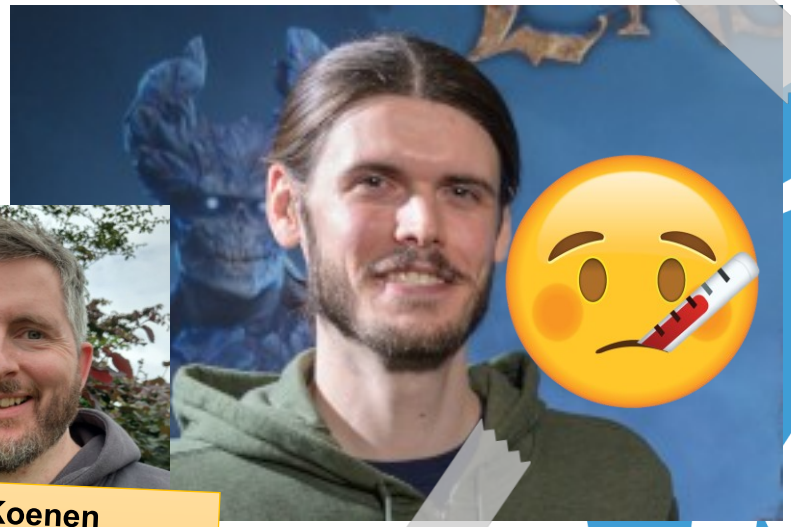
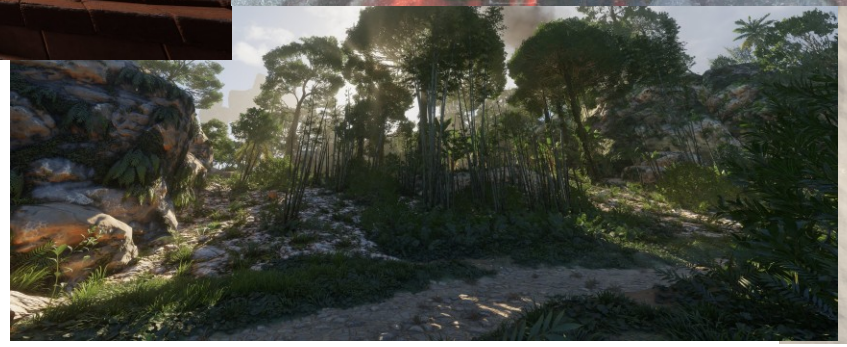


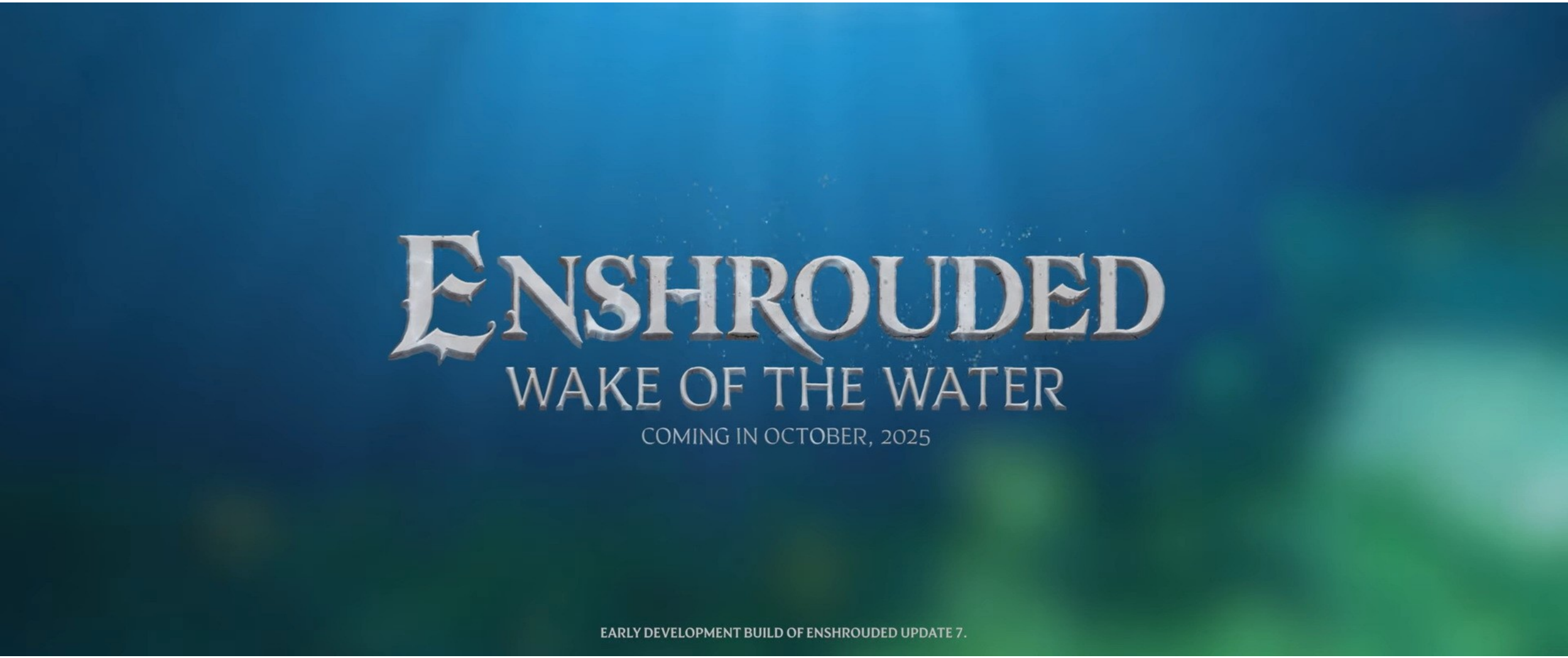
Julien Koenen
Technical Director
@ Keen Games

Simon René Stempfle
Junior Programmer
@ Keen Games since 2023

Andreas Mantler
Senior Graphics Programmer
@ Keen Games since 2024

What is Enshrouded?




ENSHROUDED
WAKE OF THE WATER
COMING IN OCTOBER, 2025
EARLY DEVELOPMENT BUILD OF ENSHROUDED UPDATE 7.

Why Water?

Requirements

- Dynamic / Simulated
- Work with dynamic 3D voxel based world
- Gameplay interactions: Players, NPCs, Enemies, Fire, …
- Believable and Predictable Movement
- Soft Real-Time
- Fixed memory budget
- Multiplayer (authoritative server, replication)

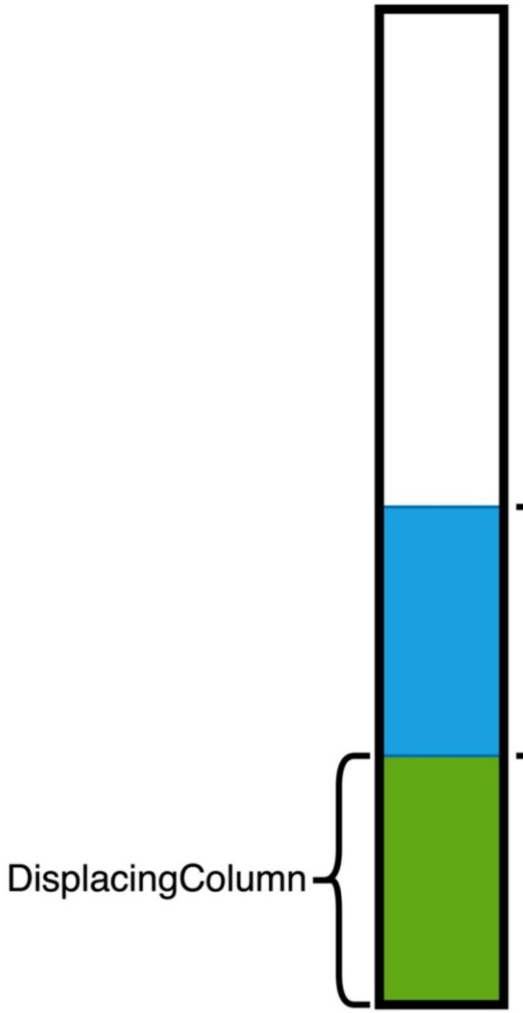
State of the Art

- Most games have static 2D planar water surfaces
  - Even close relatives: Teardown, Valheim, …
- Simulated water is usually limited
  - Hydrophobia, From Dust, 7 Days to Die: 2.5D simulation
  - Minecraft: Voxels expand by limited distance
- Academic Research usually too fine-grained/cost intensive and not client-server ready


