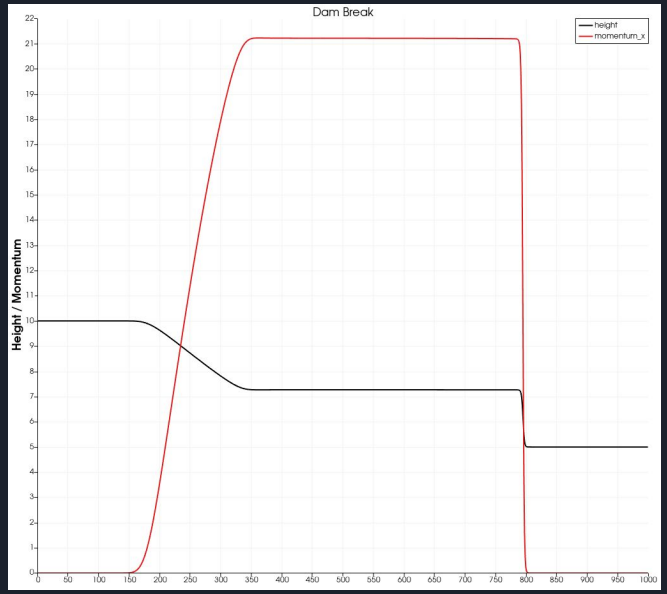
Tsunami Extension for Rem's Engine

Antonio Noack

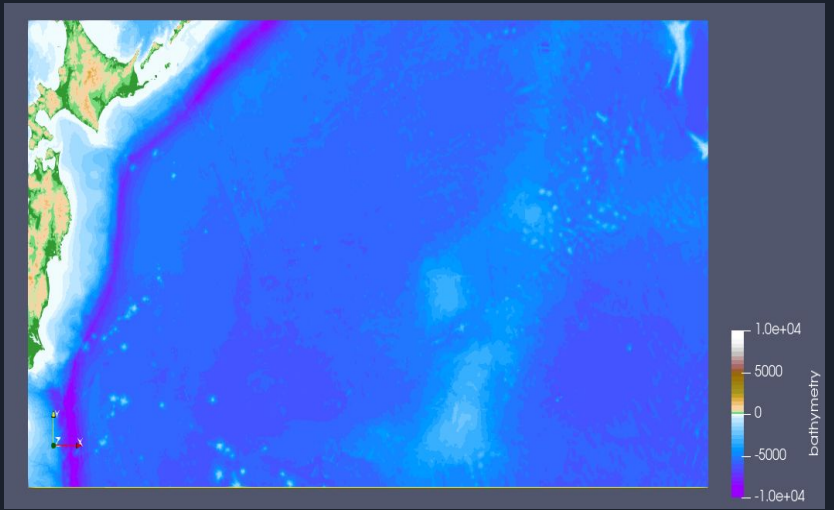
Tsunami-Lab Module

- Gained overview over solvers
- Shallow water equations
- Height, Momentum, Bathymetry - Model
- Discretization in 1d & 2d



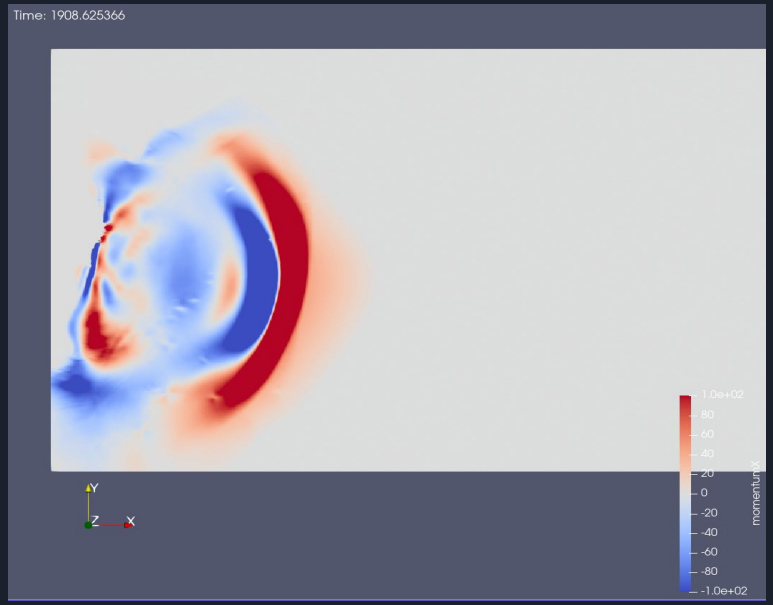
Tsunami-Lab Module

- Gained overview over solvers
- Shallow water equations
- Height, Momentum, Bathymetry - Model
- Discretization in 1d & 2d
- Outflow Boundary Conditions
- IO with NetCDF library
- Tsunami simulations: Tohoku 2011, Chile 2010

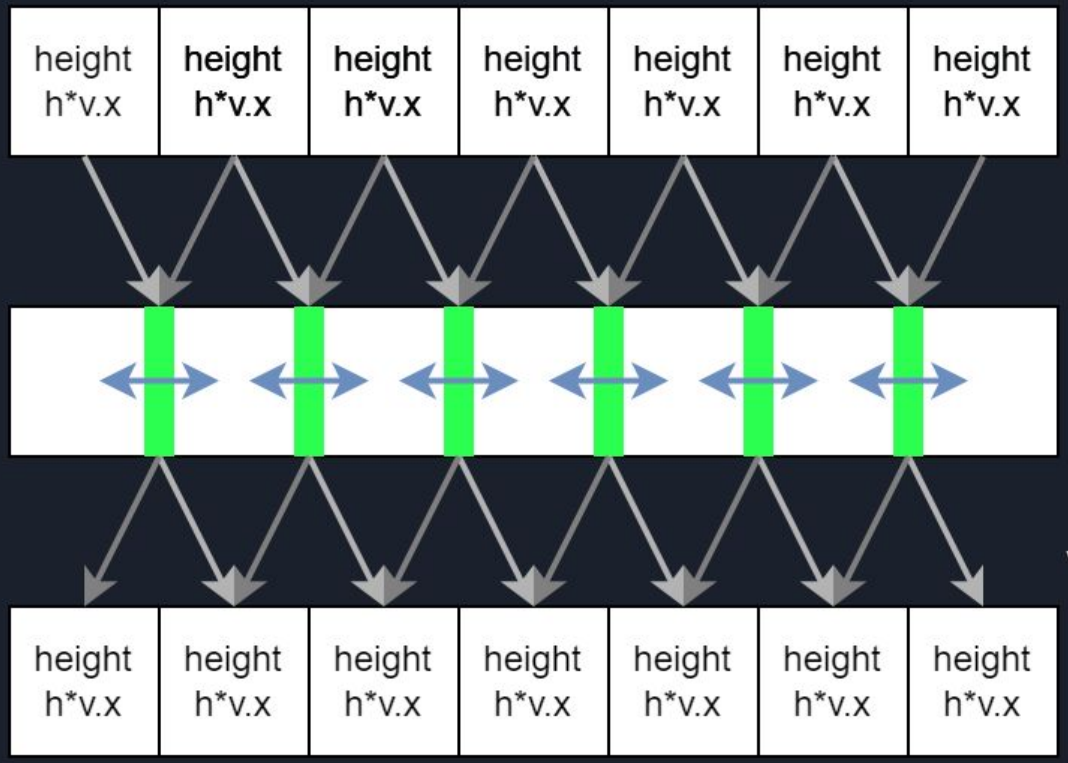


Tsunami-Lab Module

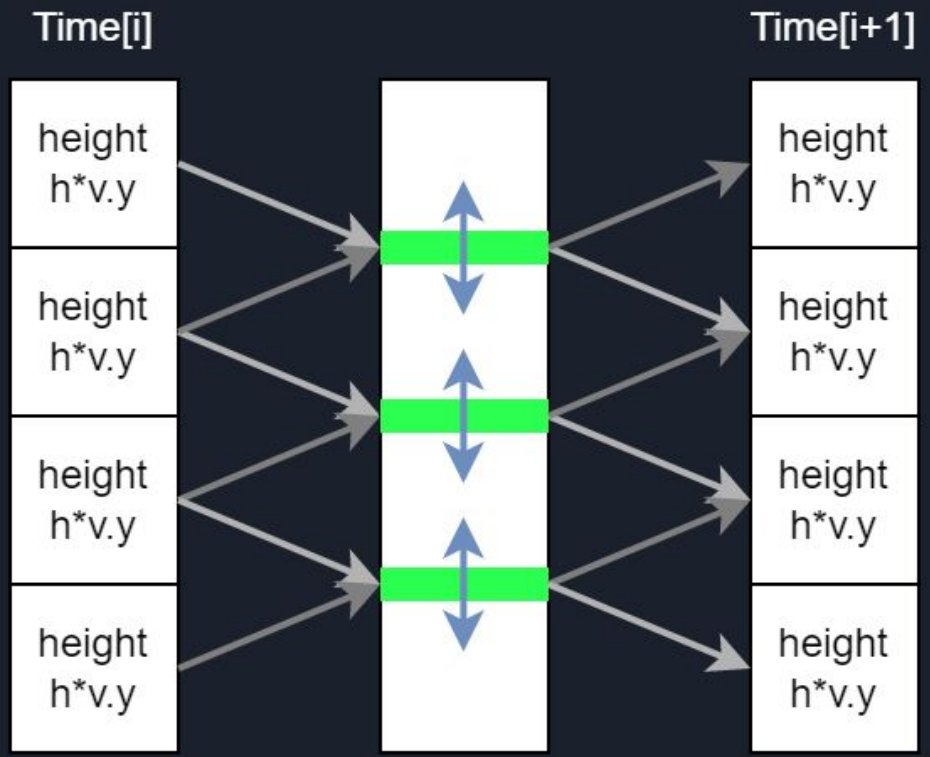
- Gained overview over solvers
- Shallow water equations
- Height, Momentum, Bathymetry - Model
- Discretization in 1d & 2d
- Outflow Boundary Conditions
- IO with NetCDF library
- Tsunami simulations: Tohoku 2011, Chile 2010
- Checkpointing, coarse output
- Optimizations & Parallelization



