



**SIGGRAPH** THINK  
BEYOND  
2020 S2020.SIGGRAPH.ORG

---

## LARGE VOXEL LANDSCAPES ON MOBILE

Arseny Kapoulkine, Roblox

# What is Roblox?

- Online multiplayer game creation platform
- All content is user generated
- 100M+ MAU, 5M+ CCU
- Windows, macOS, iOS, Android, Xbox One
- Direct3D 9/11, OpenGL 2/3, OpenGL ES 2/3, Metal, Vulkan





### Explorer

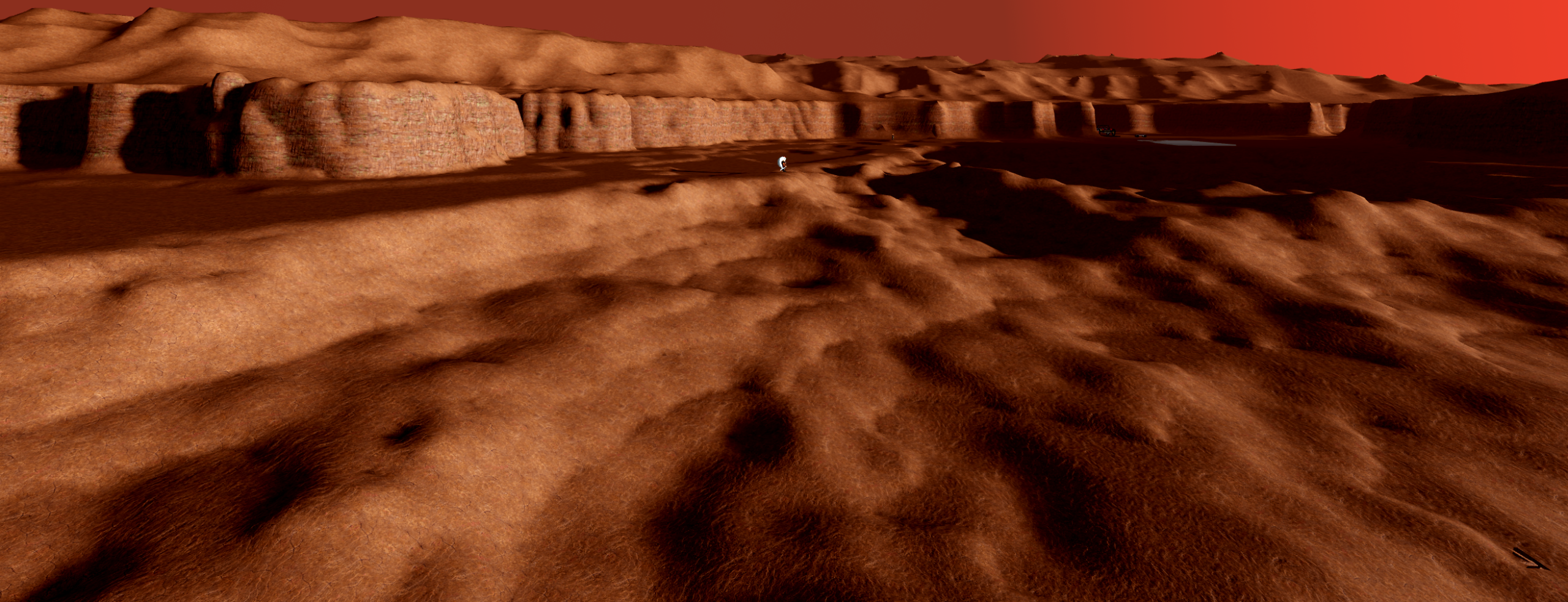
Filter workspace (Ctrl+Shift+X)

- Workspace
  - Camera
  - Terrain
  - SpawnLocation
  - AtollIsland
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Cannon
  - Chest
  - Chest
  - Chest
  - Cutter
  - DesertedIsland
  - FortIsland
  - Model
  - OpenChest
  - VolcanoIsland

### Properties - Terrain "Terrain"

Filter Properties (Ctrl+Shift+P)

- Appearance
  - Decoration
- Material
  - Asphalt [115, 123, 107]
  - Basalt [30, 30, 37]
  - Brick [138, 86, 62]
  - Cobblestone [132, 123, 90]
  - Concrete [127, 102, 63]
  - CrackedLava [232, 156, 74]
  - Glacier [101, 176, 234]
  - Grass [106, 127, 63]
  - Ground [102, 92, 59]
  - Ice [129, 194, 224]
  - LeafyGrass [115, 132, 74]
  - Limestone [206, 173, 148]
  - Mud [58, 46, 36]
  - Pavement [148, 148, 140]
  - Rock [102, 108, 111]
  - Salt [198, 189, 181]
  - Sand [143, 126, 95]
  - Sandstone [137, 90, 71]
  - Slate [63, 127, 107]



# Terrain system: goals

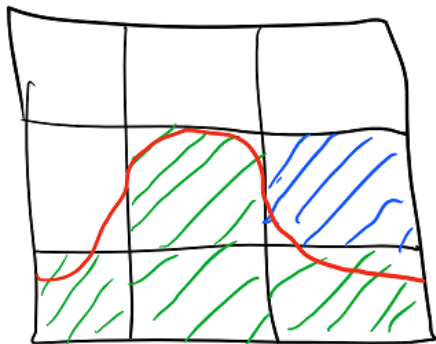
- No baking; any area can change at any point
- Fully 3D; caves, overhangs, bridges
- Scale up to reasonably large landscapes (10+ km<sup>2</sup>)
- Scale down to very small devices (iPad 2)
- Easy to use tools and API
- Rich materials (semantics, not just visuals)

# Voxel terrain

- Terrain is built out of voxels
- Voxels are sparse\* and multiresolution\*
- Each voxel has a material and occupancy
- All other controls are per material

# Voxel representation

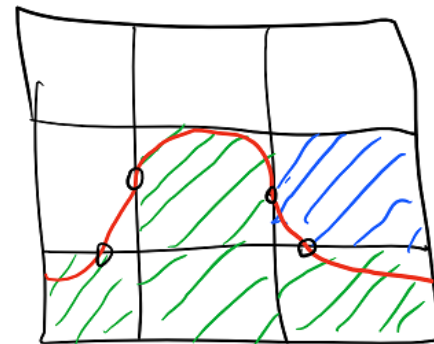
- Early experiments with different types of data
- Occupancy over other representations for simplicity