PORTAL
KNIGHTS

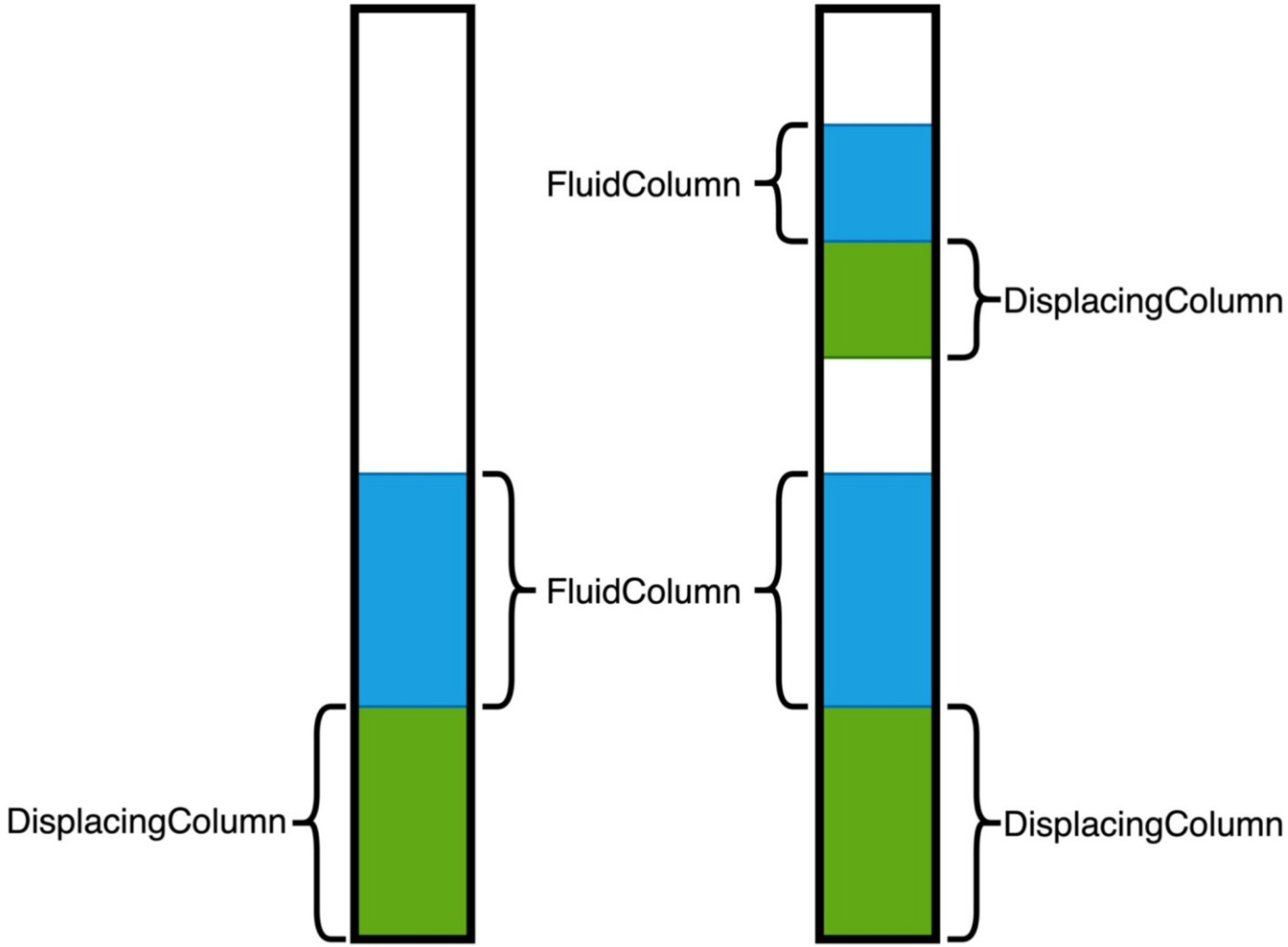
Simulation – Columns

- Bottom & Height
- Fluid section
- Displacement section
  - Terrain/buildings/blocker