Solver in 1d


height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
compute flow of
height / momentum
apply
value += flow * dt / dx
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x
height
h*v.x

Solver in 2d (Dimensional Splitting)



Project Plan Overview

1) Execute tsunami simulation on GPUs with OpenGL
2) Test out Graphics & Compute Pipeline
3) Optimize tsunami kernels
4) Implement tsunami simulations in my game engine "Rem's Engine"
5) Visualize these simulations in real-time

Solver Implementations

OpenGL Graphics Pipeline

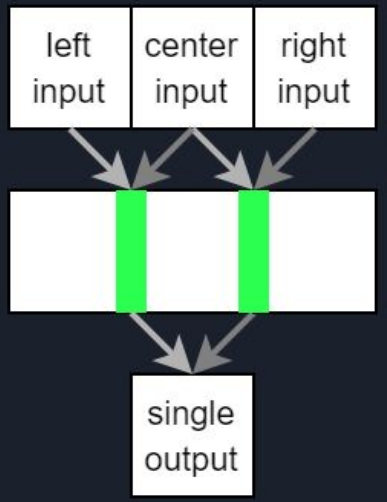
- Primitives: Points, Lines, Triangles, Quads
- Multiple stages: Vertex, Tessellation, Geometry, Fragment shaders
- Each kernel call (fragment shader) writes exactly one pixel (may be on multiple buffers) within primitives
- Complex reading/writing functions: texture filtering, blending, mipmaps, …
- Numerical gradients available (from 2x2 cells)
- Wide hardware support

OpenGL Compute Pipeline

- Single stage: Compute
- Flexible writes: at any position, multiple times per kernel
- Atomic Operations, but only for integers :/
- Shared Memory within work-group
- Built-in 3d coordinates (like CUDA)

Base GPU Kernel

- Load data from left, center and right cell,
- Compute flow between left/center and center/right
- Apply update on cell
- Writes cell to result buffer

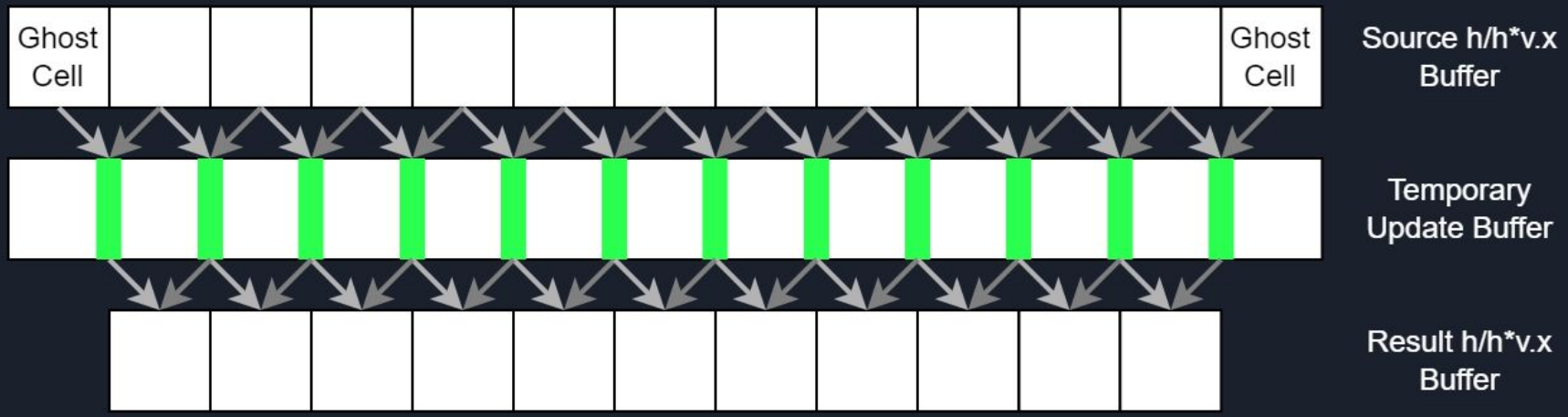


➔ 2x more computations than on CPU

Two Passes Solver

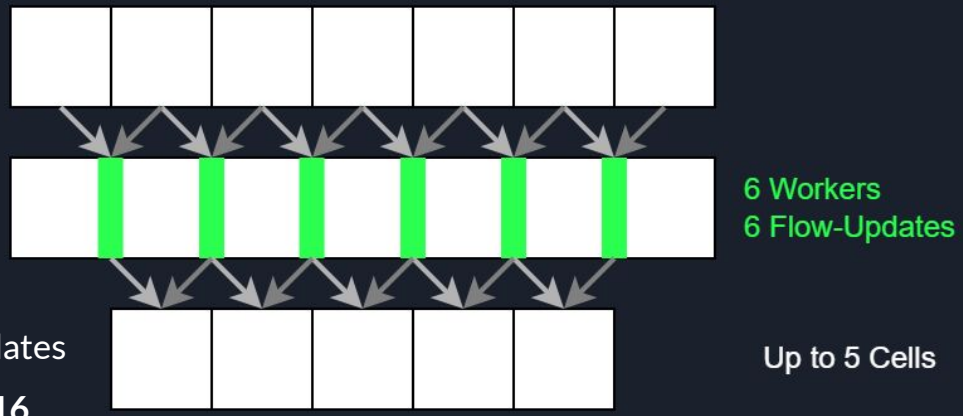
Previously computed result from every edge twice ➔ wasted computations

- first compute the updates, then apply them
- in total 4 passes for every timestep



Shared Memory Solver

- Same idea as two-passes-solver
- Use less memory bandwidth
  by keeping temporary results in caches
- Each group computes **16 x 16** flows - updates
- Update within a group is applied to **15 x 16**
  inner cells



YX Memory Accesses

Spoiler: Compute performance on Tesla P100 bad

➔ maybe memory accesses are bad?

Changing work order, changes strides for memory accesses

```
1 void main(){                                          .yx
2     ivec2 uv1 = ivec2(gl_GlobalInvocationID.xy);
3     if(uv1.x <= maxUV.x && uv1.y <= maxUV.y){
4         ivec2 deltaUV = ivec2(1,0);
```

Computing in FP16

Memory bandwidth probably bottleneck ➔ Reduce bandwidth

Idea: store information in half floating point precision instead of full precision

Two issues:

- Correctness? Errors will be larger

Computing in FP16

Two issues:

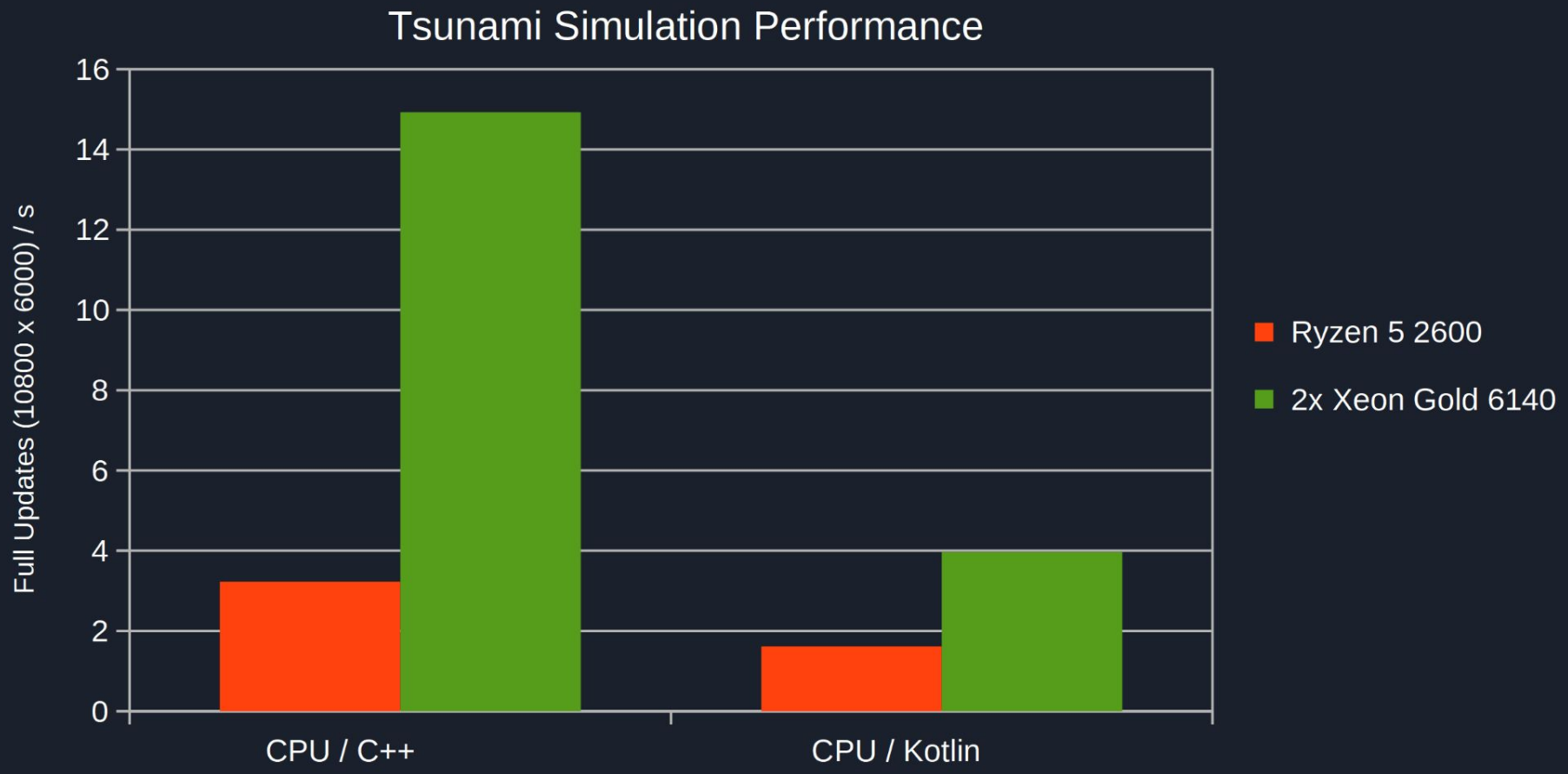
- Correctness? Numerical errors will be larger

