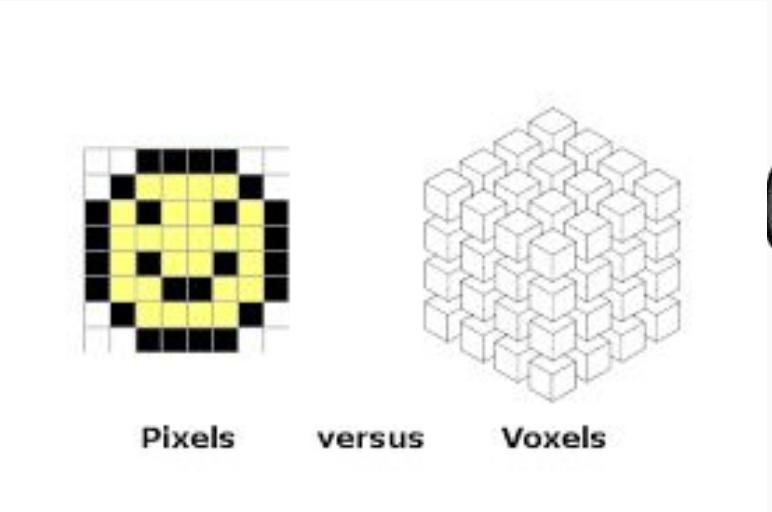
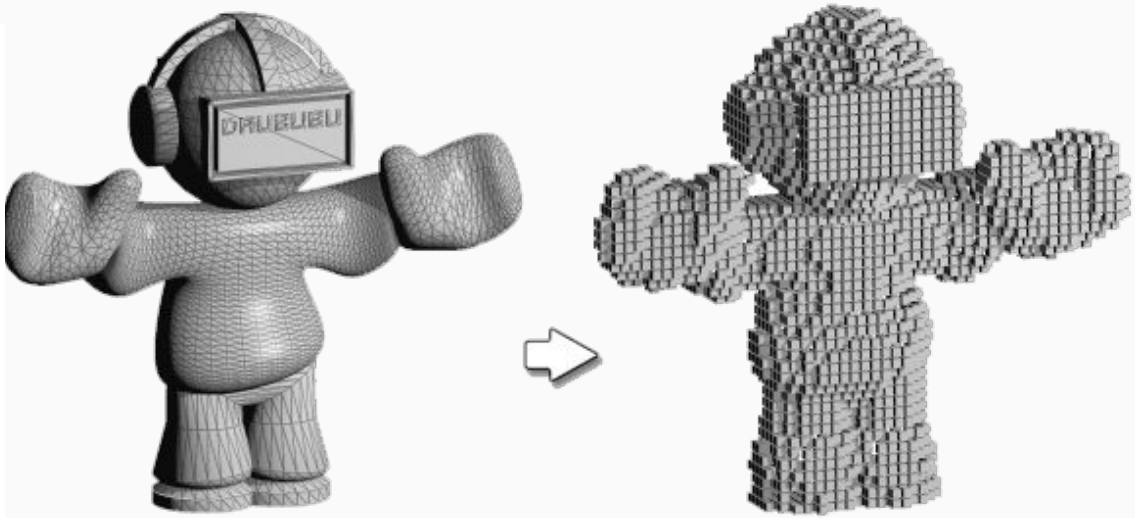
Voxels