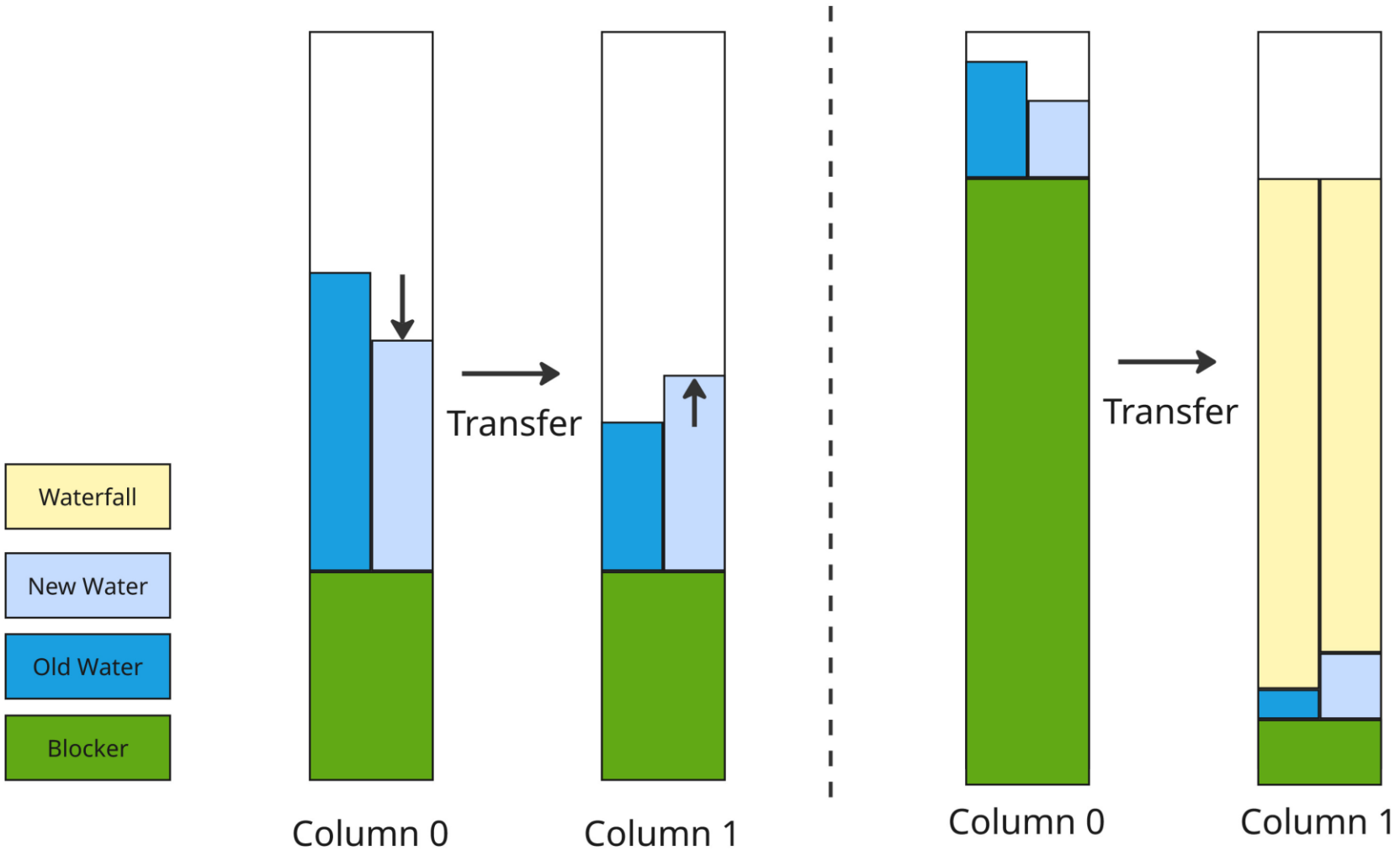


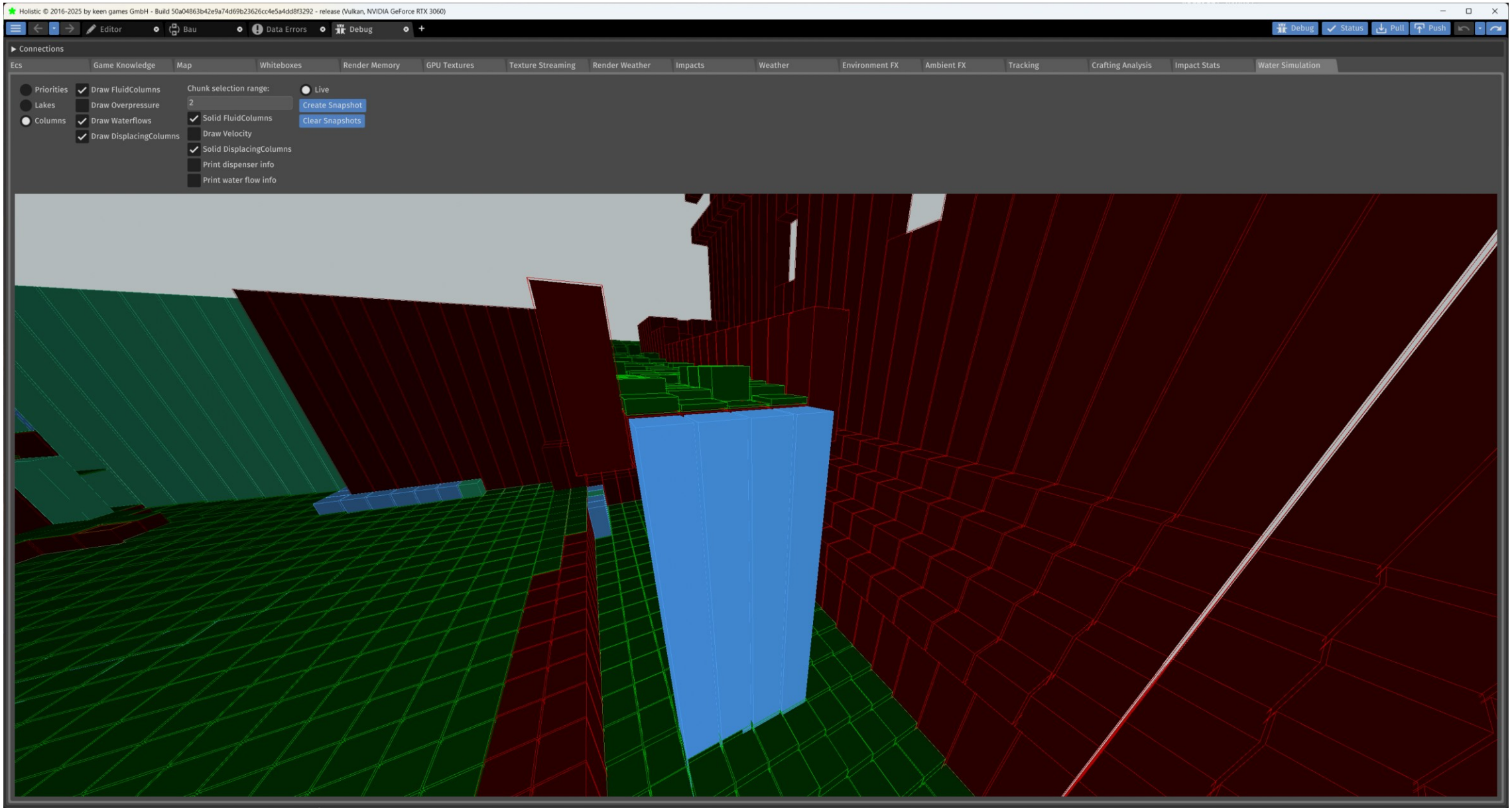
Simulation – Columns

- Bottom & Height
- Fluid sections
- Displacement sections
  - Terrain/buildings
- Stackable!