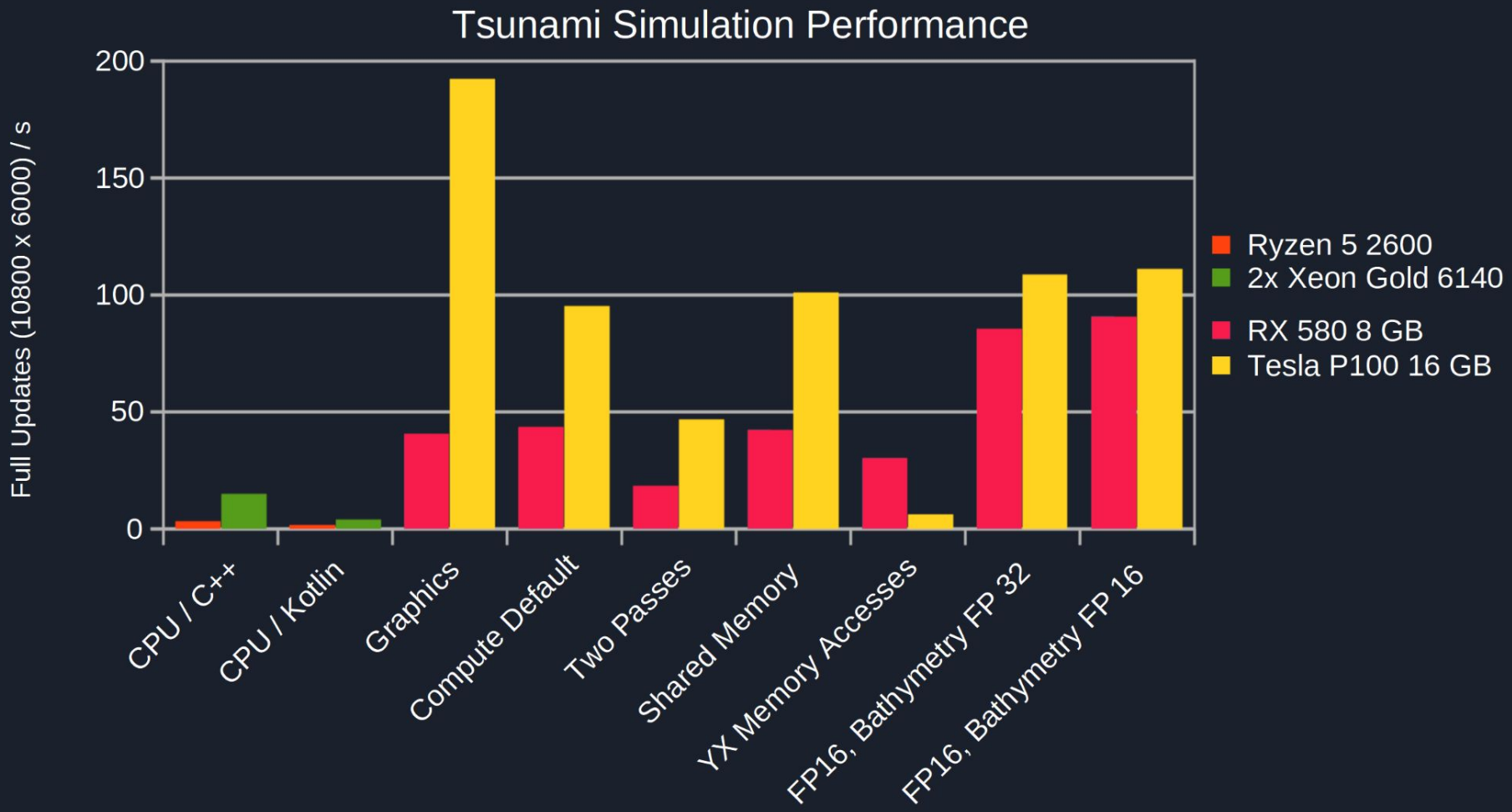
- Water depth 5000m, Surface height +/-5m
  10 bits of mantissa ➔ 1024 relative resolution ➔ barely could store surface height

  Solution: load / store surface instead of height

```
vec3 load(ivec2 uv){
    float surface  = imageLoad(srcSurface, uv).x;
    float momentum = imageLoad(srcMomentum, uv).x;
    float bath     = imageLoad(srcBathymetry, uv).x;
    return vec3(surface - bath, momentum, bath);
}
```

Performance Measurements

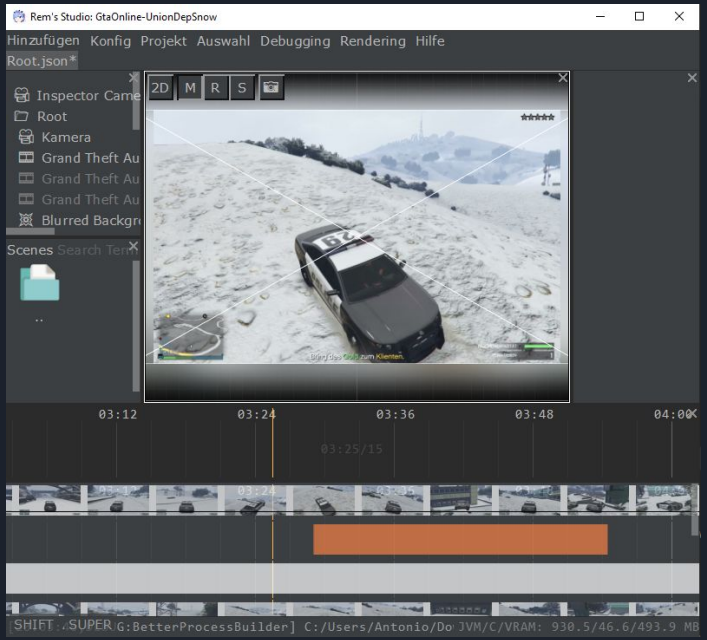

Tsunami Simulation Performance
Full Updates (10800 x 6000) / s
16
14
12
10
8
6
4
2
0
CPU / C++
CPU / Kotlin
Ryzen 5 2600
2x Xeon Gold 6140


Tsunami Simulation Performance
Full Updates (10800 x 6000) / s
200
150
100
50
0
CPU / C++
CPU / Kotlin
Graphics
Compute Default
Two Passes
Shared Memory
YX Memory Accesses
FP16, Bathymetry FP 32
FP16, Bathymetry FP 16
Ryzen 5 2600
2x Xeon Gold 6140
RX 580 8 GB
Tesla P100 16 GB

Integration into Rem’s Engine

Rem’s Engine

- Originally parts of many personal projects,
- Then Rem’s Studio (a video editor)



Rem's Engine

- Originally parts of many personal projects,
- Then Rem's Studio (a video editor),
- Then a **E**ntity-**C**omponent-**S**ystem based game engine*
  like Unity / Unreal Engine / Godot