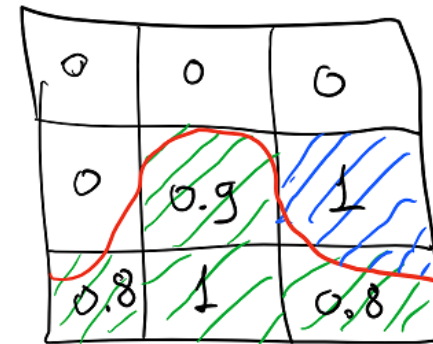
Material



Signed distance



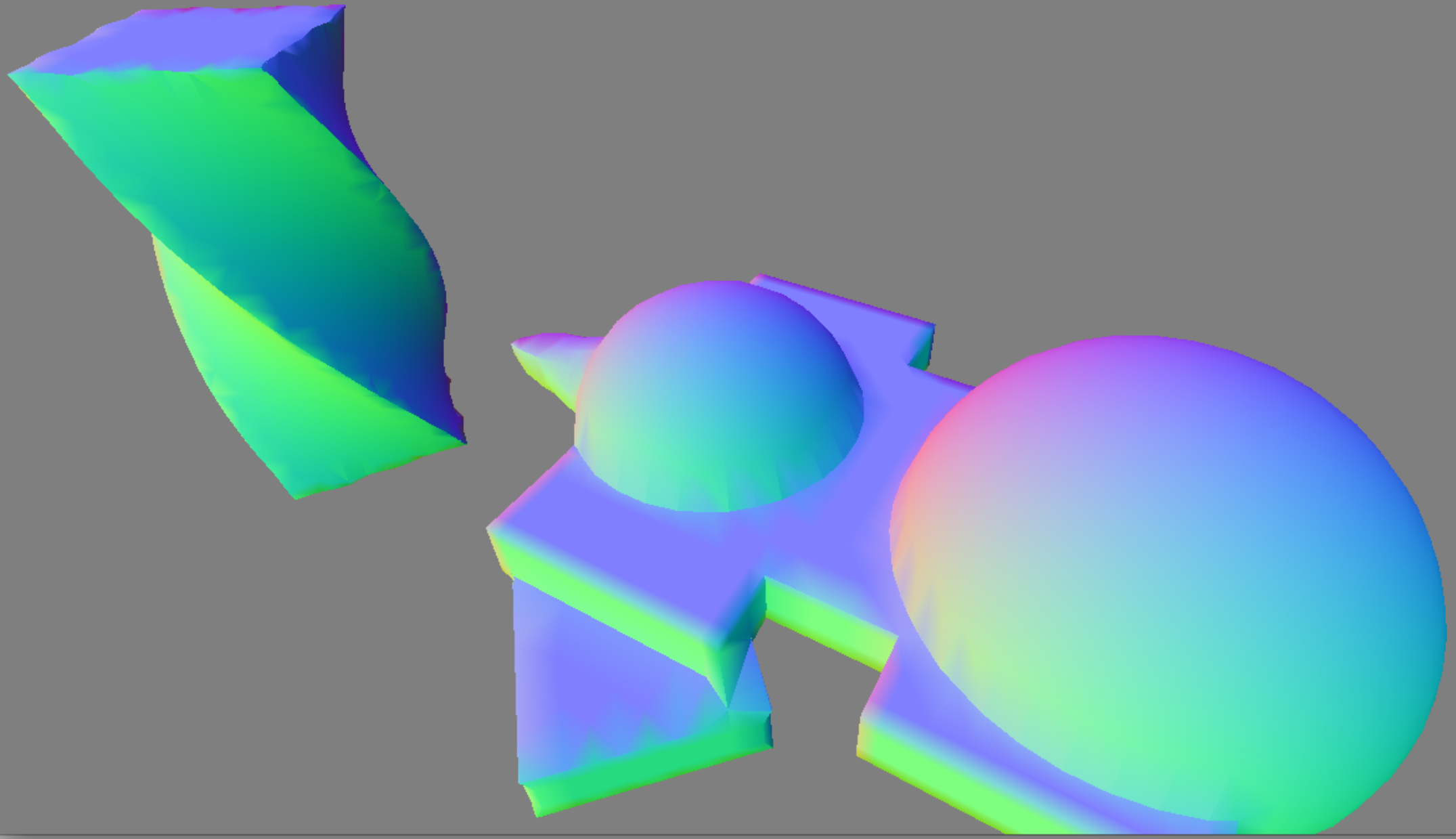
Hermite data

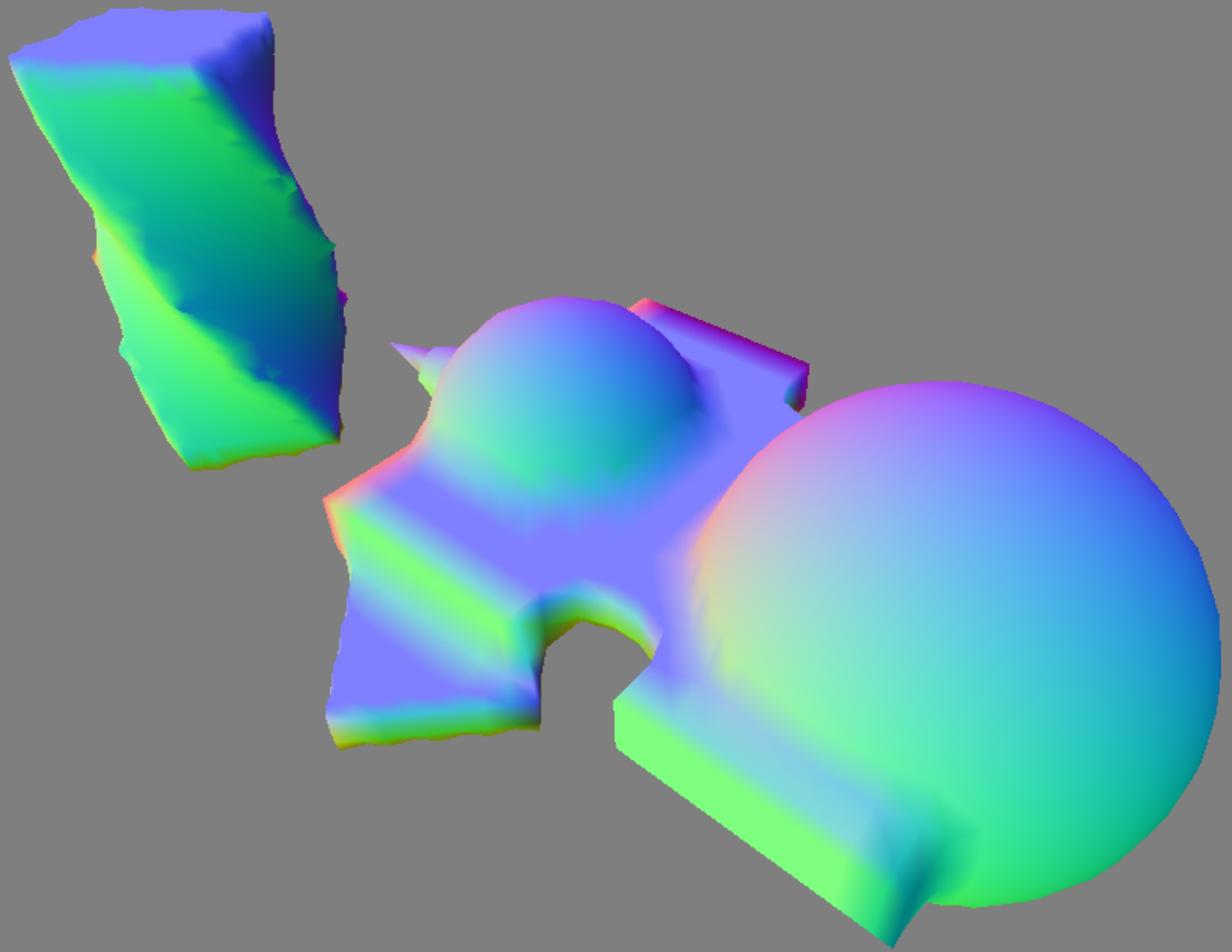


occupancy

▣ grass  
▣ water  
□ air  
○ surface  
○ edge point







# Voxel storage: grid

- Sparse set of fixed-size chunks
- Each chunk stores a mip pyramid ( $1^3$  ..  $32^3$ )
- Top levels can be skipped!
- Streaming mips in/out based on memory pressure

# Voxel storage: mips

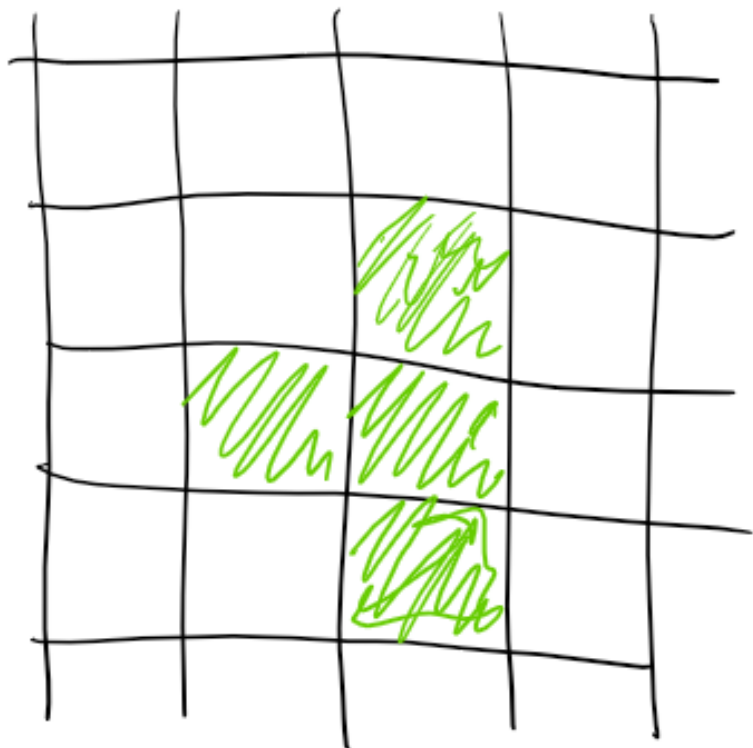
- Empty/full mip: 1 byte (material)
- Compressed rows: 1 byte (material) per row
- Uncompressed rows: 2 bytes (material + occupancy) per voxel
- Repack dynamically after voxel writes

# Mesher: Marching Cubes?

- Non-uniform topology
- Needs tie breaking rules
- Non-intuitive and restrictive vertex placement