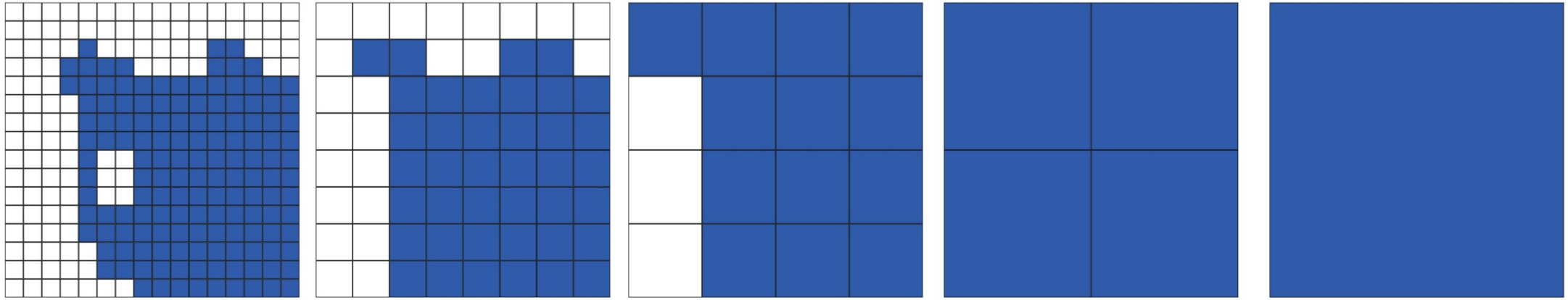


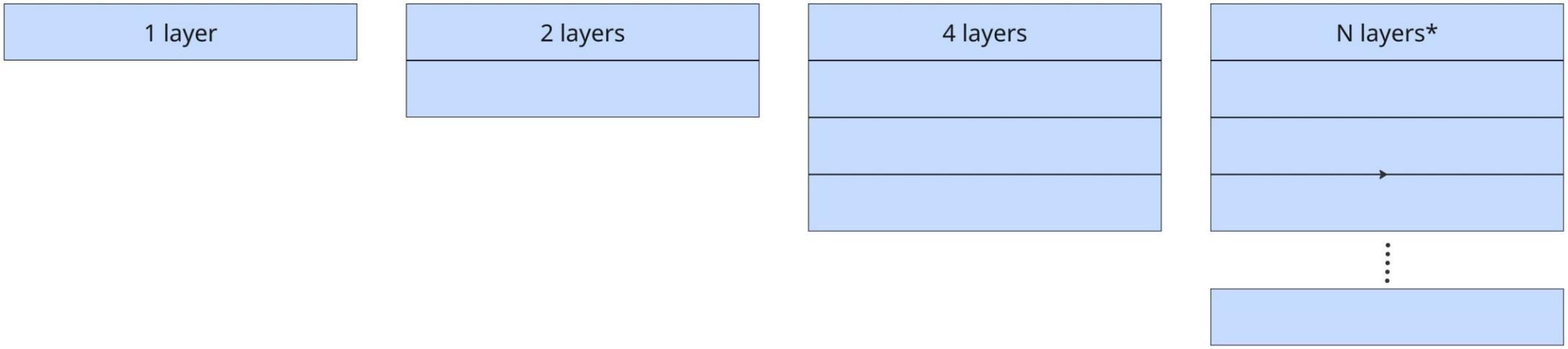
Simulation - Columns

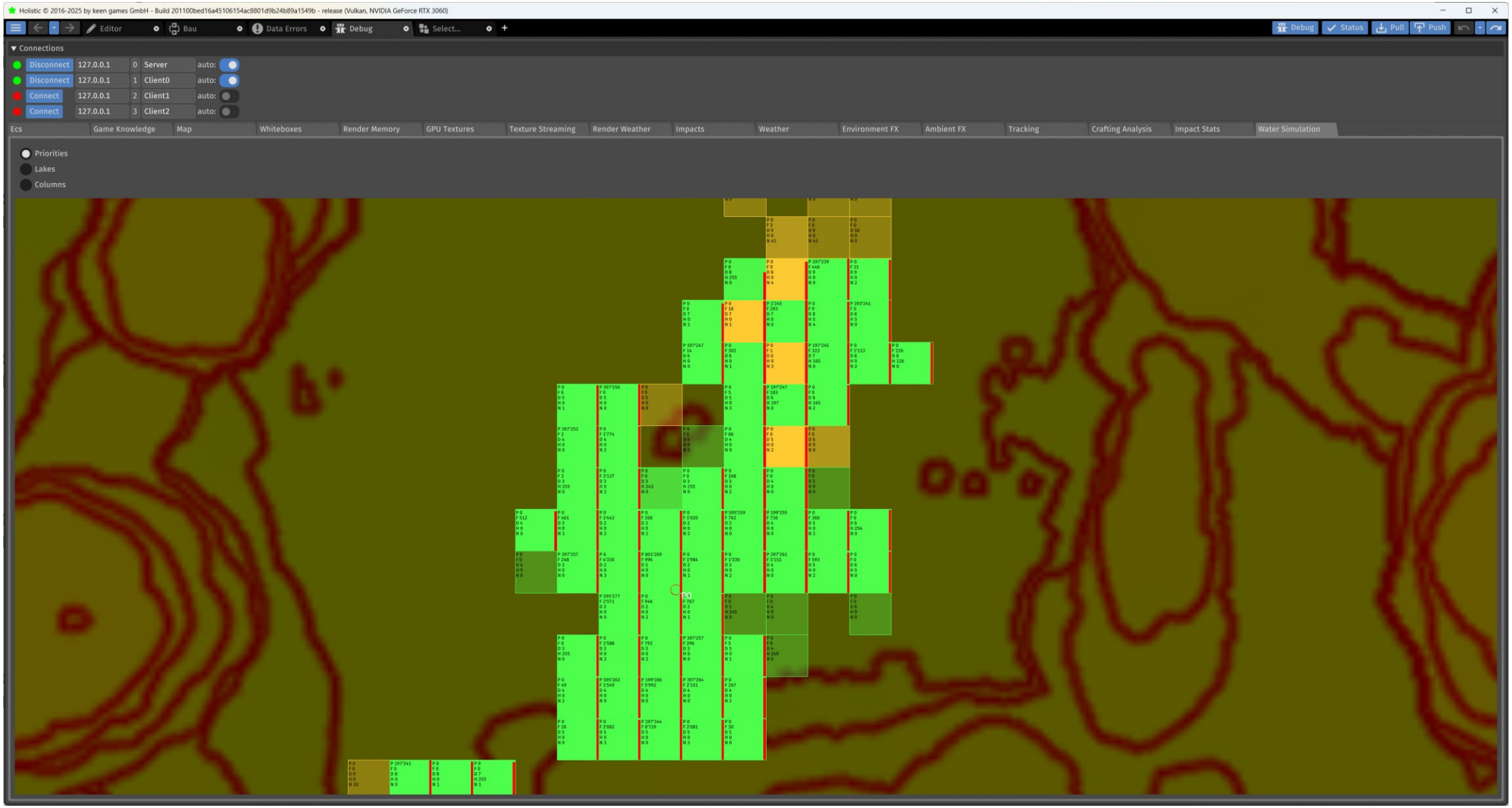



Holistic © 2016-2025 by keen games GmbH - Build 50a04863b42e9a74d69b23626cc4e5a4dd8f3292 - release (Vulkan, NVIDIA GeForce RTX 3060)
Editor
Bau
Data Errors
Debug
Debug
Status
Pull
Push
Connections
Ecs
Game Knowledge
Map
Whiteboxes
Render Memory
GPU Textures
Texture Streaming
Render Weather
Impacts
Weather
Environment FX
Ambient FX
Tracking
Crafting Analysis
Impact Stats
Water Simulation
Priorities
Lakes
Columns
Draw FluidColumns
Draw Overpressure
Draw Waterflows
Draw DisplacingColumns
Chunk selection range:
2
Solid FluidColumns
Draw Velocity
Solid DisplacingColumns
Print dispenser info
Print water flow info
Live
Create Snapshot
Clear Snapshots

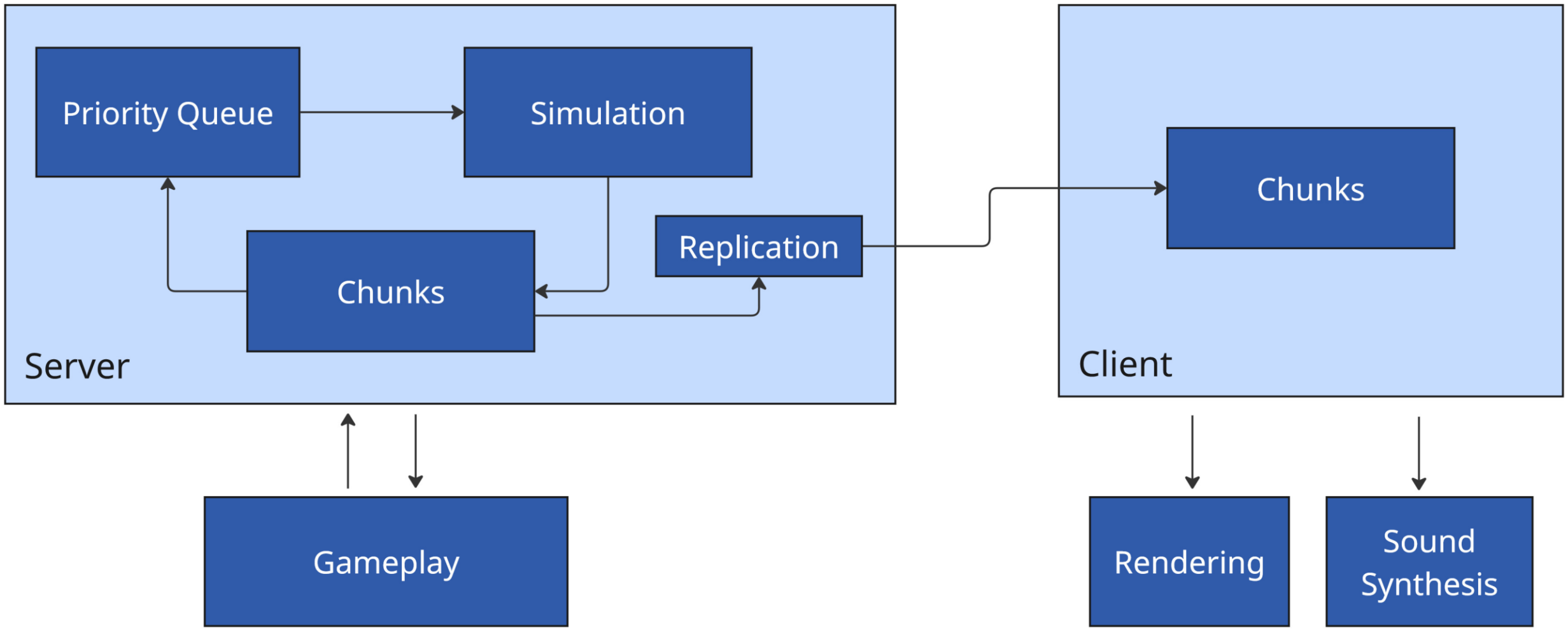
Simulation – Chunks






Holistic © 2016-2025 by keen games GmbH - Build 201100bed16a45106154ac8801d9b24b89a1549b - release (Vulkan, NVIDIA GeForce RTX 3060)
Editor
Bau
Data Errors
Debug
Select...
Debug
Status
Pull
Push
Connections
Disconnect 127.0.0.1 0 Server auto:
Disconnect 127.0.0.1 1 Client0 auto:
Connect 127.0.0.1 2 Client1 auto:
Connect 127.0.0.1 3 Client2 auto:
Ecs
Game Knowledge
Map
Whiteboxes
Render Memory
GPU Textures
Texture Streaming
Render Weather
Impacts
Weather
Environment FX
Ambient FX
Tracking
Crafting Analysis
Impact Stats
Water Simulation
Priorities
Lakes
Columns

Simulation


Priority Queue
Simulation
Chunks
Replication
Server
Chunks
Client
Gameplay
Rendering
Sound Synthesis

Rendering Water

- Focus here is **GPU Representation** and **Surface Phenomena**

- For (water) volumetrics please check out Philipp Krause's talk:
  - "The Fog is Lifting, Volumetric Rendering in Enshrouded"

- For more (water?) VFX tech insights there's Lukas Feller's talk:
  - "Lessons learned from shipping a GPU Particle System"

From Columns to Voxels

- System dependency: Renderer -> Water Simulation Client
- Everything voxelized here
- Fetch queued “dirty boxes” each frame to trigger updates
- Dirty boxes are 3D voxel bounding boxes
- Simple interface:

```
bool fillWaterVoxelArray( const WaterSimulationClient* pWater, ArrayView<uint8> buffer, uint3 bufferSize, uint3 bufferStride,
                          uint level, int3 position, ArrayView< ColumnData > columnData, bool includeWaterFlows = true );

bool hasDirtyBoxes( const WaterSimulationClient* pWater, uint32 since );
bool getDirtyBoxes( Slice< WaterBox >* pDirtyBoxes, const WaterSimulationClient* pWater, uint32* pSince );
```

Data Considerations

Observations:

- Way too many 0.5^3m^3 voxels in a 10k x 10k x 4k world
- Most voxels are either completely empty or completely filled with water

$\Rightarrow$Voxels containing a water surface are most interesting

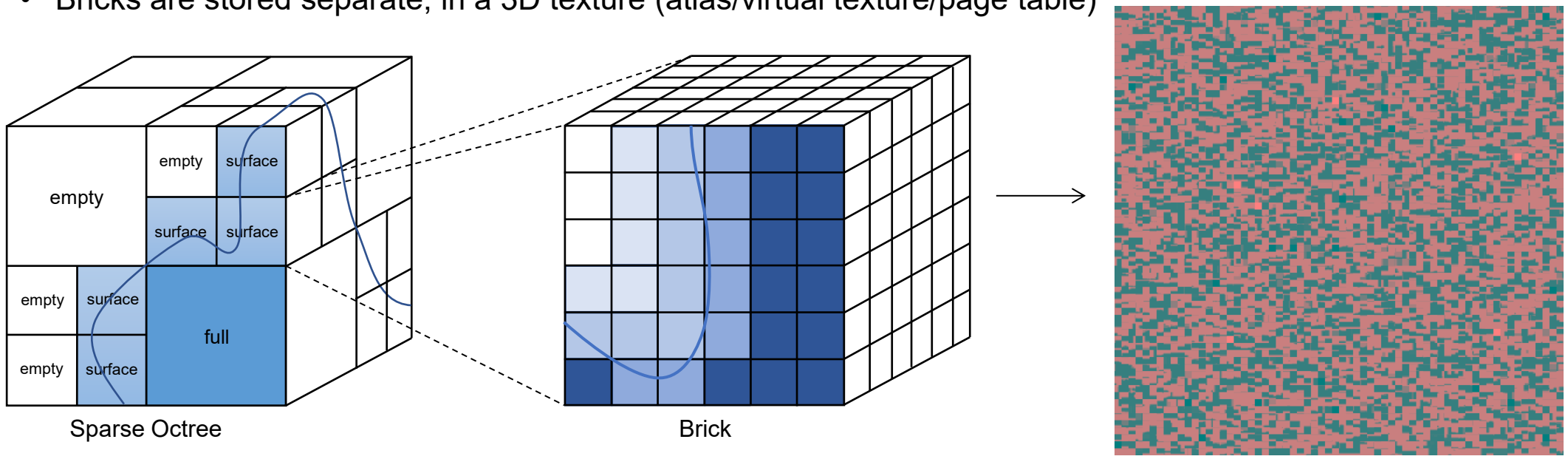
- Voxels are blocky, but we want a smooth representation