Tech Team - Johnny Mercado, Michael Matonis, Glen Giffey, John Jackson

Pixel -> Voxel





Appearance in Games

Comanche: Maximum Overkill - 1992

Minecraft - 2011

Guncraft - 2013

CodeSpell

https://www.youtube.com/watch?v=NN5mQxX-Zd0

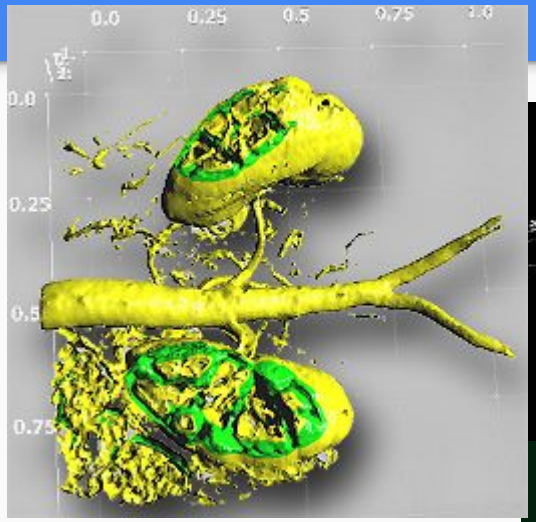
Voxel Applications

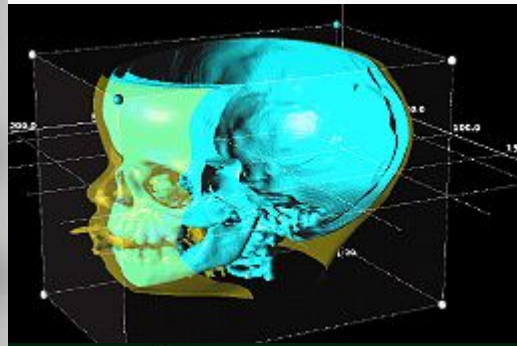
Animation - https://vimeo.com/11832919

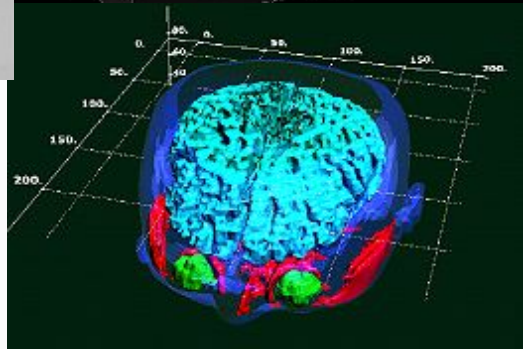
Data Visualization

CGI

Games



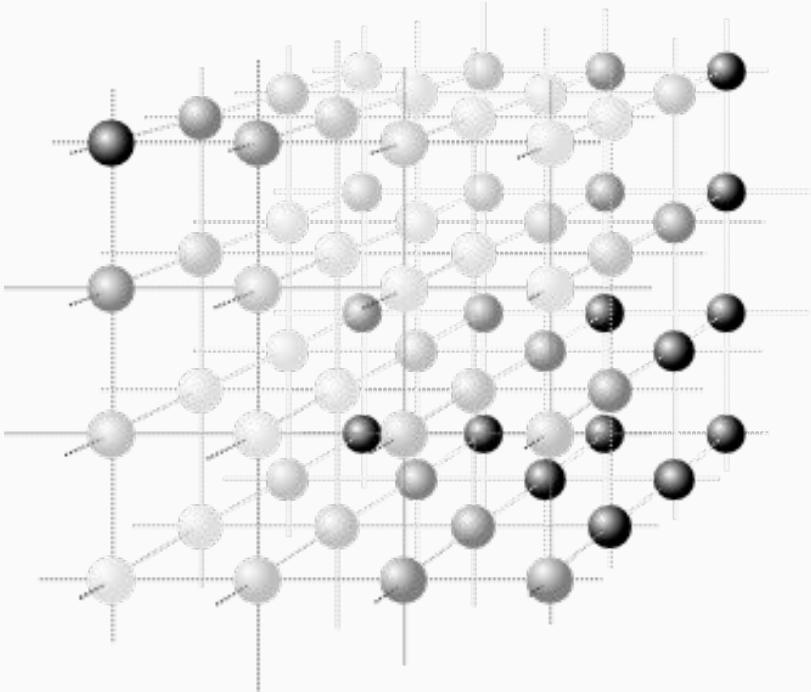




Voxel Structure

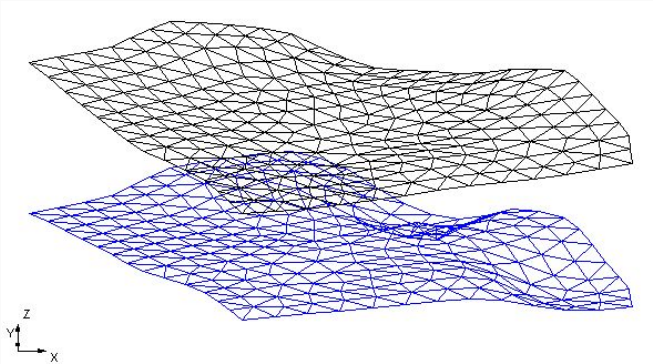
- A voxel represents only a single point, not a volume
- Individual voxels contain position, material, and possibly texture data

- direct volume rendering
- extract polygon iso-surfaces (i.e. Marching Cubes)



Some Graphics Background

- In order to generate a nice 3D shape, we create a “mesh”