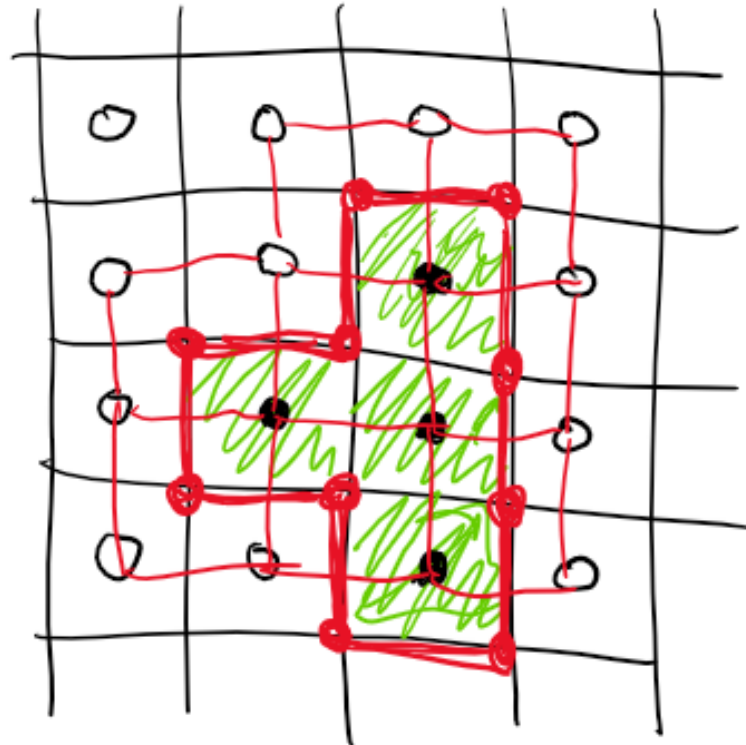
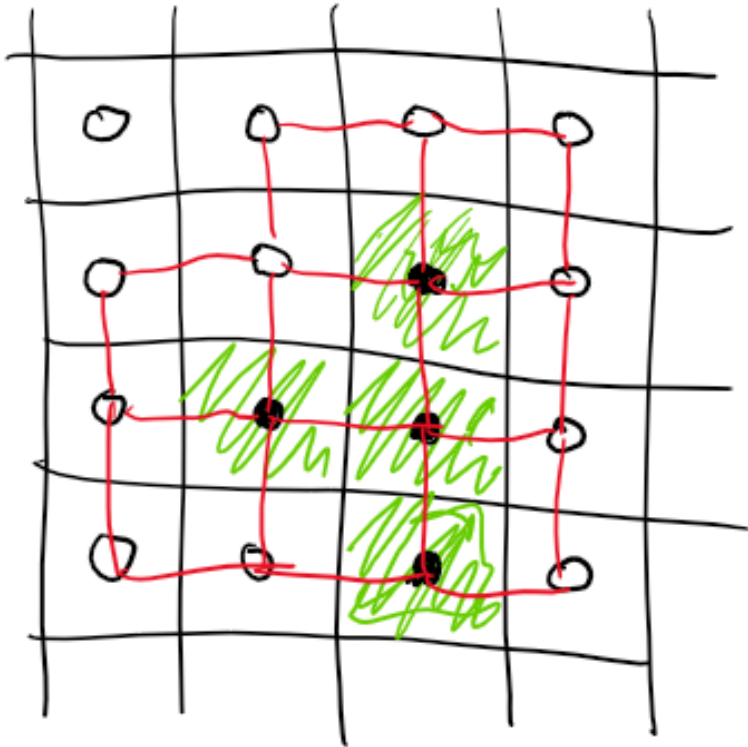
# Mesher: dual method

- Inspired by Dual Contouring and Naive Surface Nets
- Runs on CPU (carefully optimized)



# Mesher: dual method

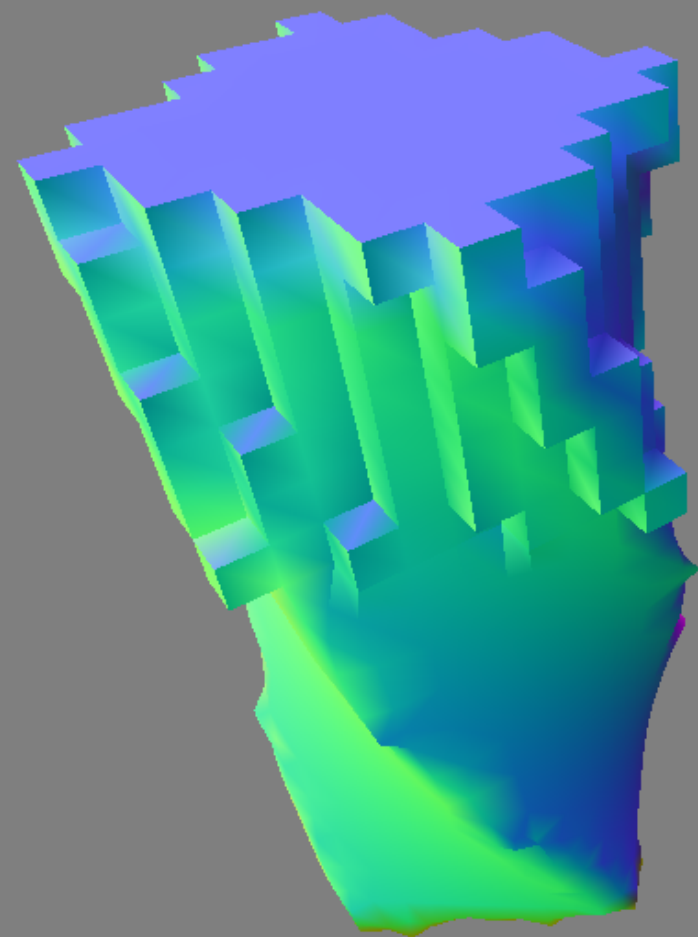
- One vertex per cell
- Neighboring cells are connected with quads

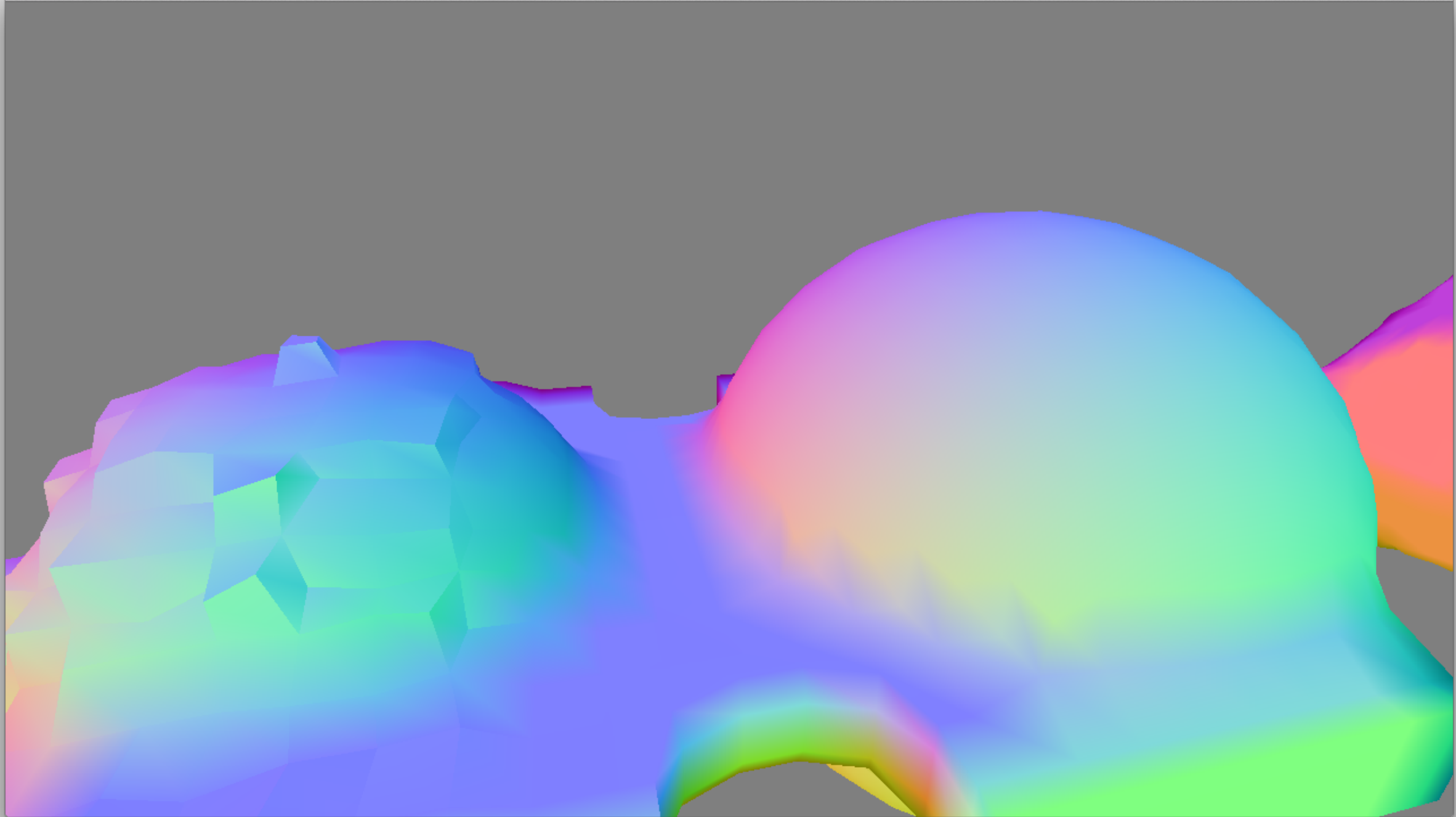


# Mesher: vertex data

- For shading we decided to use 1 material per vertex
  - Dominant material based on occupancy from grid points
- Baseline position is computed as an average of edge points
  - See Naive Surface Nets
- Normal is computed as an average of triangle normals
- ... but we don't stop there

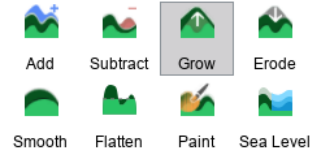






# Mesher: vertex deformation

- Materials define a procedural deformation:
  - Shift: pseudo-random offset
  - Cubify: lerp to box center
  - Quantize: round to a multiple of  $1/K$
  - Barrel: cubify along Y
  - ... and more (special math for water, etc.)
- Materials also define soft/hard edges for shading