- Open Source, Kotlin, github.com/AntonioNoack/RemsStudio

**not really production ready, no official release yet*

Creating a mod for Rem's Engine

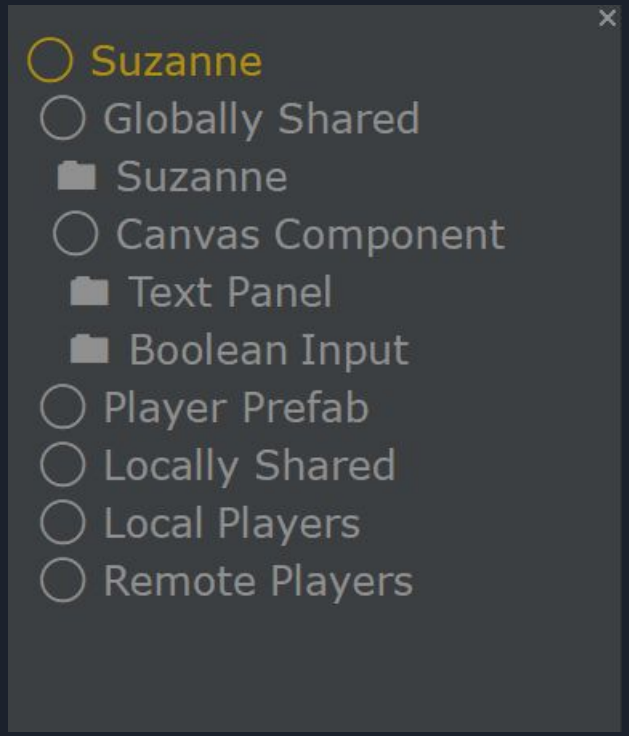
- Similar to Bukkit plugins / Forge mods (Minecraft)
- Create a main class, and let it extend me.anno.extensions.mods.Mod
- Create a file called "extension.info", and list main class, author, version, …
- Override `onPreInit()`/`onInit()`/`onPostInit()`, `onExit()`, and register your component classes using `me.anno.io.ISaveable.registerCustomClass(SampleInstance())`

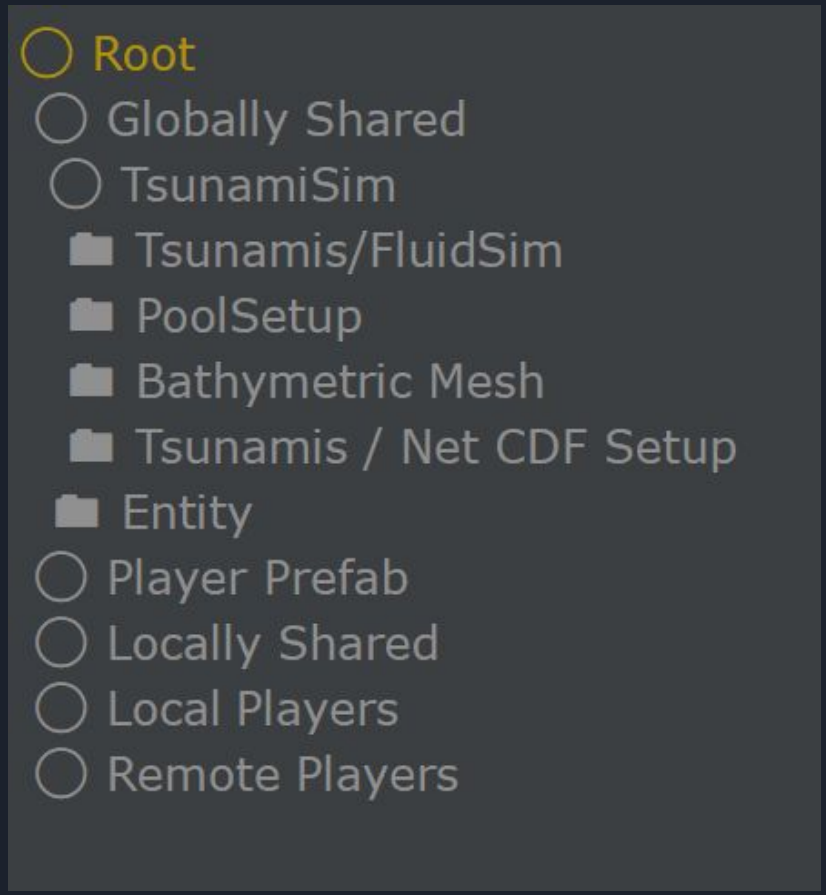
Mods can be tested by calling `me.anno.extensions.ExtensionLoader.loadMainInfo()`, and then creating & running an instance of `me.anno.engine.RemsEngine`

Full tutorial: https://github.com/AntonioNoack/RemsStudio/wiki/Creating-Custom-Extensions