- A “mesh” is made up of triangles
- Each triangle has a set of the following:
  - 3 Vertices
  - Normals used for lighting
  - “UV” coordinates used to apply texturing

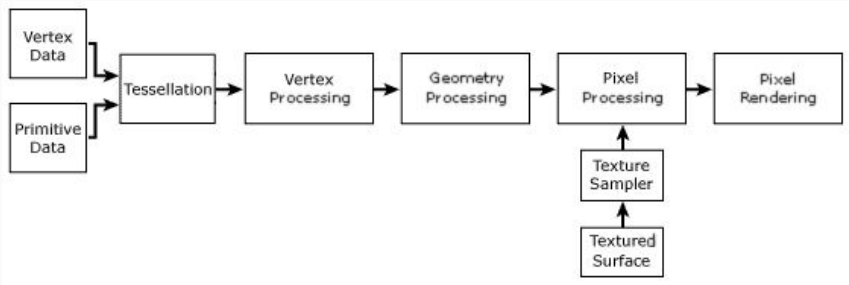


Some Graphics Background (Cntd.)

- Each mesh will have some large number of vertices/triangles
- Unity will render the meshes using your computer's Graphics Processing Unit (GPU)
- Due to the way graphics work, the world must be divided into a set of meshes and each mesh must be rendered

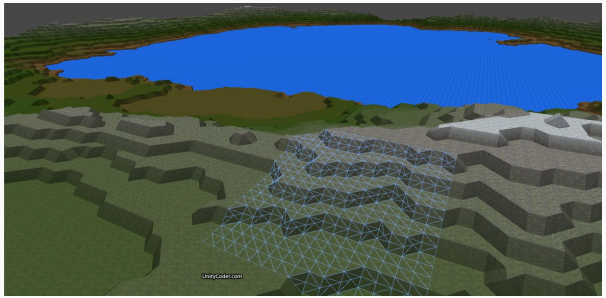
What is a GPU?

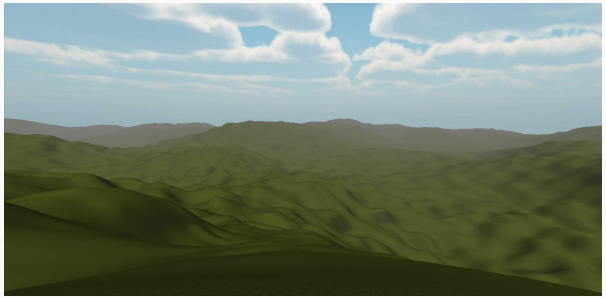
- In order to process potentially millions of vertices, you need A LOT of processing power.
- CPU improvements have been reducing over time.
- Solution: Have a large number of cheaper/weaker chips work in parallel on the process
- GPUs process all the vertices in parallel