### Brush Settings



Base Size  5

Strength  1

Pivot Position  Bot  Cen  Top

Plane Lock

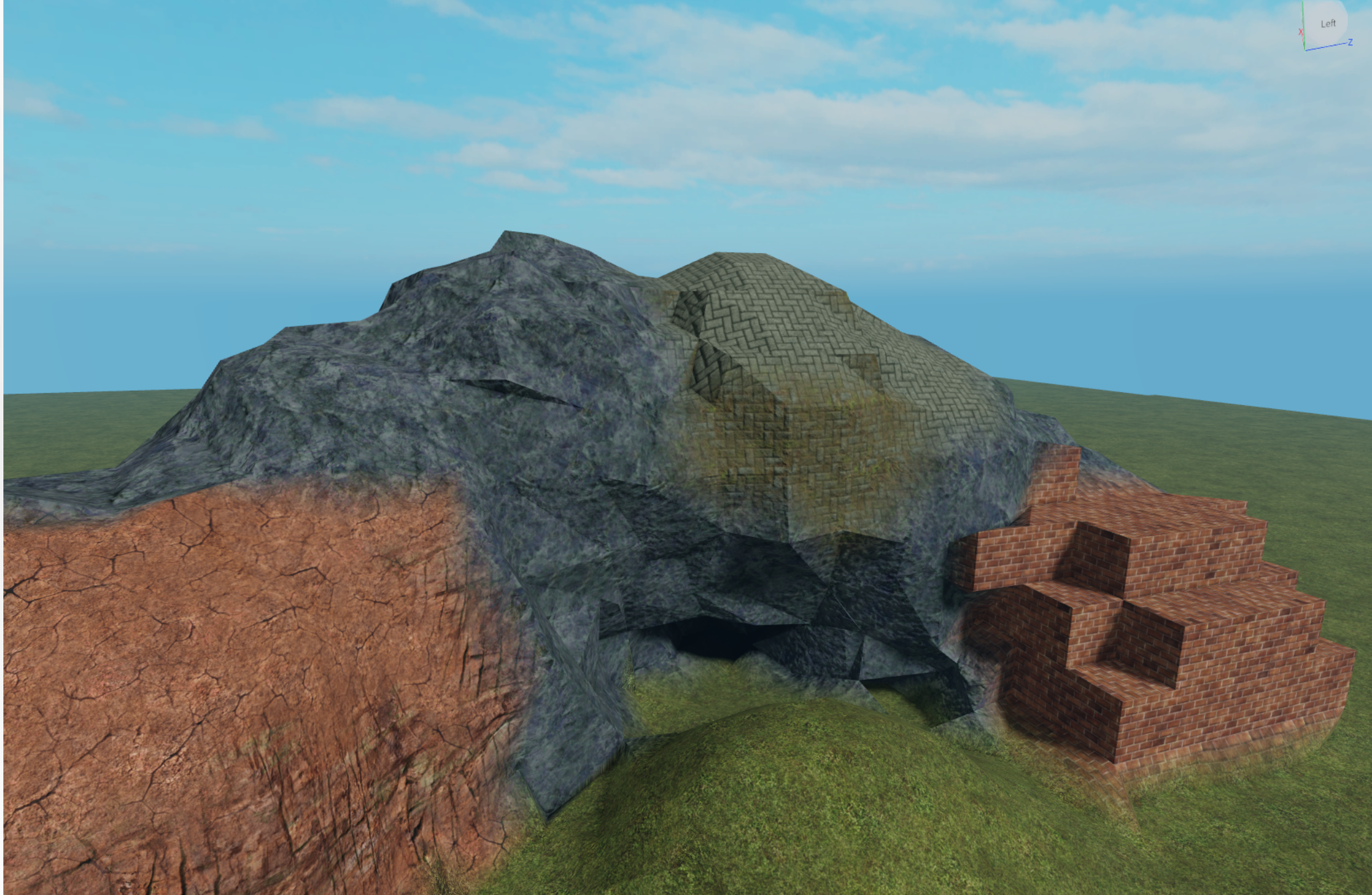
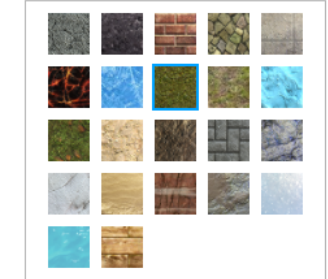
Snap to Grid

Ignore Water

### Material Settings

Auto Material

Choose a material to apply



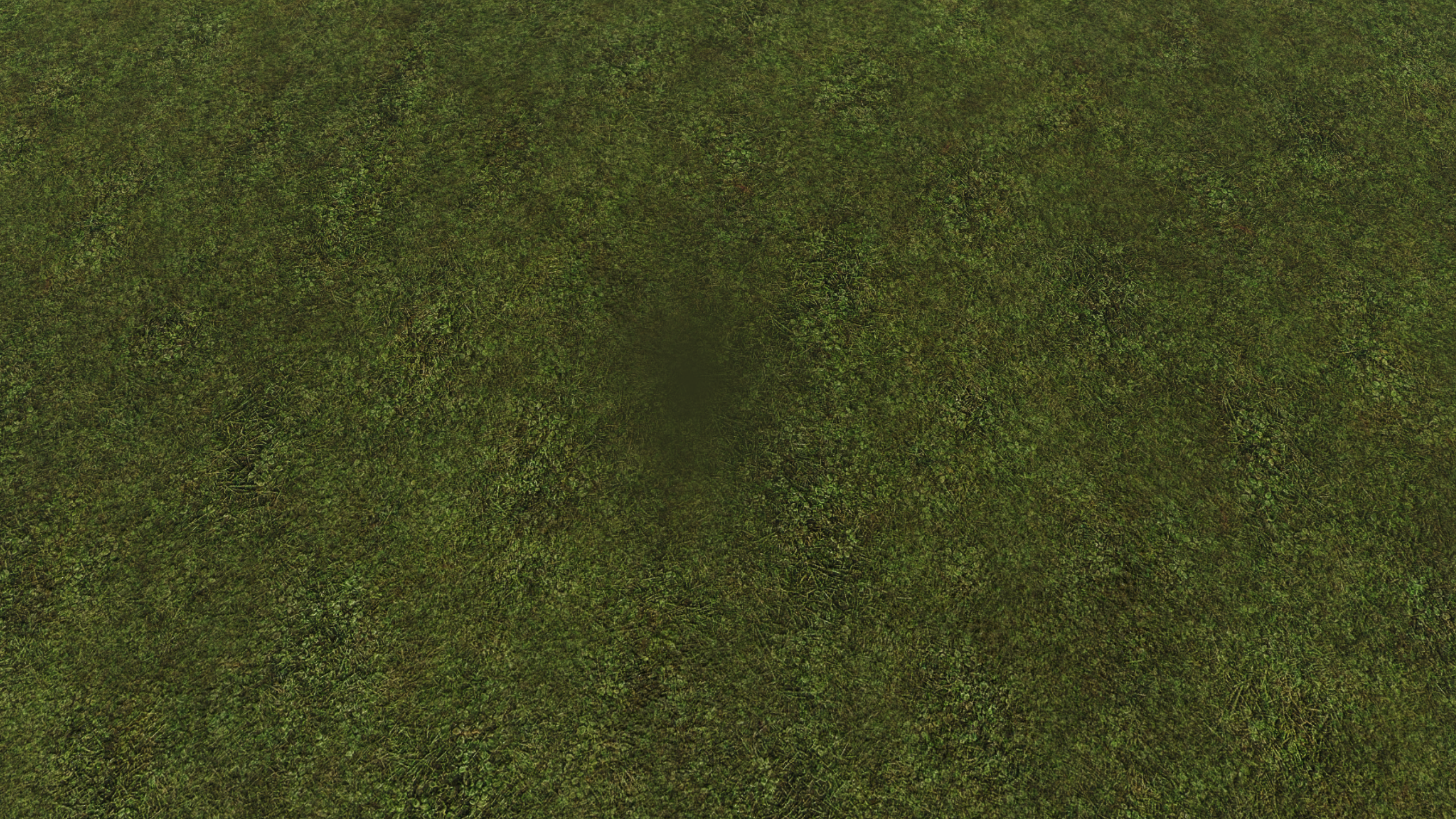
# Texture mapping

- Given a unique material per vertex, how do we shade a pixel?
- Sample textures from 3 materials, blend based on barycentrics
- Unclear how to project - perhaps triplanar?
- 3 material samples x 3 triplanar samples x 3 textures = 27 fetches
- ... no.