Idea:

- Convert voxels to SDF grid only close to water surfaces
- Store SDF in a sparse GPU-friendly data structure
- Partition sparse space into either full or empty

The Brick Tree

- Sparse octree, nodes with water surfaces are subdivided to desired LOD
- Empty and full octree nodes do not get subdivided
- Leaf nodes are “Bricks”
- Brick: 6x6x6 8bit SDF values with interpolation border (so 8x8x8 = 512bytes)
- Octree nodes are allocated from a node pool buffer, synced between CPU & GPU
- Bricks are stored separate, in a 3D texture (atlas/virtual texture/page table)



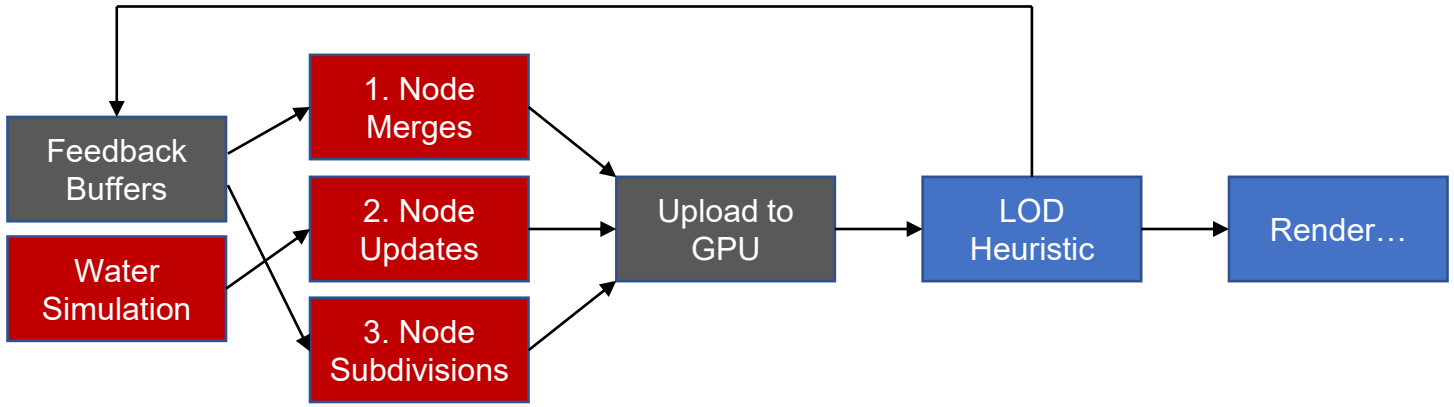
Managing the Brick Tree

Want to change nodes from both CPU and GPU

- E.g. changes from simulation, LOD changes from compute shader
- But staging / readback delays make data structure sync difficult

Solution:

- CPU: solely responsible for brick tree changes
- GPU: output LOD feedback buffers with versioned requests
- CPU: Ignore requests with node data version mismatches
- CPU: Apply remaining N items, sorted by priority



Managing the Brick Tree

Data requests from Simulation take time

⇒ handled async (task threading)

⇒ Brick Tree operations can take 2 update cycles

⇒ State machine in each node:

```
enum class WaterBrickTreeNodeState : uint8
{
    Unallocated = 0u,
    Loaded,
    Updating,
    Subdividing,
    ChildrenLoaded,
    Merging,
    ParentMerging,
    Count,
};
```

Growing a Brick Forest

Problem:

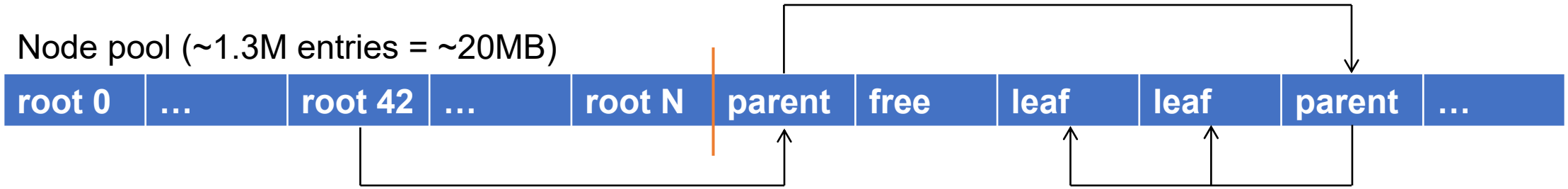
- Simulation API only provides 5 LOD levels max (chunks are 16x16 columns)
- Single big brick tree hierarchy too deep for fast spatial GPU lookups & tracing rays

Solution: Truncate the hierarchy at the top by 7 levels

- Forest of 107x30x107 smaller brick trees, each covers $96^3 m^3$
- Each root node pre-allocated, stored dense at the beginning of a node pool

⇒ Lookup for the correct tree for a world position is O(1)

⇒ Only levels that could actually contain any data need to be traversed!



Liquifying the Bricks

Problem:

- Simulation state updates irregularly (whenever new network packets arrive)
- Different/lower frequency than output frame rate
- Quantized to only 16 fill states per voxel

⇒ Each change is a big visual discontinuity (popping)

Solution:

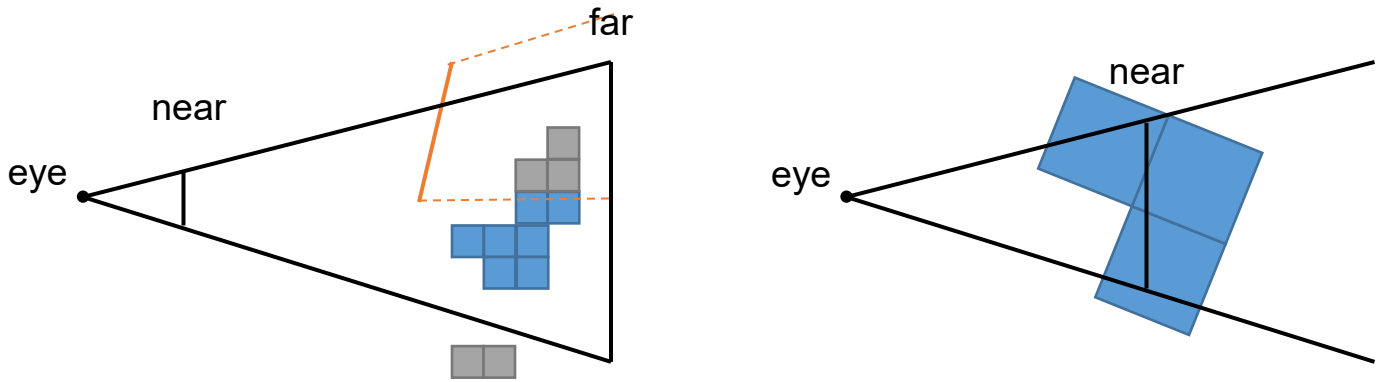
- Bricks can either be **static** or **interpolating**
- Interpolation:
  - Lerp 8bit SDF data towards a separate target state in the page table
  - Temporally dither for sub-8bit perceived precision steps
- Subdividing nodes transitions instantly, popping no big problem -> far away
- New Problem: Only GPU knows which bricks finished interpolating

⇒ Feedback buffer to free up page table entries on CPU

Liquifying the Bricks

Visibility Culling

- Compute shader scans all leaf nodes. For each leaf AABB:
  - Frustum culling
  - Occlusion culling against depth pyramid of scene
  - Visible? => Append brick index to buffer for drawing