Generating Voxel Terrains

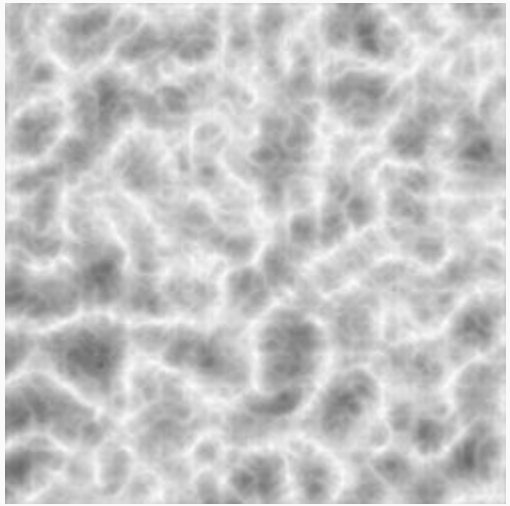
- Need an algorithm to generate the voxel “cubes” themselves
- Need an algorithm to create a mesh of triangles from this data
- Many possible algorithms
- We need to keep in mind the computational complexity of covering every voxel
    - Higher resolution means a better picture quality, but far more computation





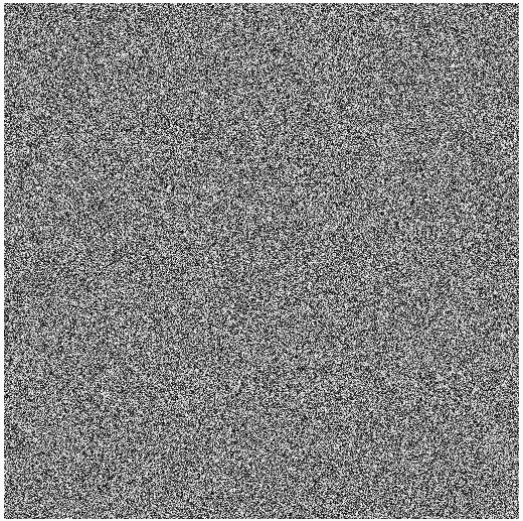
Pseudo-Random Noise

- Noise functions in general are a way of getting “psuedo”-random noise.
- It appears random from the users view, but is actually more like a pre-prepared complex math function
- There are many different types of noise used in software and each has a particular purpose.



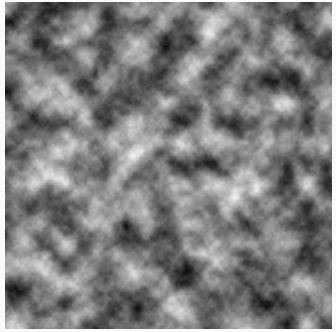
rand(int seed)

- Many people are familiar with basic random noise functions from various
- If you displayed this as a picture it would like TV static.
- This could be generated by something like the following:
  - newSeed = (a*seed + c) % m;
  - Where a, c and m are just arbitrary large numbers



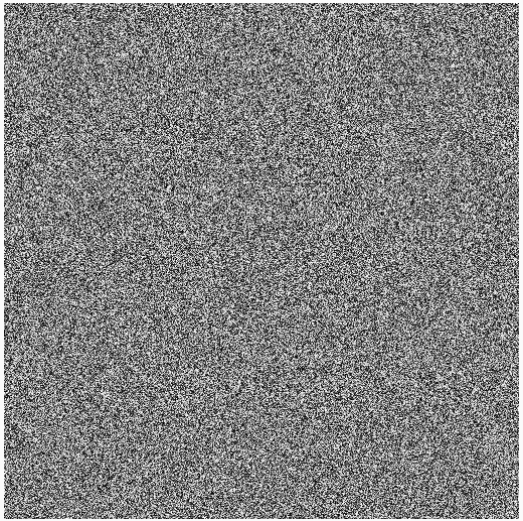
Decent Noise Compile Time

- Voxels can be generated off of a noise function to create decent terrain.
- Without a decent function, we might generate something like this (see right)
- This could be pre-generated in photoshop and loaded into our game’s voxel engine quite easily



