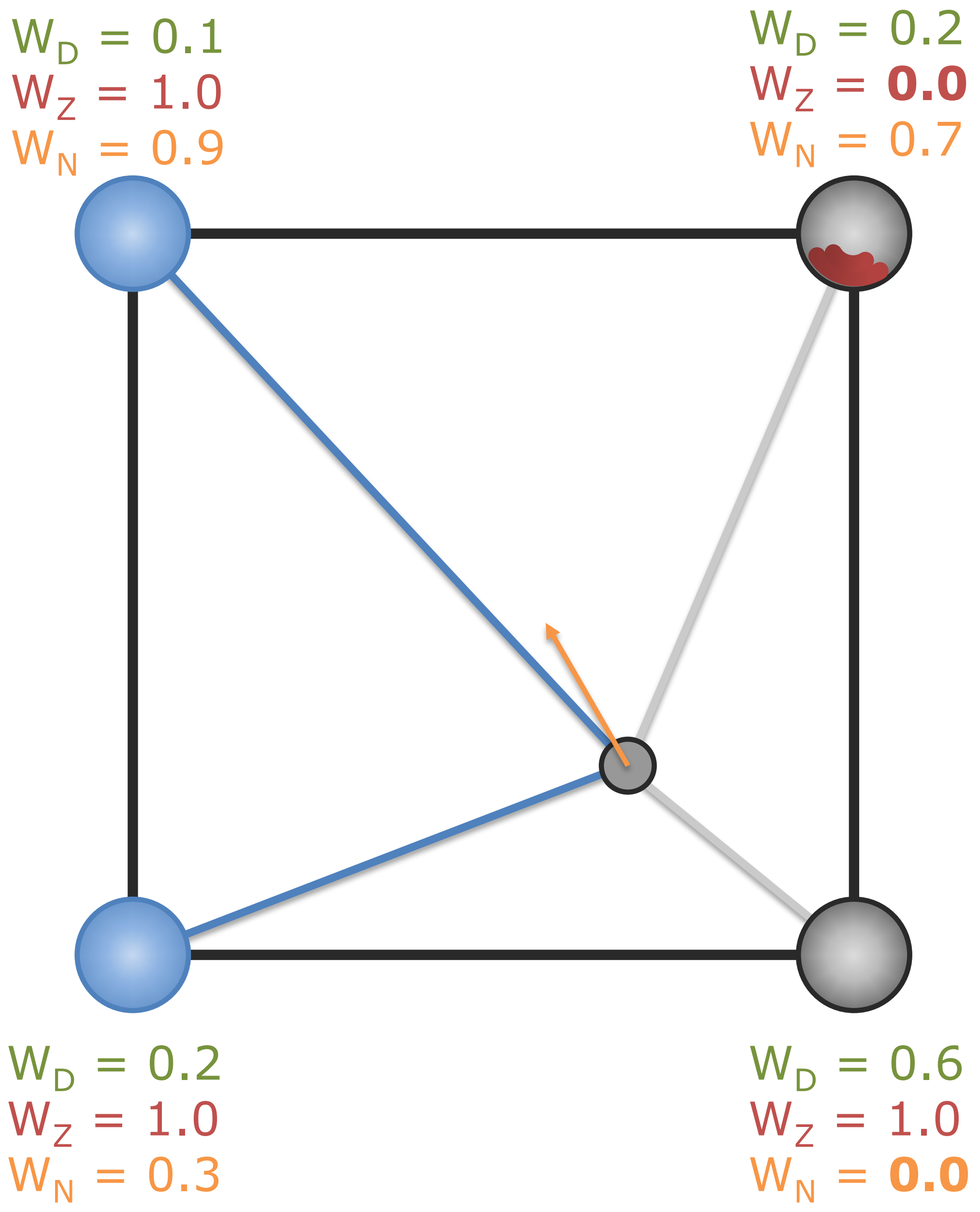


Acceleration

- Probe LUT
  - Updated with lighting changes
  - Irradiance texture
    - Reduce cost of diffuse lookup vs 9 coeff SH
    - 16x16 octahedral unwrap
  - Dominant light direction & color; used for…
    - Fog lighting, wrap lighting, water SSS
    - Ambient capsule shadows
    - Screen-space ambient shadows