# Texture mapping: quilting

- Idea: SIGGRAPH 2011, Real-time Image Quilting by Hugh Malan
- For each vertex, we're going to sample material just once
  - Total samples: 3 on low quality (albedo), 9 on high quality (PBR)
- We're going to pick a projection plane to minimize distortion
  - Not quite triplanar, but can be close enough!
  - We pick one of 18 planes and encode plane id in vertex







# Texture mapping: detiling

- After projecting position on the UV plane, we apply random xform
  - Shift and rotate based on per-vertex seed
  - Material controls the transformation
- When all 3 materials are the same, this hides tiling artifacts





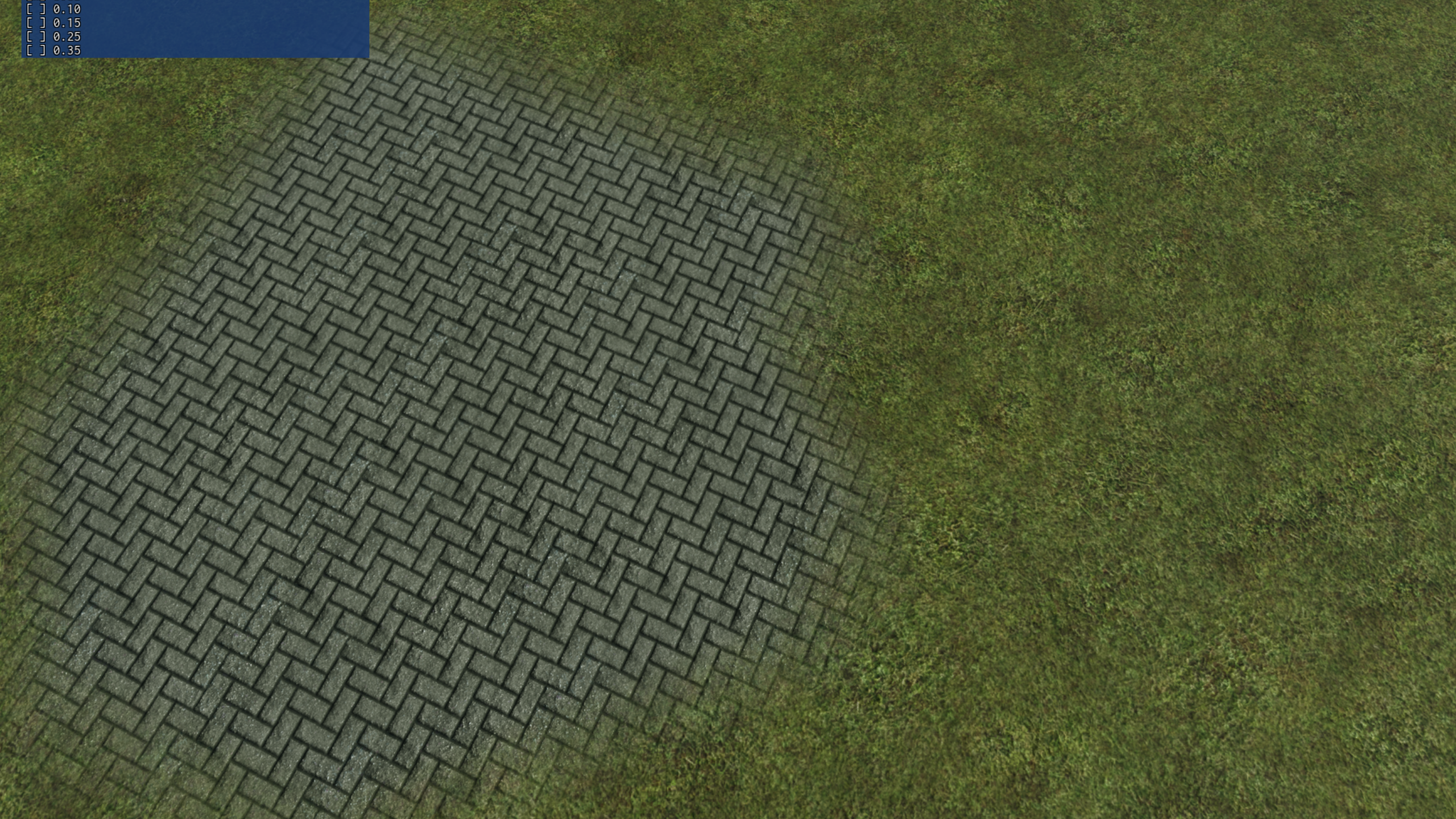
# Texture mapping: scaling down

- Effectively each of 3 samples can act as:
  - Material sample (for 3-material blend)
  - Triplanar sample (for blend between material layers)
  - Detiling sample (to break up tiling)
- On low-quality we can only use one with largest weight
  - This results in sharp seams between materials

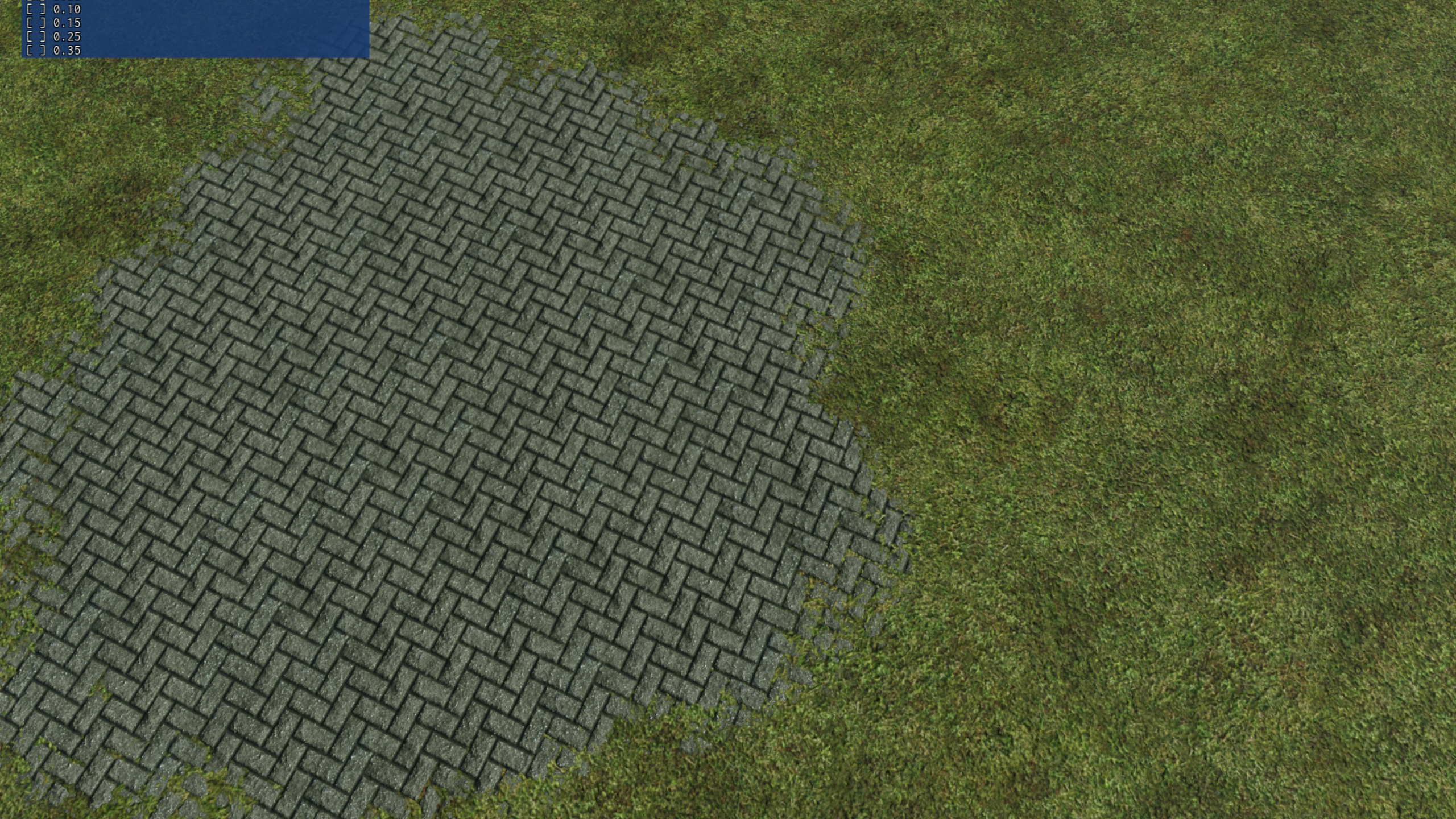
# Texture mapping: scaling up

- Linear blend reduces contrast and results in unnatural blends
- Height-based blending (fixes blends)
- Histogram-preserving blending (fixes loss of contrast)

[ ] 0.10  
[ ] 0.15  
[ ] 0.25  
[ ] 0.35



[ ] 0.10  
[ ] 0.15  
[ ] 0.25  
[ ] 0.35



# Vertex packing

- Unfortunately we need to store triangle material information
- Experimented with GS, promising results - future is mesh shaders?

```
// 20 bytes/vertex
struct Vertex
{
    int16_t position[3];
    int16_t id;           // vertex index (1-3)
    uint8_t normal[4];   // xyz = normal, w = random seed 0
    uint8_t material0[4]; // xyz = layer index (0-?), w = random seed 1
    uint8_t material1[4]; // xyz = normal segment (0-17), w = random seed 2
};
```



# Draw calls

- On OpenGL ES draw calls are pretty expensive
- All material layers are packed into an atlas or texture array
- Single draw call per chunk
  - Nice side effect: material choice doesn't affect performance!
- Minimal setup per draw
  - Just need to update one uniform and vertex/index buffers