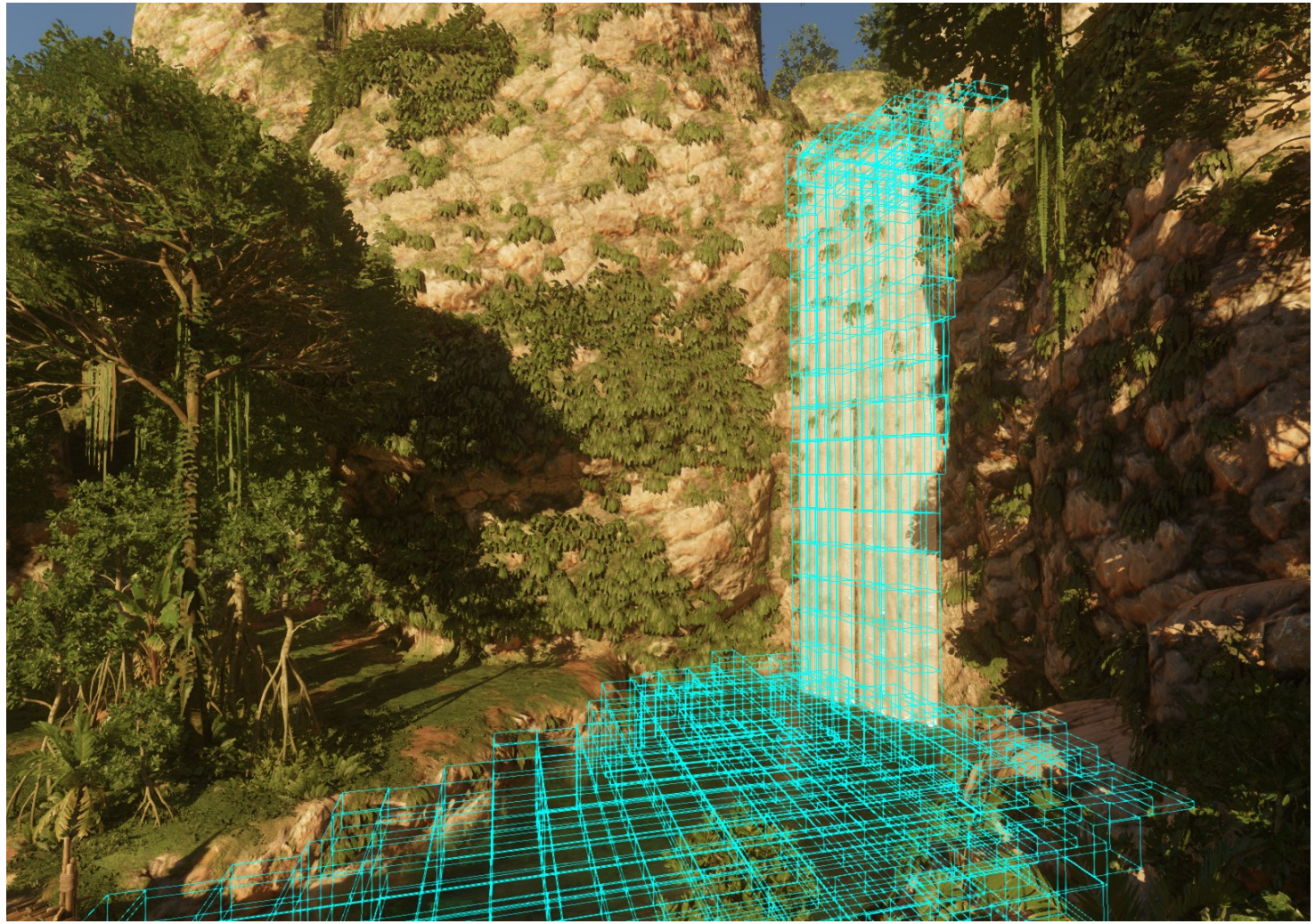


- Edge case: Bricks intersecting the camera near plane
  - We support both over and under water pixels on screen at the same time
  - One extra screen space draw if camera is close to or inside water

Surface Rendering: Brick AABBs

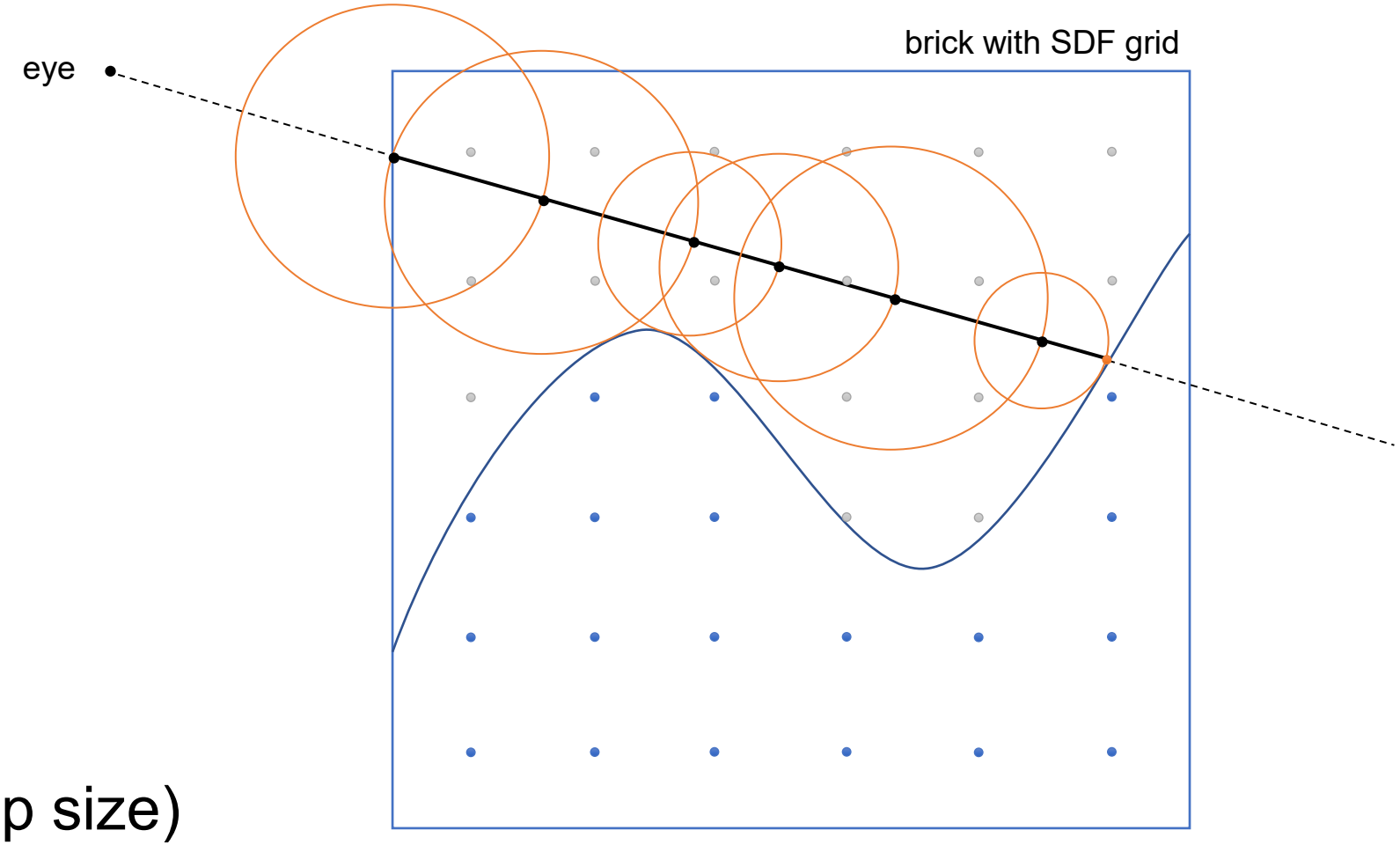
Instanced indirect draw of culled brick AABB front faces

AABBs are optimized to more tightly wrap where the surface is



Surface Rendering: Sphere Tracing

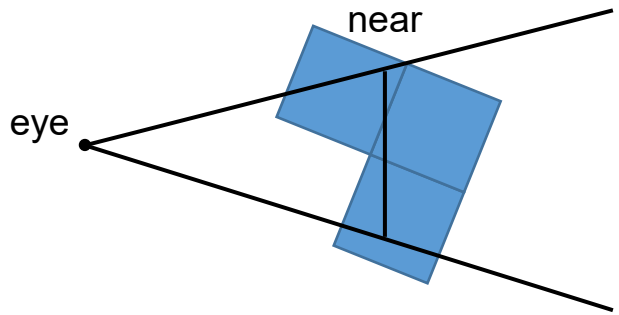
For each brick pixel:
SDF sphere trace
through the brick



(SDF range == max step size)

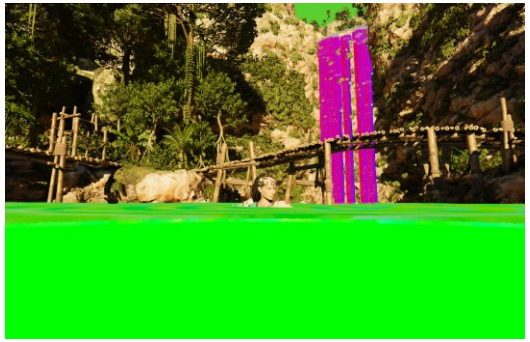
Surface Rendering: Screen Space Pass

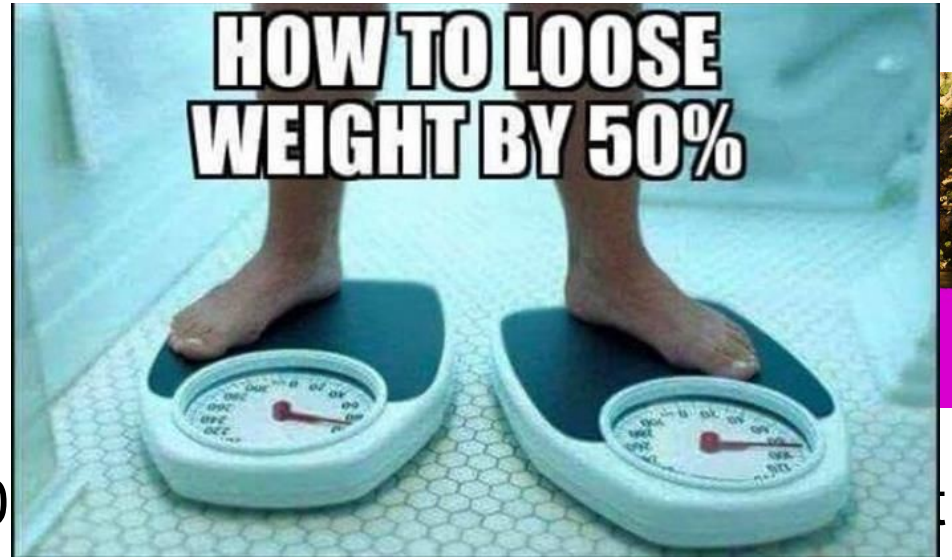
- Similar to brick AABBs pass
- Start points either within one or more bricks (or none)
- Which ones? => Per pixel brick tree lookup at near plane
- Calculate ray origin in brick, sphere trace from there