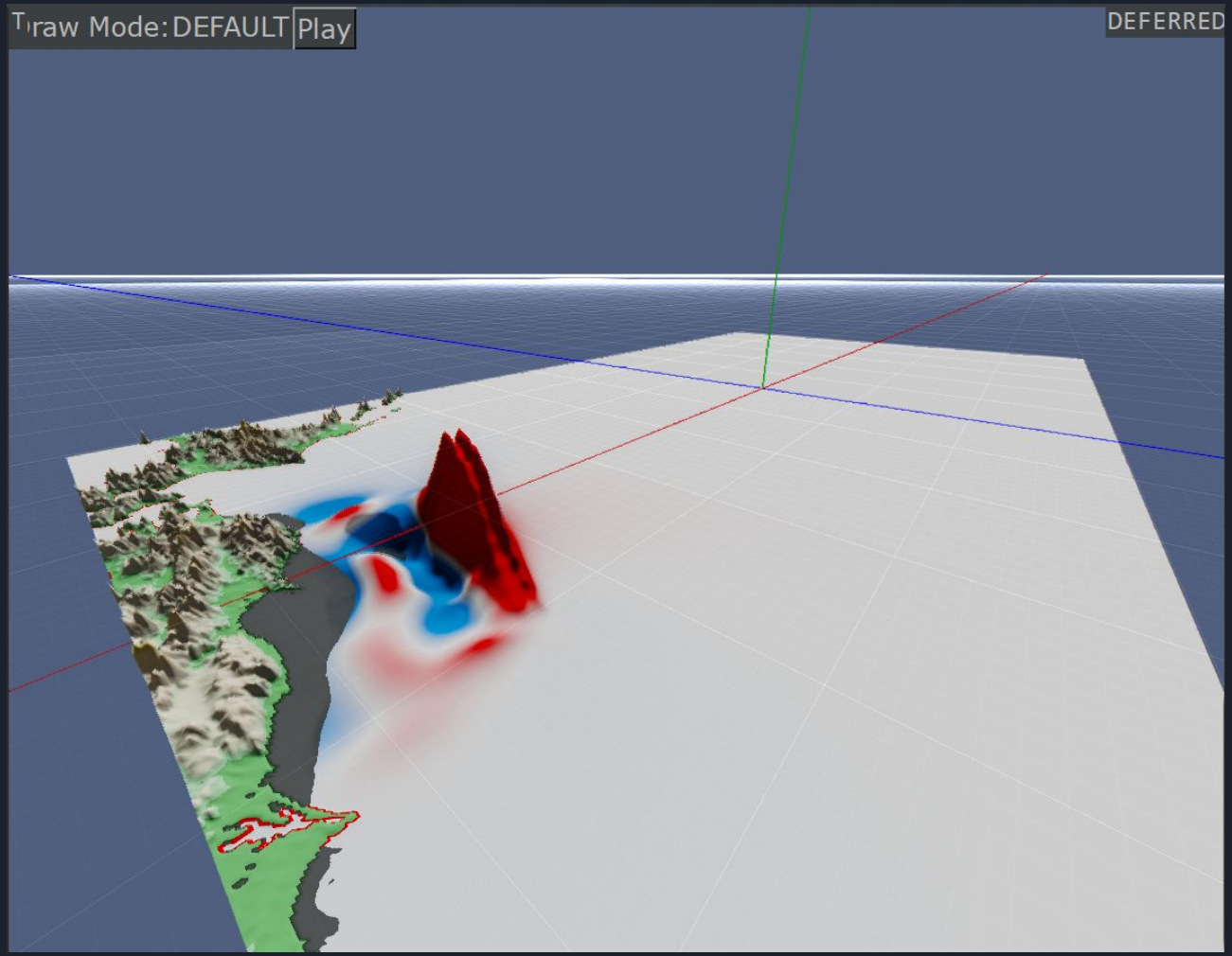
Showcase

Scene Hierarchy

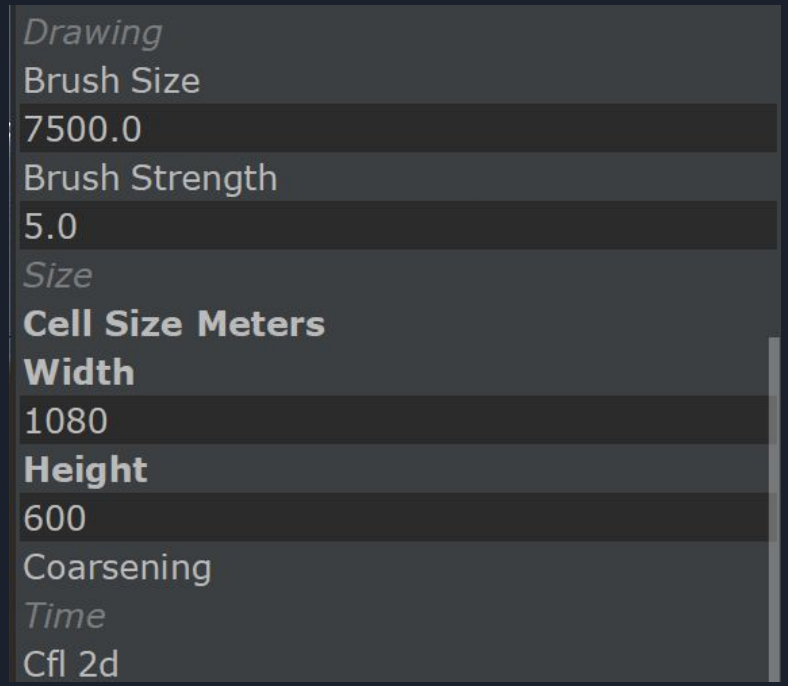


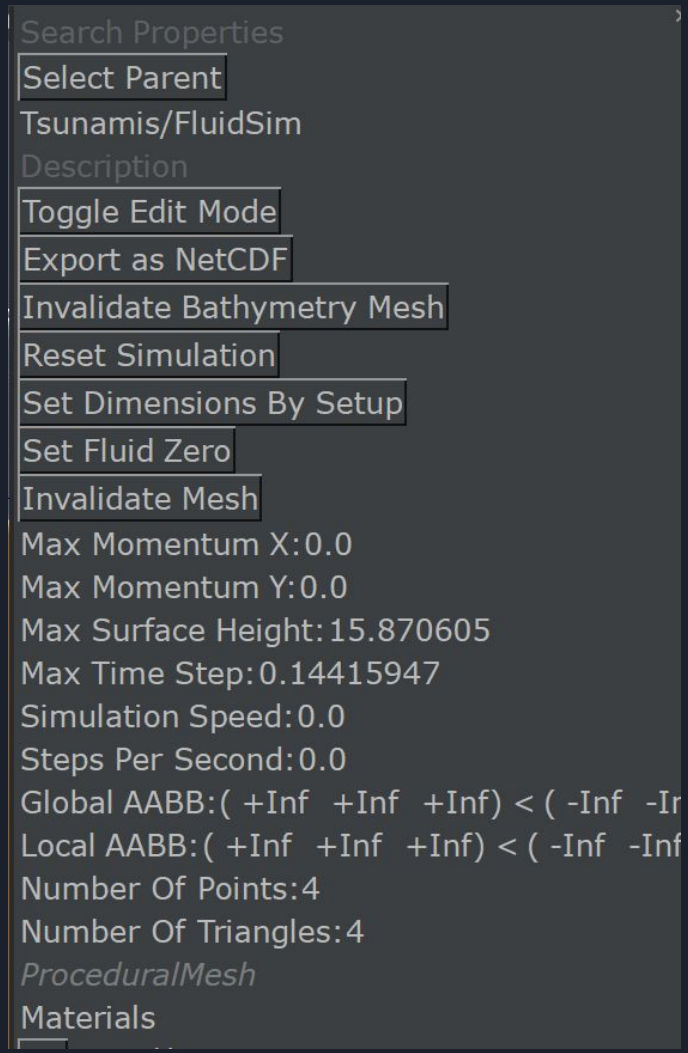


Scene View

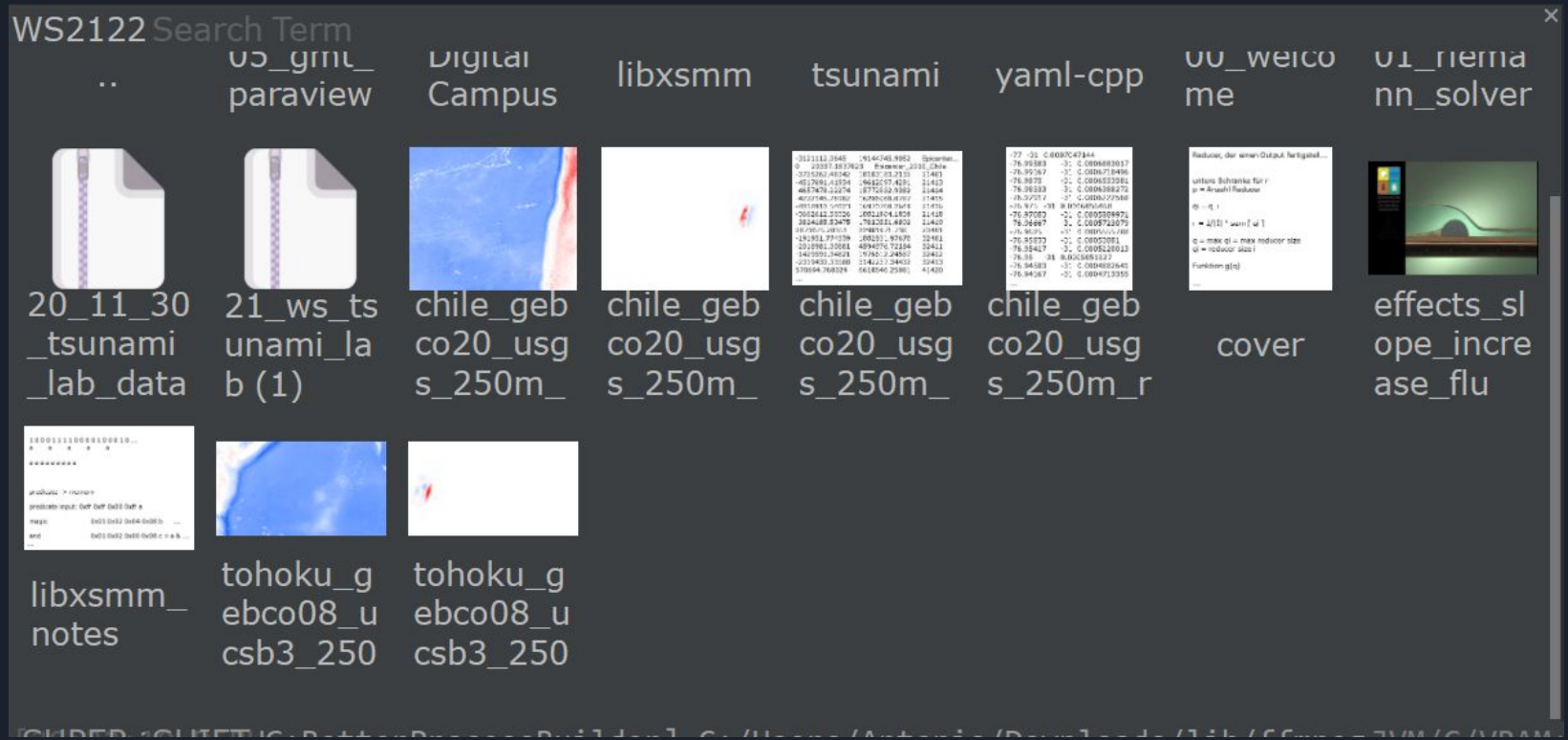


Property Inspector