# Level of detail

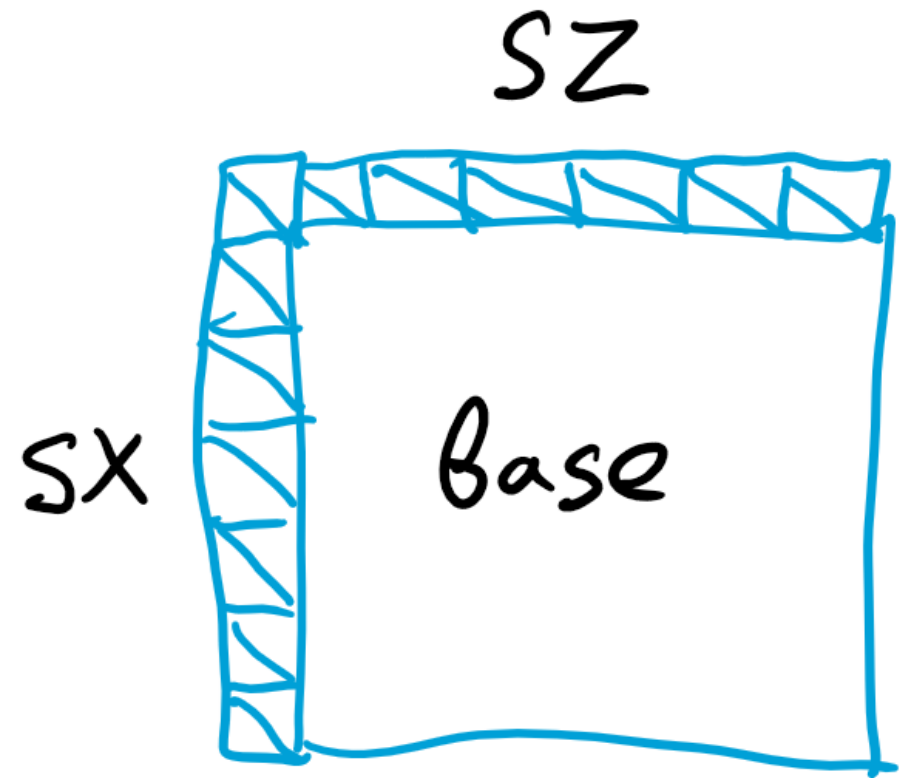
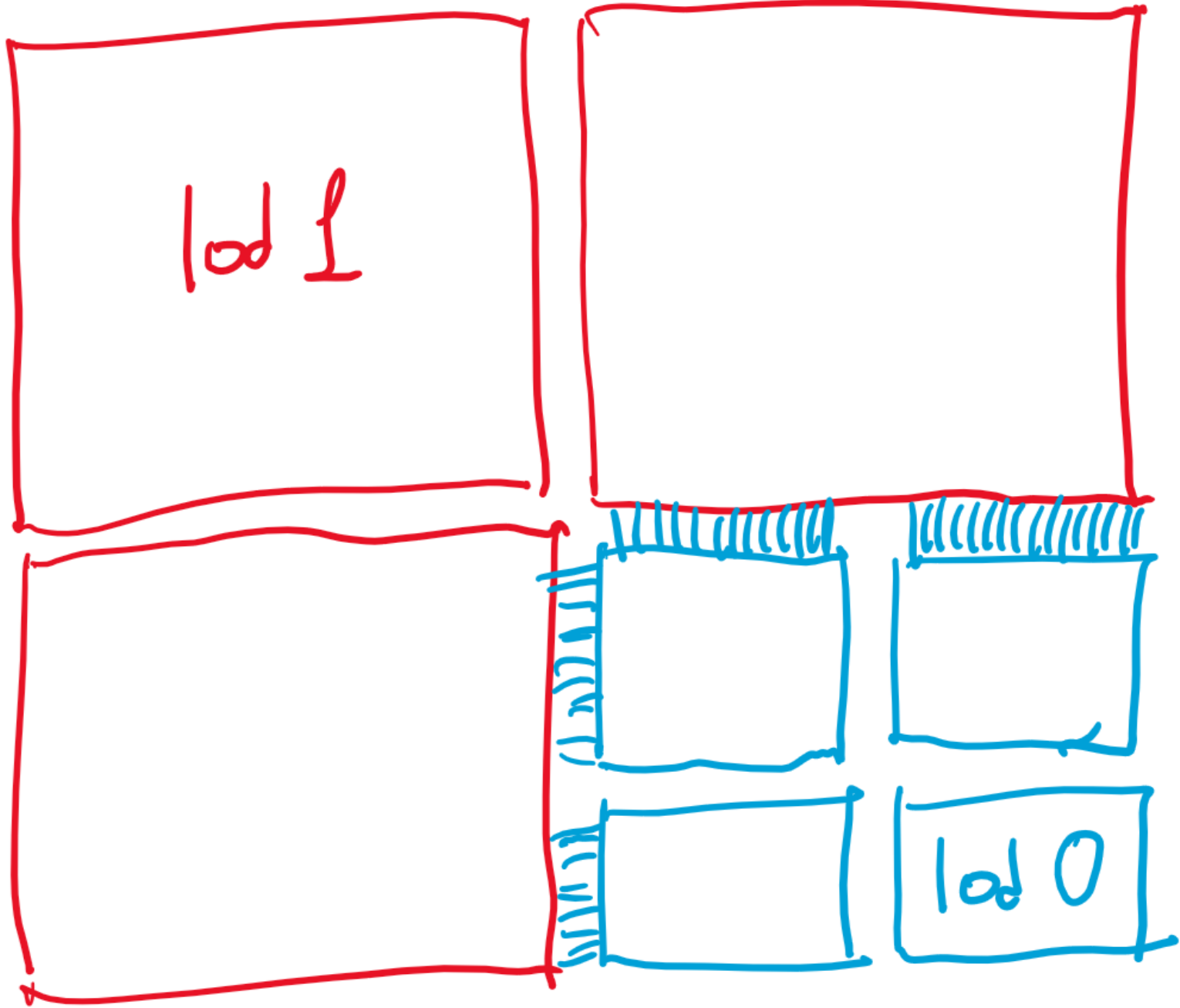
- We have to vary geometric detail for performance
- We also don't always have top mip available!
- Use octree to store render representation
- Split leaves into  $2^3$  when up close, merge leaves when far away
- Leaf size is  $16^3$  voxels (typically 500-1000 triangles)

# Level of detail: stitches

- Each node uses the meshing algorithm on a voxel mip
- We need to stitch geometry for neighboring chunks together
- Ideally, we'd generate triangles to match...
  - See "Dual Contouring of Hermite Data"
- ... but this is expensive and complicated

# Level of detail: skirts

- Instead, we generate extra overhang geometry
- Just one extra triangle is enough!
- An extra layer of voxels, patched to produce better skirts
- To avoid artifacts, we apply depth bias to skirt geometry
  - This makes sure skirts are only visible in gaps
- This is cheap enough to generate for every chunk