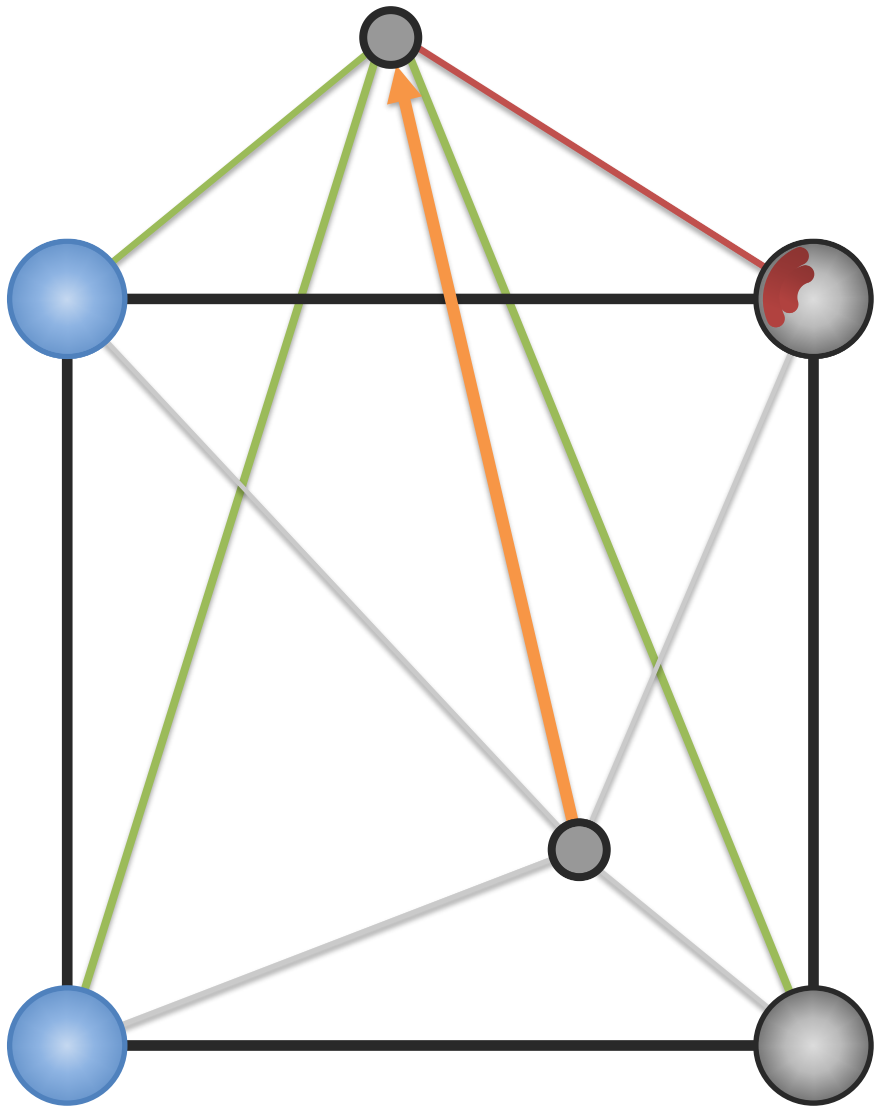
Probe Sampling

- Find cell for world position
- Loop over all (valid) cell probes
  - Generate contribution weight
    - Distance to probe (trilinear weight)
    - PCF occlusion test vs probe radial Zbuffer
    - Direction to probe vs sample Normal (optional)
  - Accum probe data x combined weight
- Normalize result



Bonus: Sound Occlusion

- Point to point visibility
  - CPU query, GPU result in 2 frames
  - Sample cell probes at start location
  - Test probe depth against endpoint
  - Blend probe results



Bonus: Sound Occlusion

- 360° earpoint radial “visibility” distance
  - Updated on GPU every frame
    - Raycast in each direction against every probe depth plane
    - Use hit closest to surface depth pos
  - Old results available to CPU

Stats

- Performance
  - Gbuffer ambient lighting (1440 / 4k): 0.8ms / 1.6ms
  - Fog ambient lighting (1440 / 4k): 0.2ms / 0.3ms
  - Relighting & bookkeeping: 0.3ms

- Memory usage
  - System: 200 MiB
  - Cell/Probe data: 200-500 MiB
  - Cell/Probe LUTs: 300 MiB