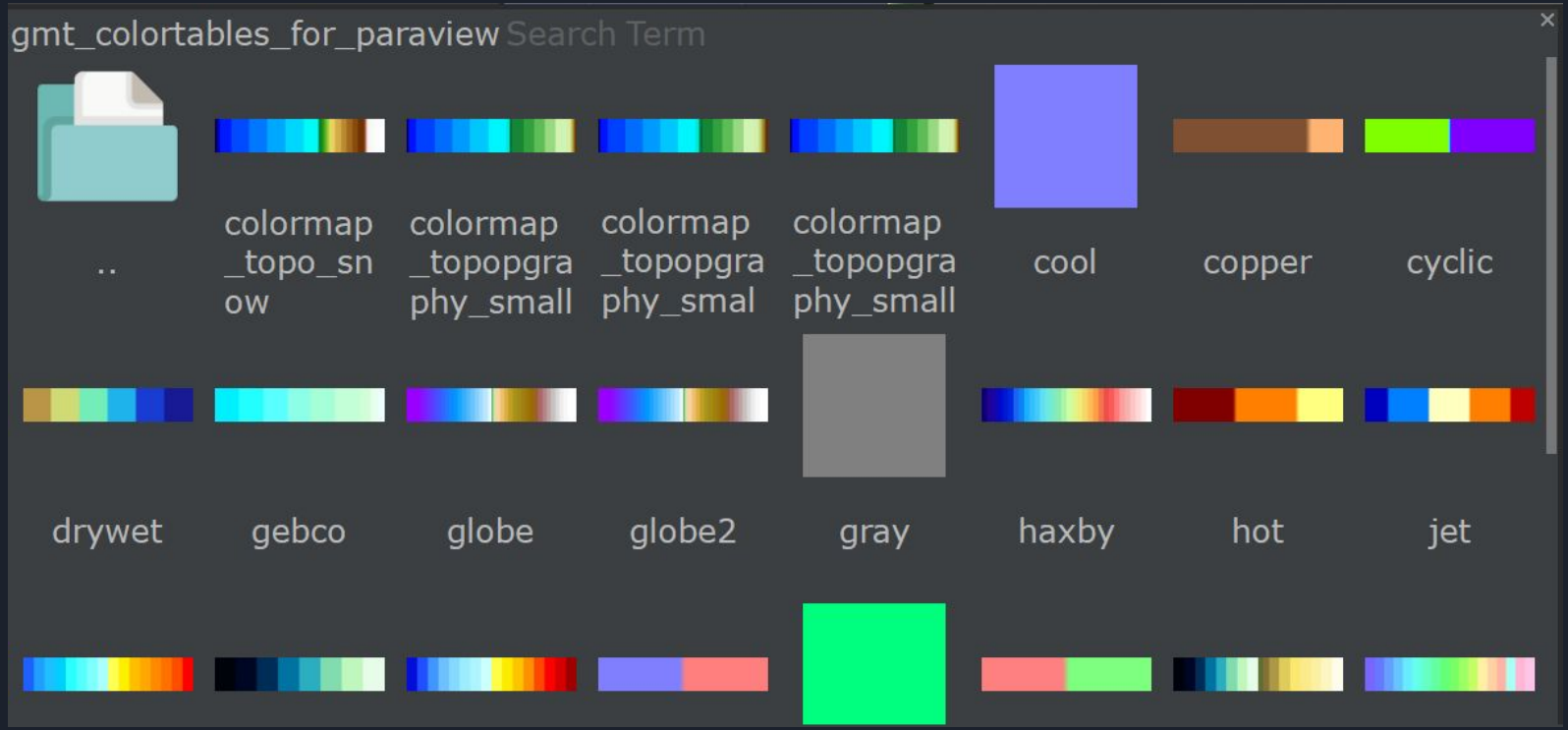




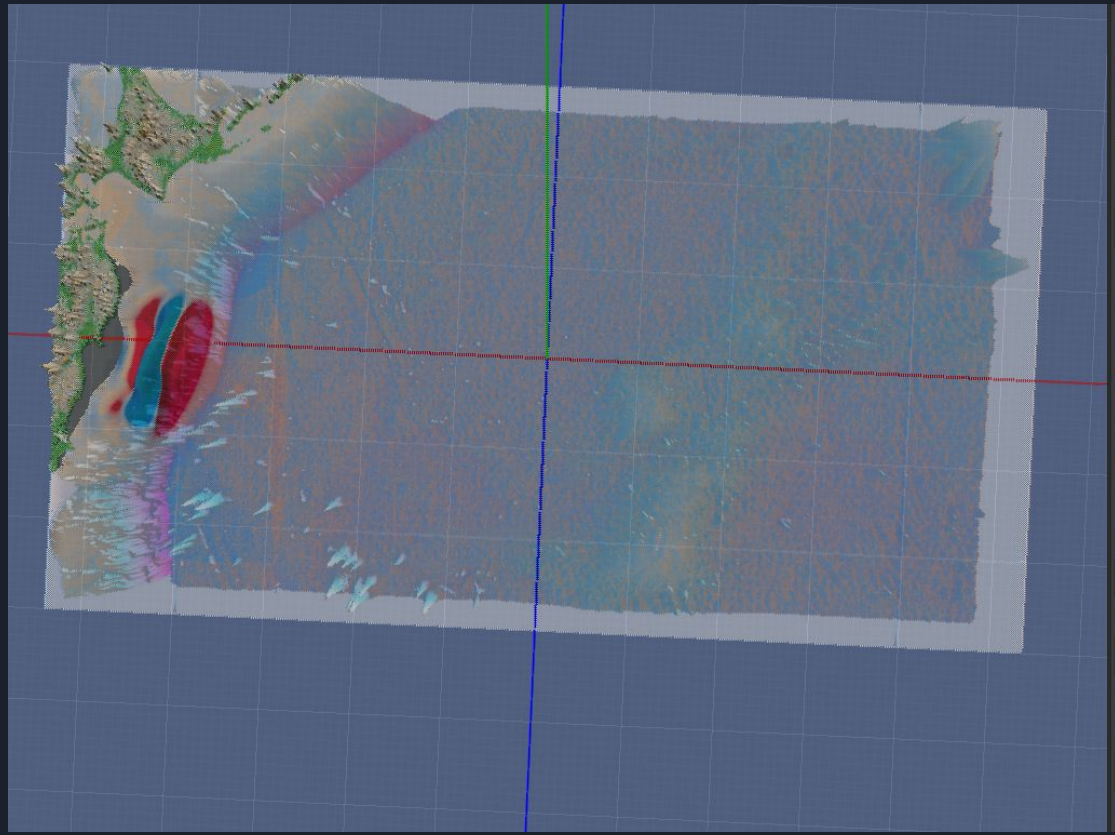
File Explorer



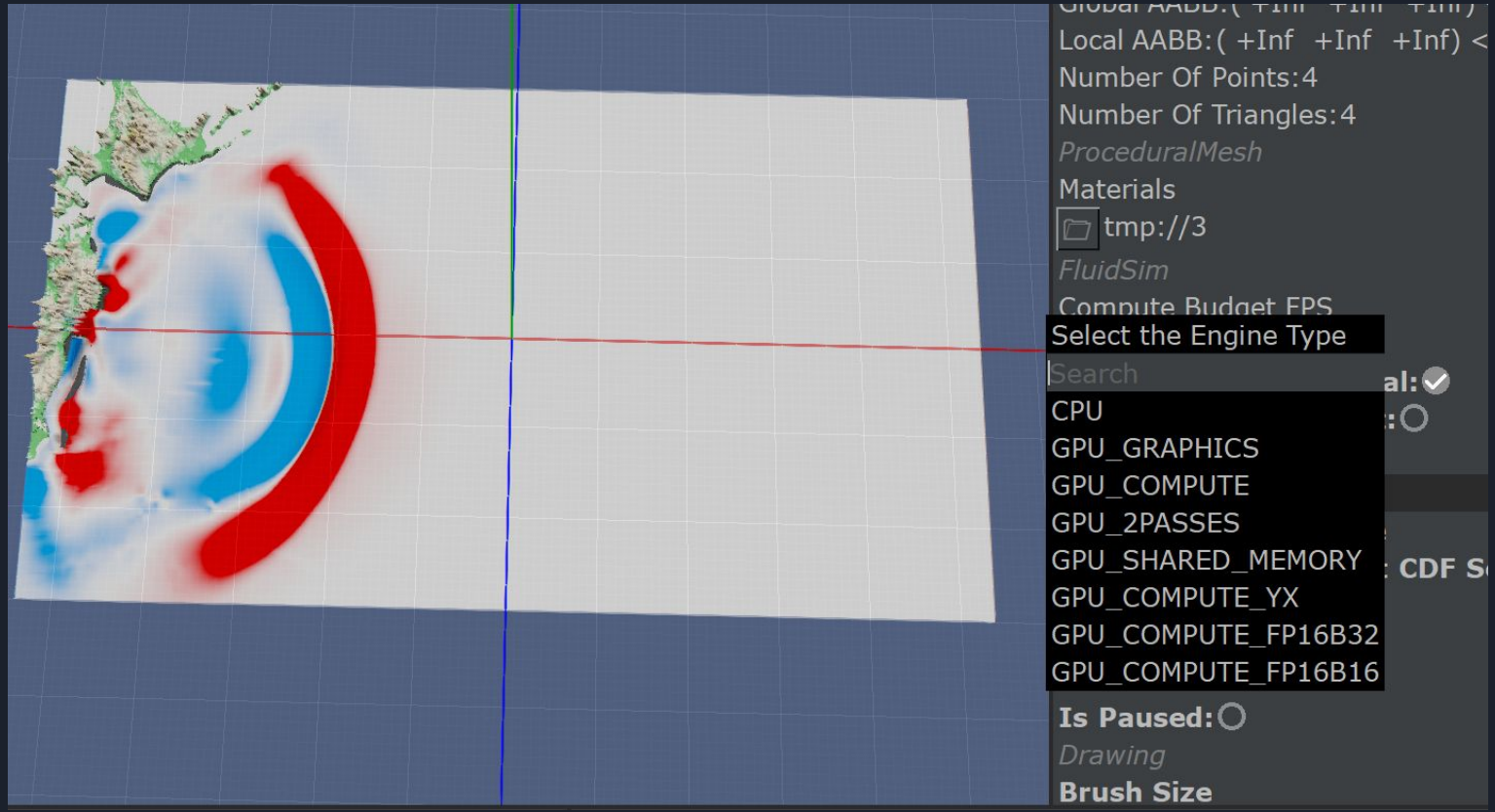
File Explorer



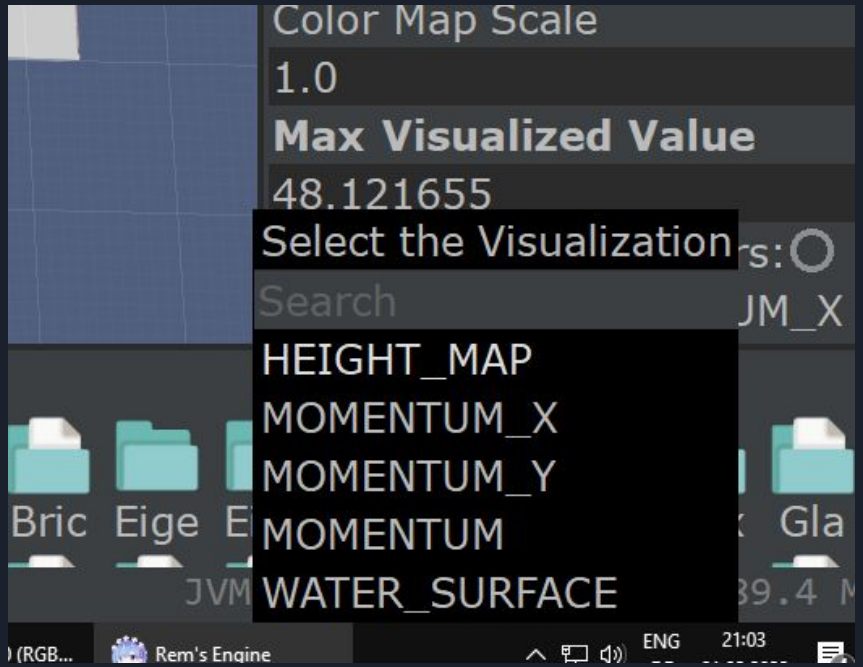
Tsunami Setup (Tohoku)