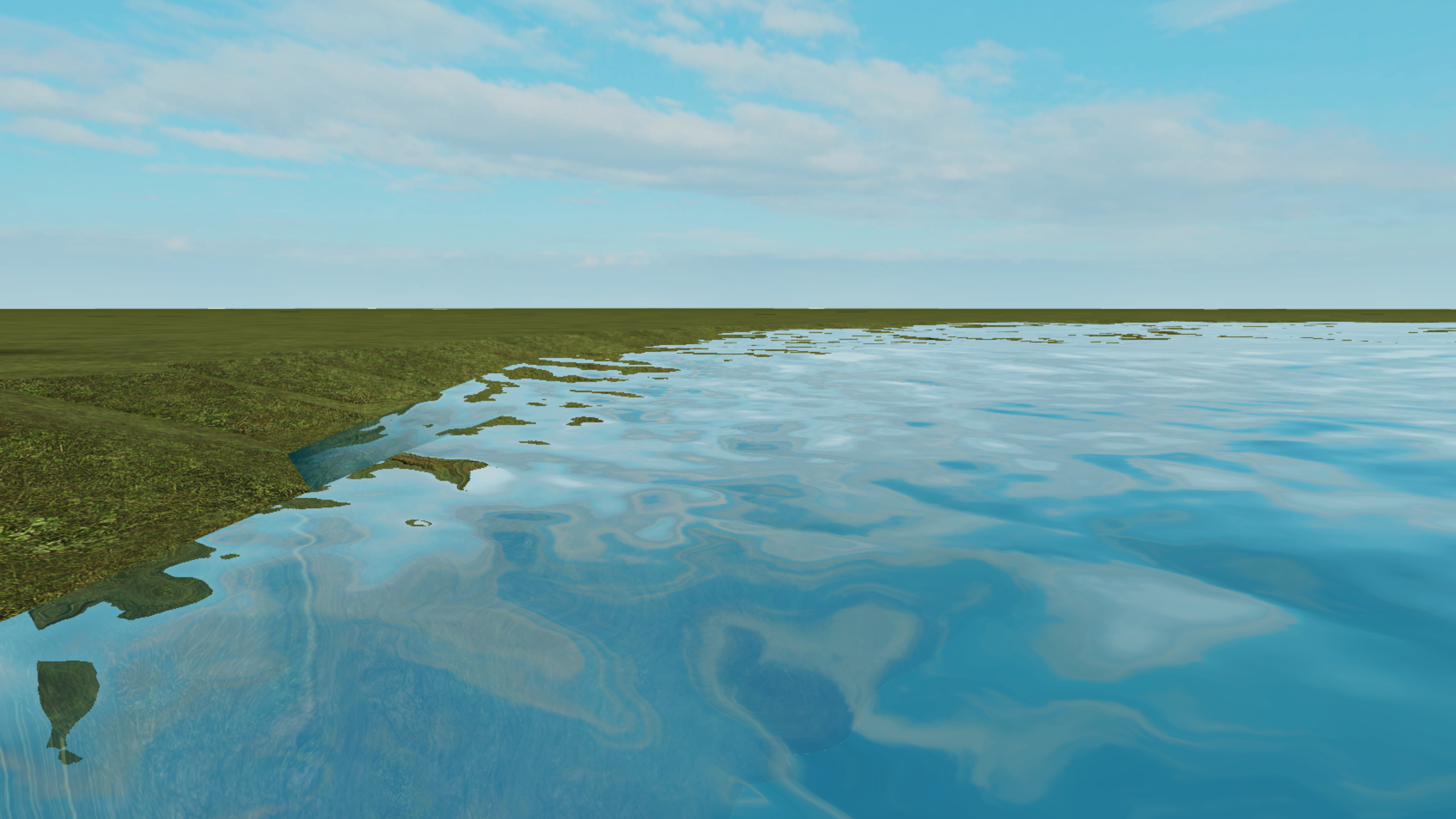


# Level of detail: stitch rendering

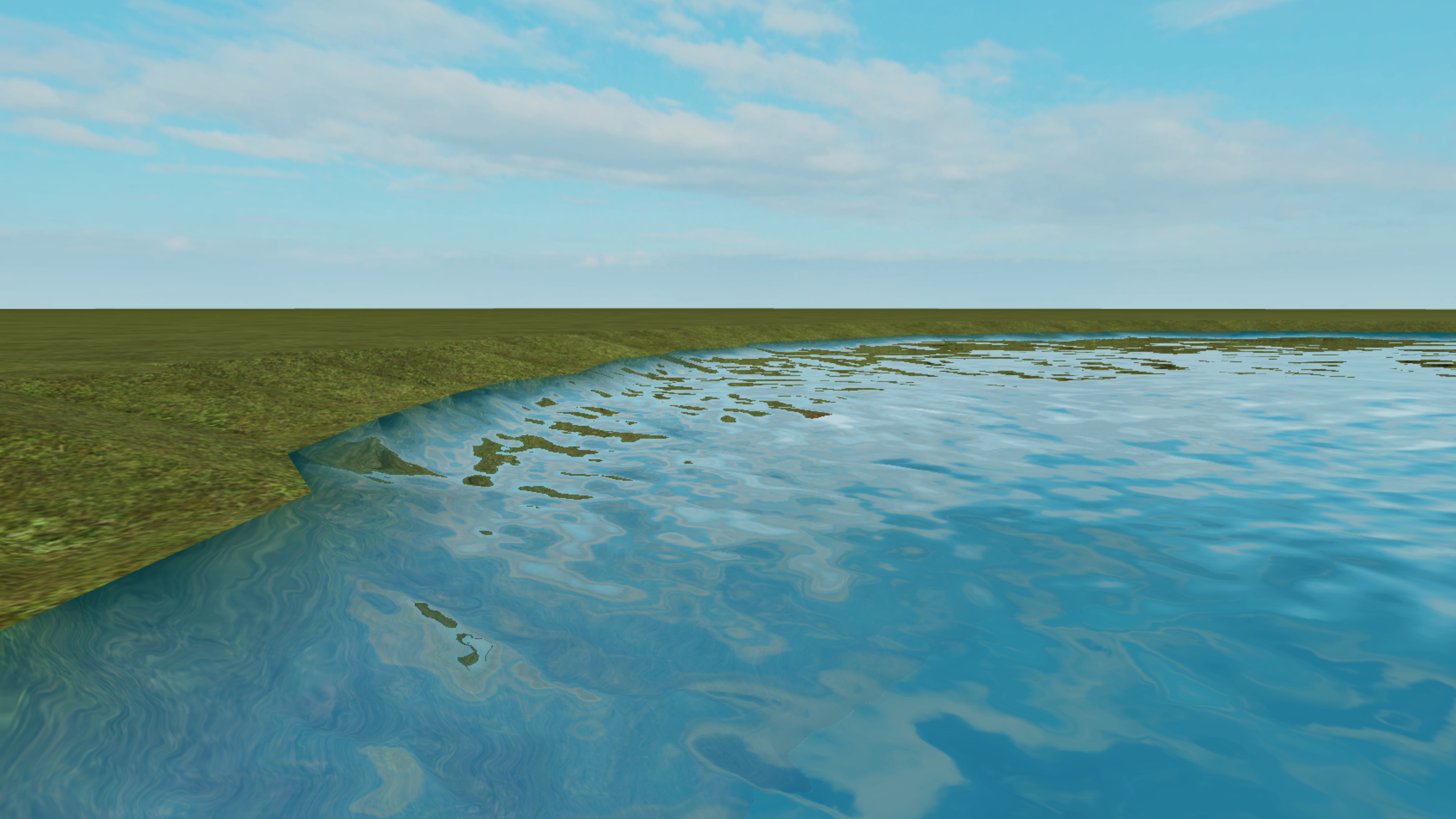
- Conveniently, each chunk has up to 3 stitches to render
- ... and Y stitch is almost never used
- Special index buffer layout: [stitchX] [base] [stitchZ] [stitchY]
- Single draw call to render base with any XZ stitches!
- Stitch vertices tagged in vertex data to apply depth bias in VS

# Water

- Water is translucent and therefore needs special care...
- We mesh solid-air, solid-water and water-air interfaces
  - Done during a single meshing pass
- Water has special material-aware deformer to avoid bulging
- Water geometry is rendered separately