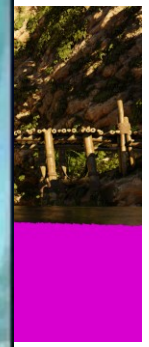


- We now know:
  - If pixel hit a surface
  - If pixel is under water (from SDF sign)
  - Surface depth (project hit position to screen)
  - Surface normal (from SDF derivatives)

Surface Buffer

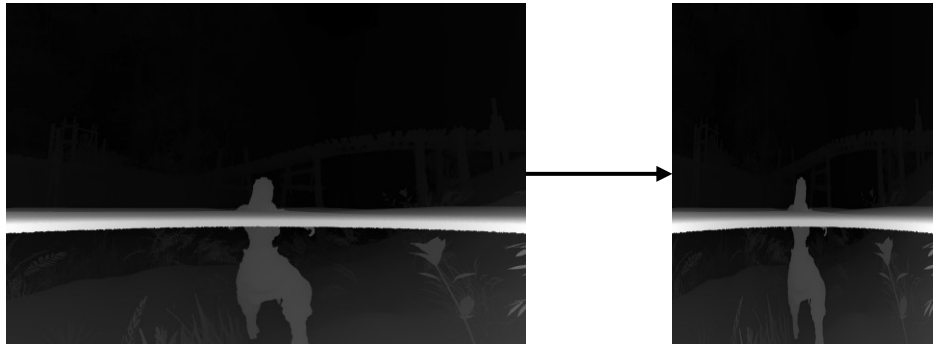






Small gbuffer with 32bit Depth & R10 t containing:

- 21bit octahedral encoded normal
- 8bit foam amount
- 1bit surface hit mask
- 1bit underwater mask
- 1bit particle mask (not shown)



Output in horizontal-half resolution [Grujic18], [Geffroy20]

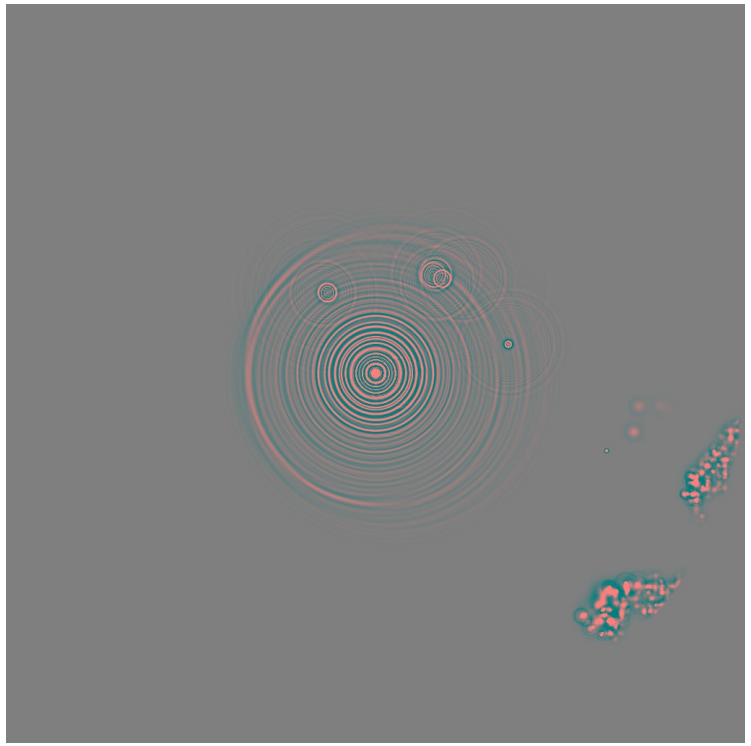
Surface FX: Waves

- In-place effects on the water surface buffer via compute passes
- Waves:
  - **Close-up waves:** Detail normal maps
  - **Far waves:** Procedural FBM noise with 2-3 octaves -> no tiling issues
  - No big FFT/Tessendorf wave simulation here (stay close to simulation output)



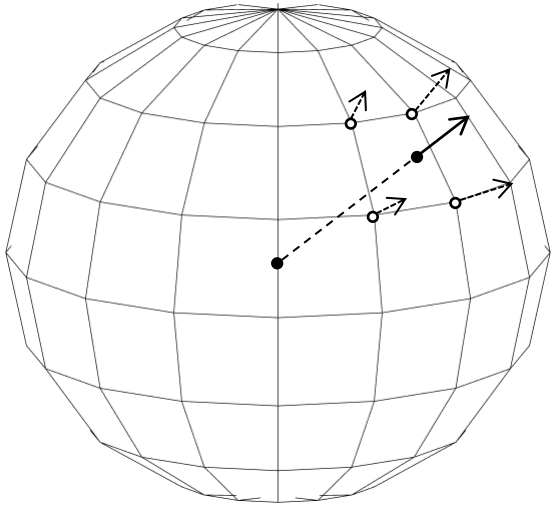
Surface FX: Ripples

- Examples: player/enemy movement, rain, projectile hits, water splash impacts
- Simple projected planar 2D simulation with just amplitude + vertical velocity
- 1024² pixels => limited range around player camera
- Inject shapes from GPU VFX particle shaders to displace the water surface



Surface FX: Flow Mapping

- No velocity data from simulation
  ⇒Flow from SDF gradients and wind
- Change scrolling material with increasing slope
  - waves => slow flow => white water => waterfall
- UVs: position projected onto plane from quantized direction angles/slopes
- Tangent basis from UVs & view vector derivatives (QuadReadAcrossX/Y())
- Bilinear interpolation between 4 neighbour UV mappings and materials

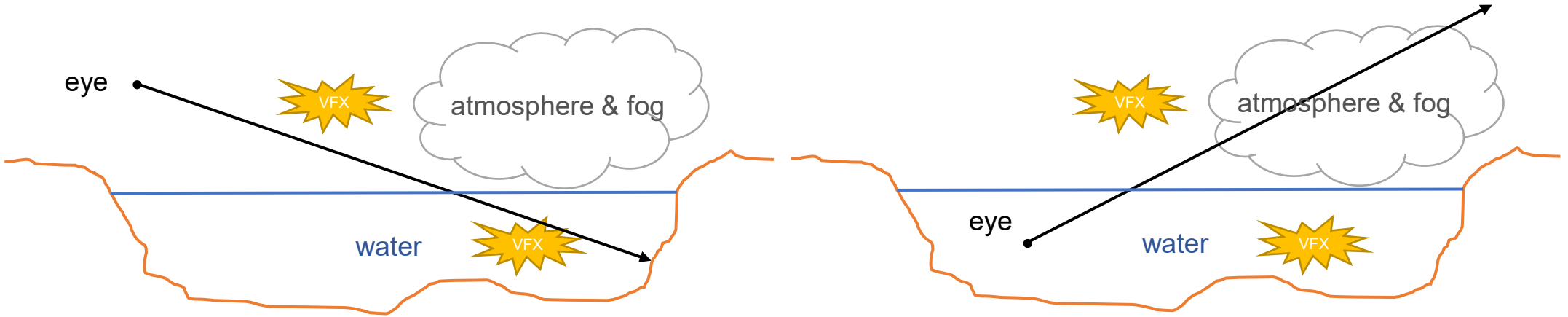


Compositing

- Surface Buffer is single air -> water, or water -> air interface
- Flips order of compositing volumetrics and VFX against the interface

Back-to-front rendering after opaque deferred shading:

1. Behind water surface: Early volumetrics & VFX
2. Water surface: Refractions, reflections, foam material, depth-aware upsampling
3. In front of water surface: Late volumetrics & VFX



Compositing: More Layers?