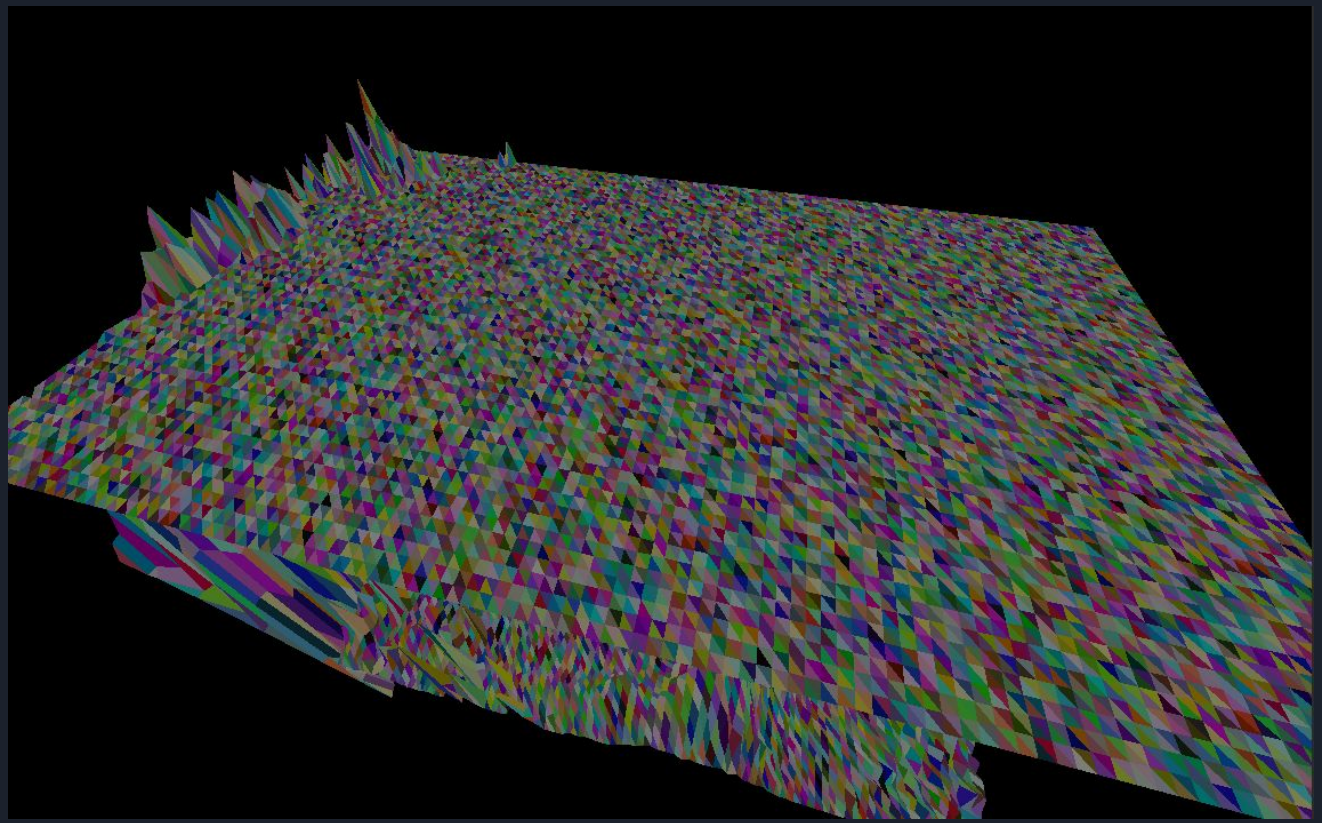
All Engine Types available



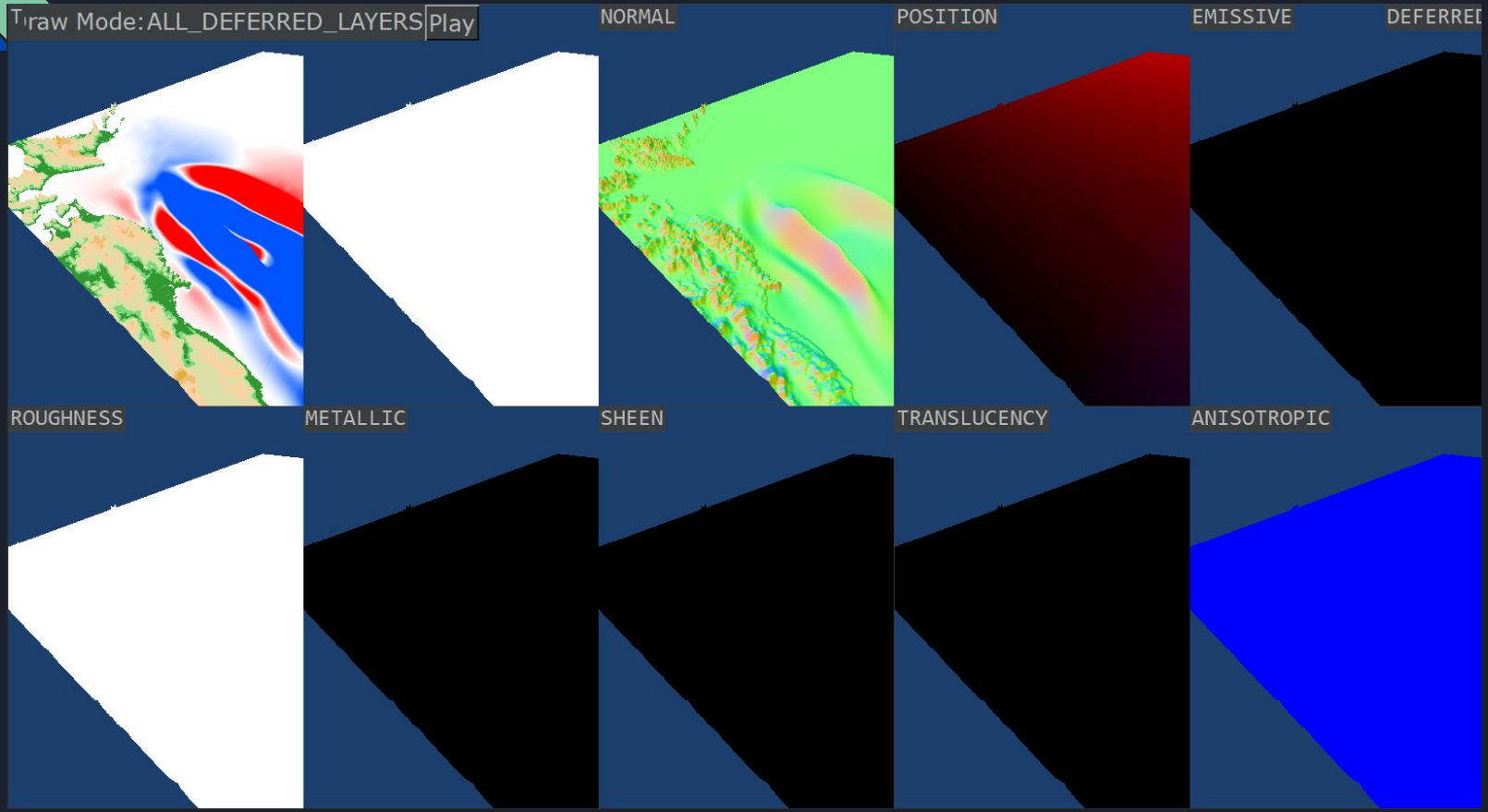
Visualisation Modes



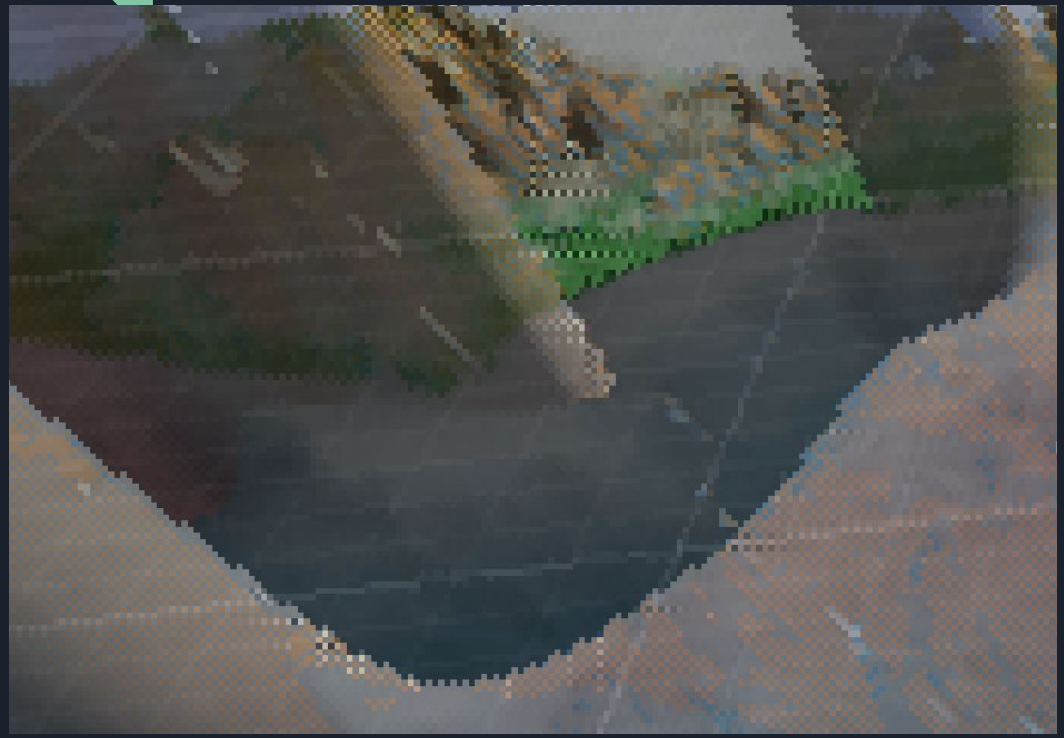
Behind the scenes: Triangle Meshes



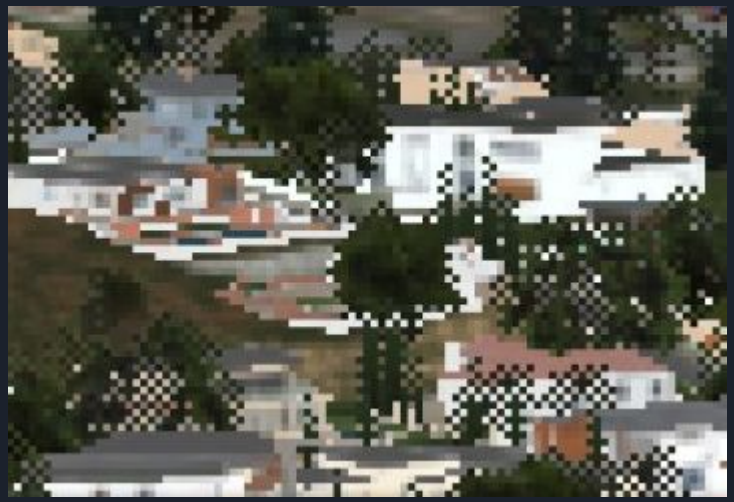
Deferred Renderer (Physically Based Rendering)



Transparency - Checkerboard Rendering

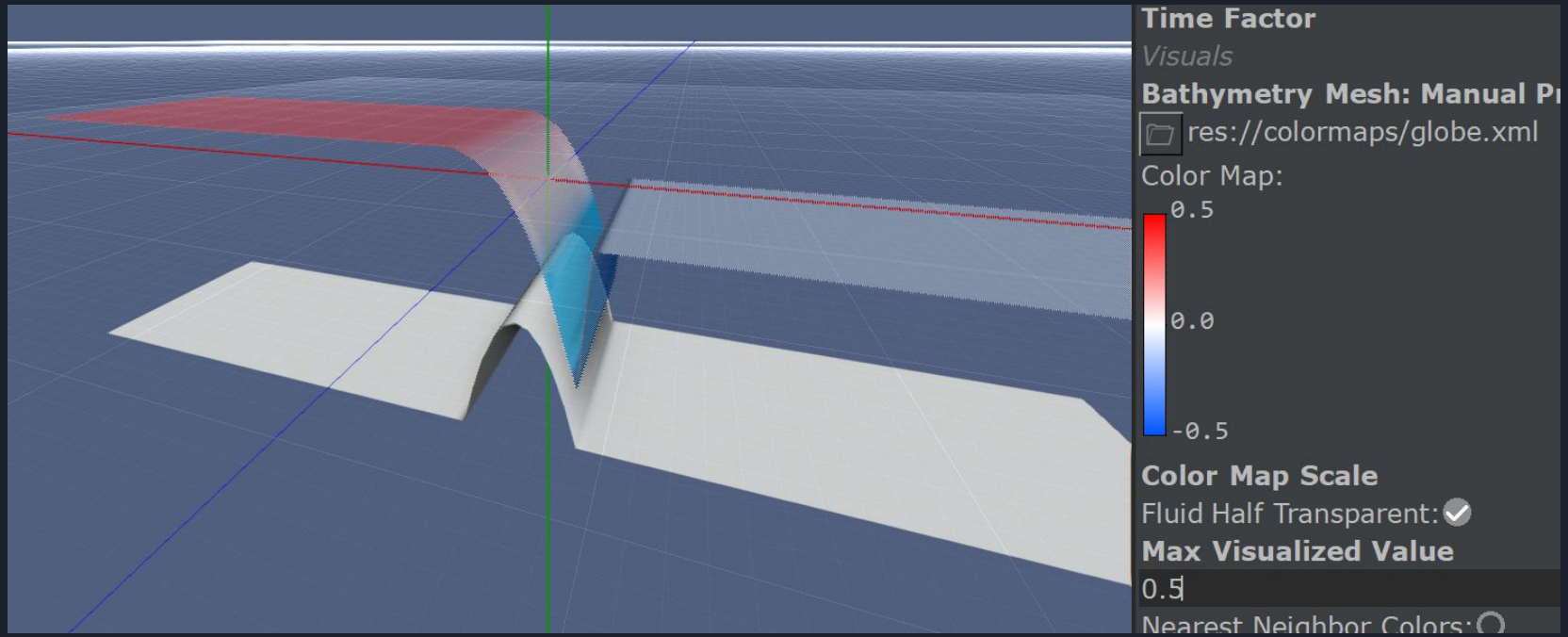


Rem's Tsunamis' Checkerboard Rendering

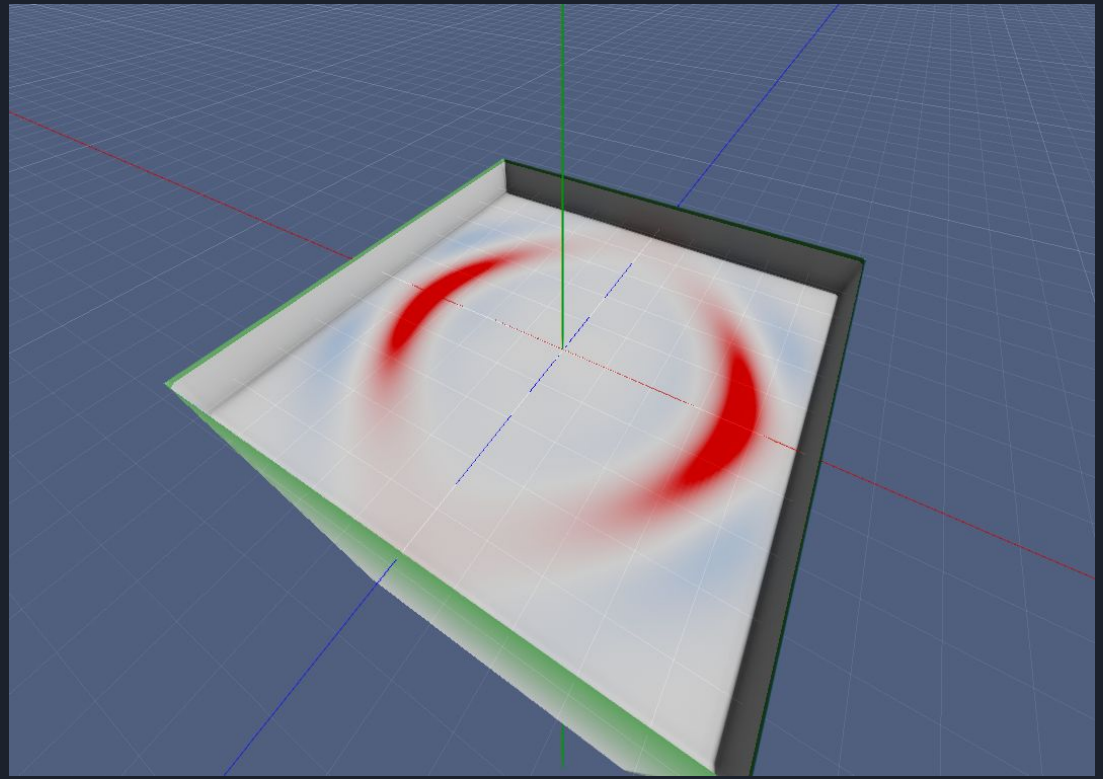


GTA V's Checkerboard Rendering

Critical Flow Setup



Tsunami-in-a-pool Setup



Links

Rem's Tsunamis extension: https://github.com/AntonioNoack/RemsTsunamis

Rem's Engine: https://github.com/AntonioNoack/RemsStudio/blob/master/RemsEngine.md

Rem's Studio: https://github.com/AntonioNoack/RemsStudio, https://remsstudio.phychi.com/

Checkerboard Rendering in GTA V:
https://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/

Thanks for listening!
Questions?