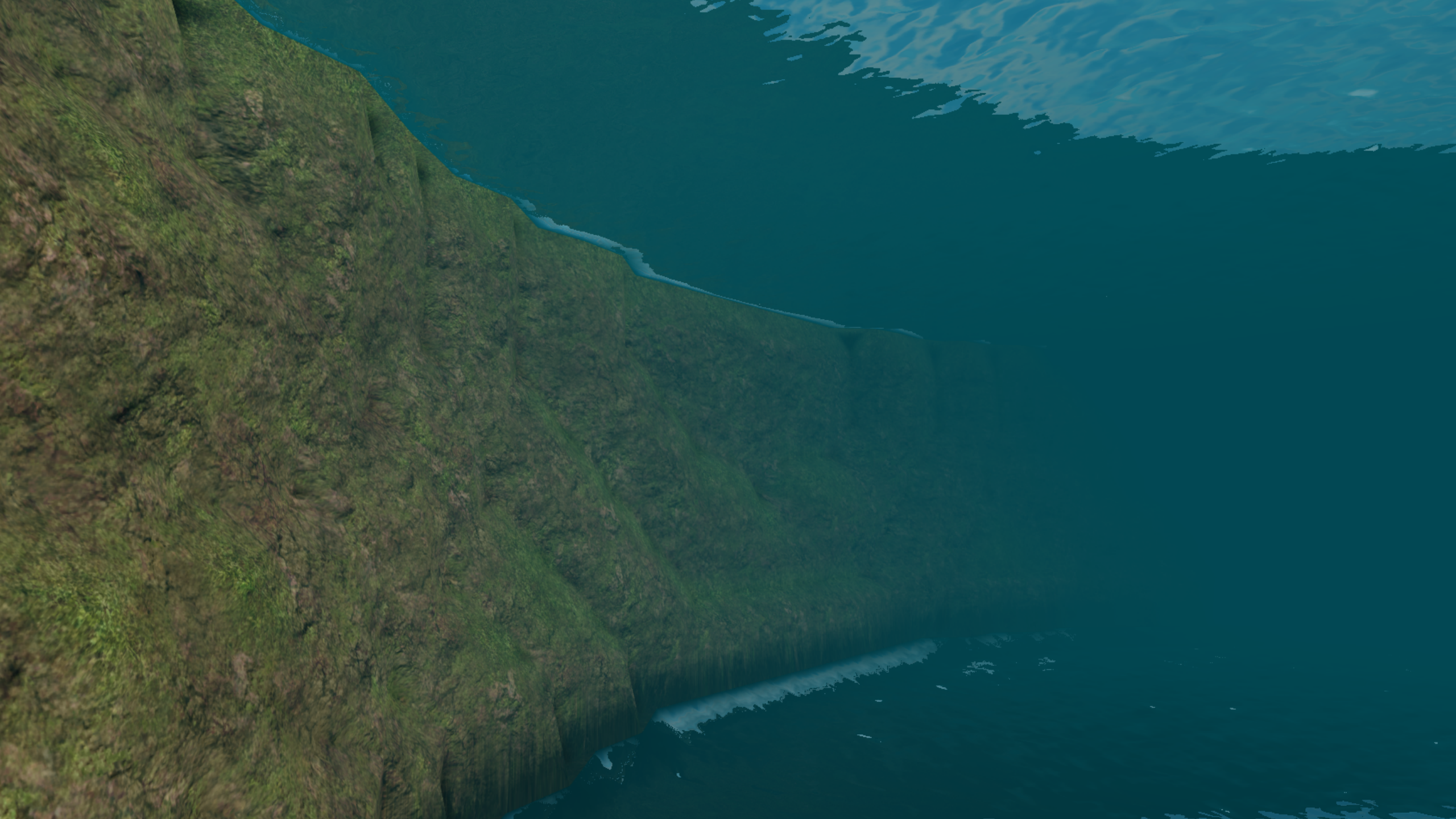


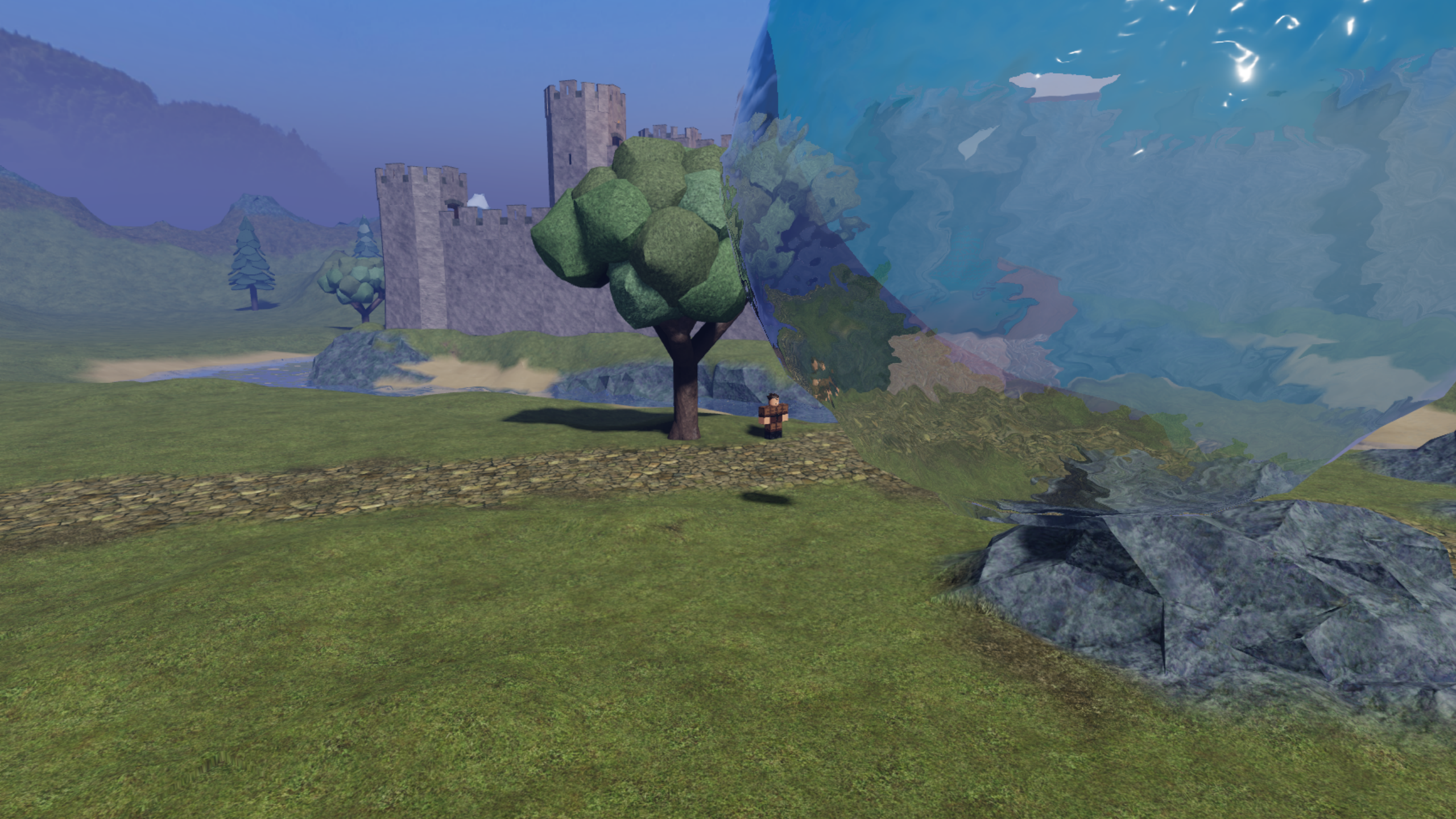




# Water rendering

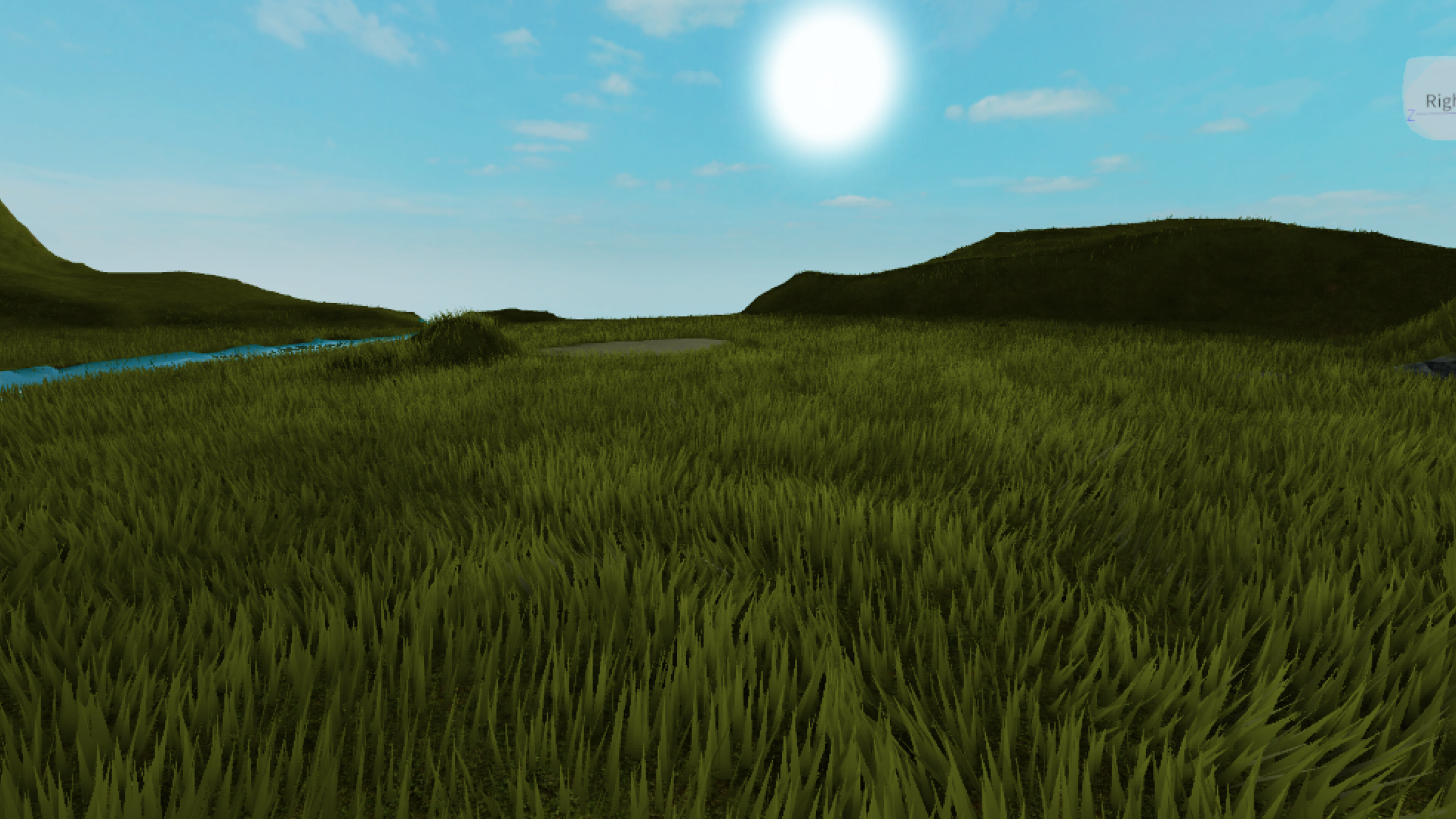
- Scrolling tiled animated normal maps (with detiling)
- Geometric waves
- Underwater fog
- On desktop:
  - Screen-space refraction
  - Screen-space reflection with 8-tap ray trace
- Future: better shorelines





# Grass

- How can we make materials even smarter?
  - Future: higher fidelity geometric shaping?
  - Present: clutter
- Experimented with card-based and geometric grass
- Geometric grass was noticeably faster on tilers



Right



# Grass rendering

- 3-5 vertices per grass blade (level of detail)
- Grass points placed using vertex seeds (stable randomness)
- Very custom shading to "approximate" the look
  - Wrap diffuse
  - More translucency-related ~~hacksmath~~
  - Height-based gradient for diffuse / specular





# Future work

- Higher fidelity geometry
- Higher fidelity shading
- Better texture mapping
- Scaling beyond 1B voxels

**Thank you!**