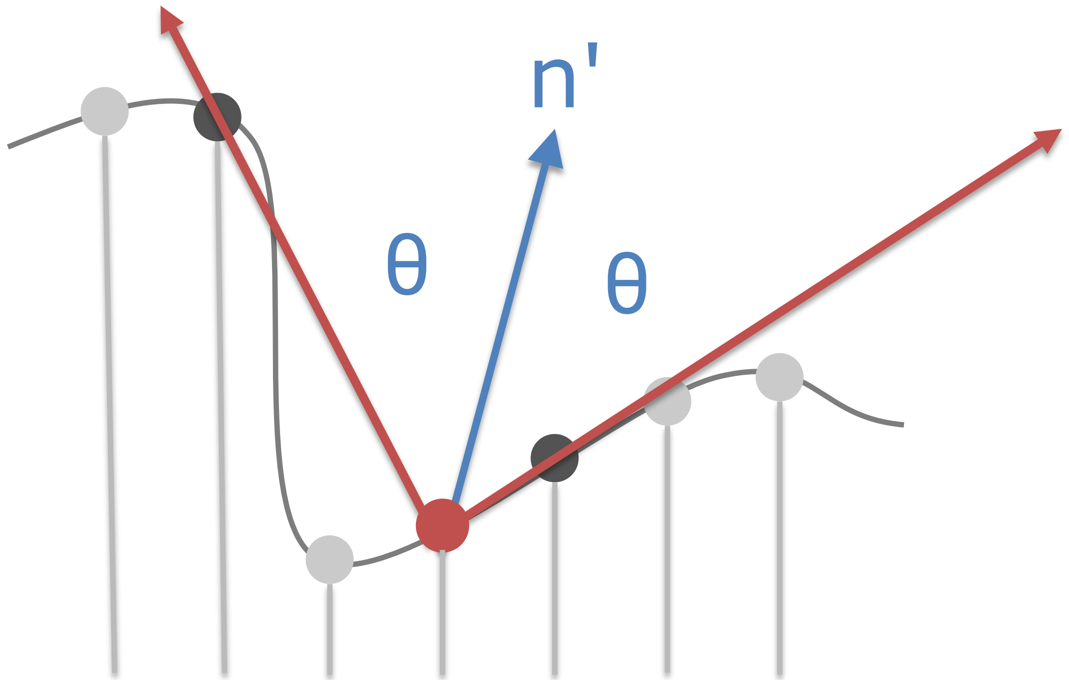
Limitations

- Significant memory usage
- Some light leaking
- Aliasing
- Slow relight (seconds, not milli-seconds)
- No specular
- No dynamic object bounce

Screen Space Shadowing

Screen Space Shadowing

- PSA for screen space bent visibility cones
  - Contact Shadows [Sousa 2011]
  - GTAO [Jimenez 2016]



Screen Space Shadowing

```
float ConeVisibility(float3 coneDir, float coneCosAngle, float3 testDir, float falloffRange = 0.5)
{
    float relativeCosAngle = dot(coneDir, testDir);
    float occlusion = saturate((coneAngle - relativeCosAngle) / falloffRange);

    return 1 - occlusion;
}
```

Screen Space Shadowing

- Ambient light shadows
  - Cone vs dominant direction / dominance
  - Cone zonal harmonics vs ambient SH
- Specular occlusion
  - Cone vs reflection direction
- Contact shadows
  - Cone vs light direction
  - Applied to *all* lights in Demon’s Souls

Omnidirectional AO

Cone Visibility

Omnidirectional AO

Cone Visibility

Thanks for Listening!

Special Thanks

- Martin Brownlow
- Dak Babcock
- Peter Dalton
- Marco Thrush
- Michael Kahn-Rose
- Justin Wagner
- Christopher Oat

We're Hiring!

https://www.bluepointgames.com

jobs@bluepointgames.com

Bluepoint

Questions?

bwoodard@bluepointgames.com

References

- [McGuire2017] Real-Time Global Illumination using Precomputed Light Field Probes, Morgan McGuire, 2017 ACM Symposium on Interactive 3D Graphics and Games https://research.nvidia.com/sites/default/files/pubs/2017-02_Real-Time-Global-Illumination/light-field-probes-final.pdf

- [Sousa2011] Secrets of CryENGINE 3 Graphics Technology, Tiago Sousa, SIGGRAPH 2011 https://www.slideshare.net/TiagoAlexSousa/secrets-of-cryengine-3-graphics-technology

- [Jimenez2016] Practical Realtime Strategies for Accurate Indirect Occlusion, Jorge Jimenez, SIGGRAPH 2016 https://www.activision.com/cdn/research/s2016_pbs_activision_occlusion.pptx