- Unbounded number of media transitions, refractions, reflections,…

⇒ More than one layer is complex, even if OIT is used

- Attempted opaque second layer written to gbuffer, but didn’t look good enough

⇒ May want to revisit

- Low resolution ray-march through all water layers

⇒ Estimate for ratio of air vs water, used as layer thickness for volumetrics

Find the Water

Surface Shading: Refractions

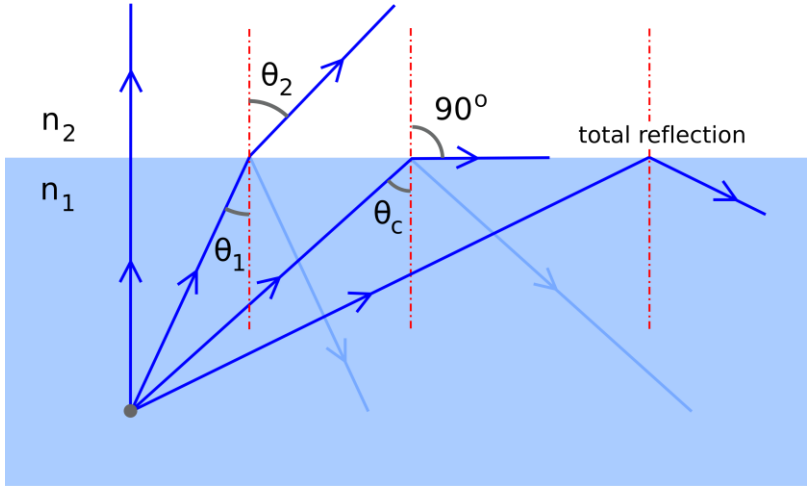
- Screen space, lots of edge case handling, but fast and still looks acceptable
- Classic trick: Offset UVs with surface normal and ground distance
- Not physically based (would require even more tracing of rays outside screen)

Surface Shading: Reflections

- Second SSSR on water surface buffer (@horizontal half resolution)
- Fallback: GI probe reflected radiance (low res, lower end settings)
- Fallback (new): World rays + sky cube (higher res, high end)

Surface Shading: Total Internal Reflections

- Based in reality
- Critical angle ~49°
  (water ⇔ air)
- Refraction angle hits limit:



By Jfmelero (adapted by Gavin R Putland).
File:ReflexionTotal.svg — subsequently translated and retouched., CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=77502540

Surface Shading: Lighting & Foam

- “Foam amount”: blend value between water surface and foam PBR material
- Foam does simple wrapped lighting diffuse to emulate (sub-surface) scattering
- Foam amount together with Fresnel term controls surface opacity

Volumetrics

Many volumetric phenomena in-game:
Fog, Clouds, Weather, Atmosphere, and now Water.

For more details:
=> Philipp Krause's talk on Volumetrics in Enshrouded

Main takeaways:

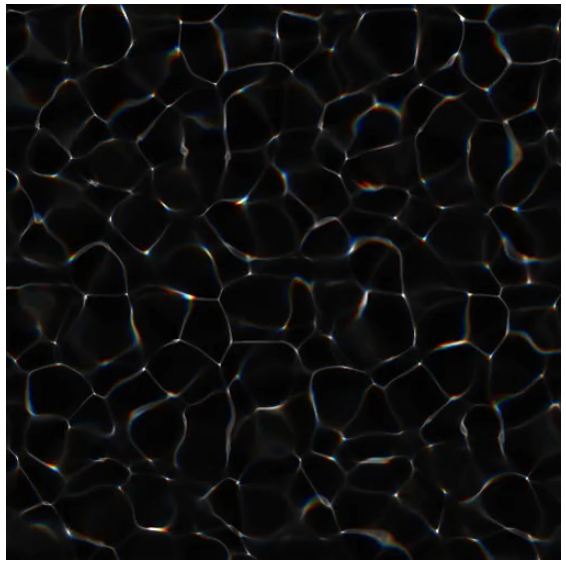
- Water is a homogeneous medium in Enshrouded
- Water displaces any other inhomogeneous media (e.g. fog)

⇒ "epipolar sampling": optimization of volumetric ray march
⇒ crisp god rays with less artefacts than usually! \o/

- Sun transmittance: Penetration depth from low resolution water shadow map cascades
- Impact on GI is emulated: Estimate player depth by checking surrounding water columns (CPU)

Caustics

- Caustics are the result of refracting sun/moon light through water surface
  - ⇒Brighter areas & darker areas
  - ⇒Caustics result must average out to conserve energy
- Photon mapped caustics would still blow the budget
- Texture procedurally distorted 3x with small spectral separation
- Pre-rendered into a tile-able texture, used in:
  - Deferred shading, applied to directional lighting under water
  - Volumetrics (=> God-Rays)



Water VFX

- Transparency sorting
  - Clouds, atmosphere, fog, water surface, underwater volume, VFX
  - For water surface: Split visible VFX into “render before water” and “render after water”
  - Intersecting VFX: rendered twice with per-pixel test against water surface depth

- Generate spawn points for GPU based VFX system
  - Procedurally scatter points on waterfall columns and splashes underneath waterfalls
  - Brick interpolation can output “wavefront” points for horizontally expanding water

  ⇒ More info on our VFX system in Lukas Feller’s talk

- Track on GPU if skinned mesh bones were in water or in rain

⇒ Marking them as wet modifies PBR material roughness etc.

Shorelines - “The Big Edge-Case”

Idea:

- Generate 2D coast distance data on flat water surfaces
- UVs from distance gradient & position
- Tangent basis from UVs & view vector derivatives
- Procedurally animate shoreline waves across the distance

But we have dynamic 3D water!?

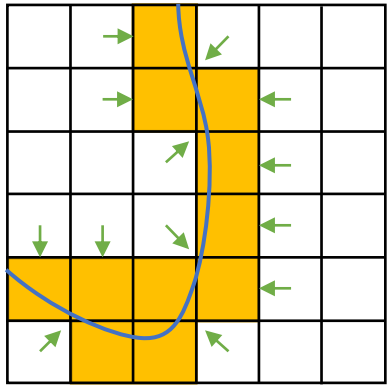
Shorelines - “The Big Edge-Case”

Additional page table with
“Brick water surface data” (R8G8B8A8):

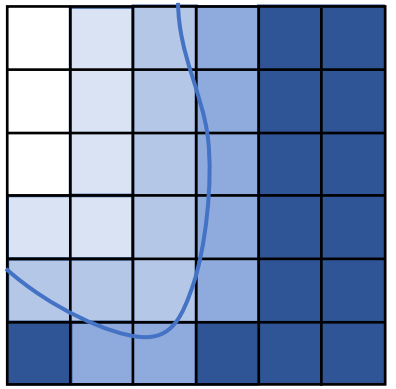
- x direction towards coast
- z direction towards coast
- 1.0f - distance to coast
- max water depth in neighbourhood (0-3m)

To fill it:

- Find intersection voxels with water surface & terrain
  ⇒ write out (0, 0, 1, 0)
- Initialize neighbour surface voxel directions
- Trace water depth (0-3m) per surface voxel
- Horizontal & vertical separable blurs/dilation
- Attenuate “illegal” shoreline gradients



Coast directions



Water depth

Rendering: GPU Memory

Brick Tree:		
Octree Node pool:	20MB	1.2M nodes
Brick SDF page table:	64MB	128k bricks
Shoreline page table:	32MB	
Rendering (horizontal half-resolution@1440p):		
Surface (depth, normals, foam, masks):	15MB + 15MB history	
Staging & Temp buffers, FX Textures, Caustics, Ripples, Motion vectors, SSSR, Epipolar, Composition, VFX sorting...	it’s complicated	

Rendering: GPU Performance

RTX 4060 Ti, “Quality” Preset

Stress Test: Fast camera motion, Water everywhere

Average: Average gameplay on/under water

Brick Tree:	Stress	Average
• Updates, Subdivisions, Merges:	200us	190us
• Shoreline updates:	125us	120us
• Culling, LODs:	125us	115us

Rendering@1440p:		
• Surface Buffer:	220us	180us
• Waves & Surface FX:	230us	205us
• Reflections (SSSR + world rays)	1.0ms	0.87ms
• Volumetrics, Refraction, Composition, …:	0.7ms	0.64ms

Rendering: The Bricked Tree

References

- [Guehl13] GigaVoxels, Real-time Voxel-based Library to Render Large and…
- [Grujic18] Water Rendering in FarCry 5
- [Kirkpatrick19] Advancements in Water and Procedural Technology
- [Geffroy20] Rendering the Hellscape of Doom Eternal
- [Mao23] Open-World Water Rendering and Real-Time Simulation
- [Lague25] Coding Adventure: Rendering Fluids

Bonus: Problems / Future work

- Geometry
  - LOD differences to terrain/building Voxels
  - Geometric differences
    - Terrain displacement maps
    - Blocking props (doors etc.) thinner than a voxel, placed freely
  - Voxel ⇔ column differences
    - fill-amount vs fill-height: edge AA, hole filling, overfilling
- Water surface
  - Tessellation
  - Layers
  - Near plane water line
    - we’re clipping at ~5cm due to sphere tracing precision
- VFX
  - Procedurally generate waterfalls?
  - Terrain & building wetness
- Performance & memory optimizations, as always ☺