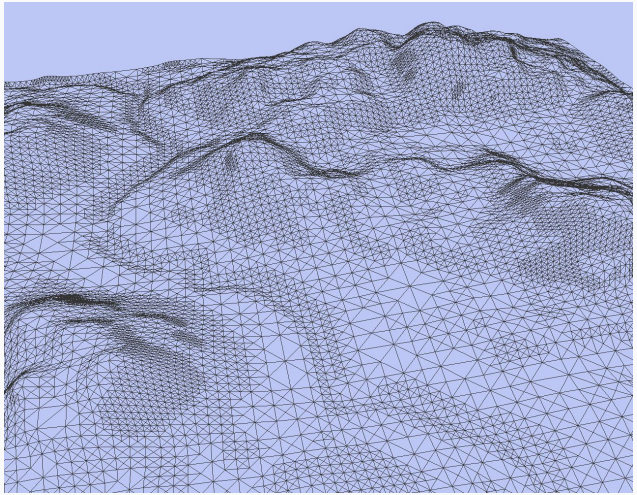
Smooth Noise

- Without going into too much detail, we want a function f(x, y, z, seed) that can give us “smooth” noise.
- Instead of entirely random, we want the values to be change slowly across each axis.
- Popular type to look into:
  - Perlin Noise/Simplex Noise
- Complicated math behind functions



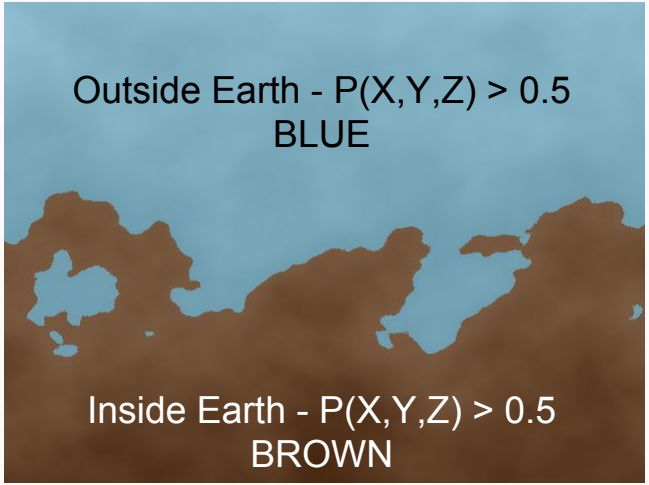
Creating a Mesh From Data

- Once we have our functions, we need a way to make our terrain from this data.
- Assumption: We have a noise function that generates smooth 3D floating points [0.0, 1.0]
- We need to filter this data to make a mesh that will act as our terrain!
- Our Method of Choice: “Marching Cubes” for simplicity/speed compromise



A Simple Overview of Marching Cubes

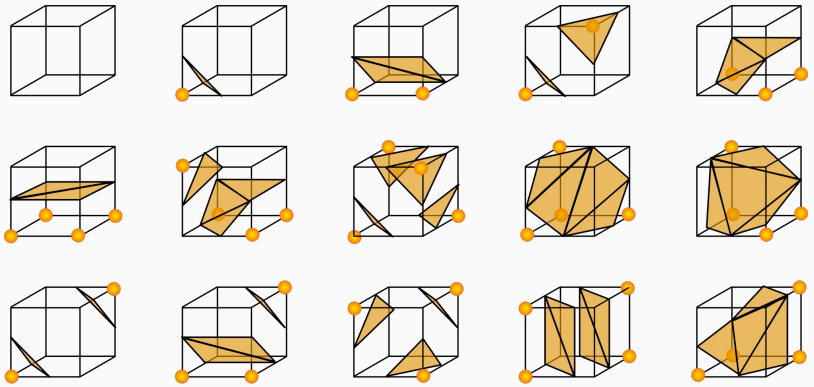
- If you try and picture the earth as a mesh inside a sphere, you might be able to see two volumes.
  - Everything ***inside*** the earth (the terrain)
  - Everything ***outside*** the earth (the sky)
- What if we look at our noise that way?
- We have [0.0,1.0].
  - Every p(x,y,z) > 0.5 is air (outside earth)
  - Every p(x,y,z) <= 0.5 is terrain (inside earth)



A slice of (X,Y) for our world.

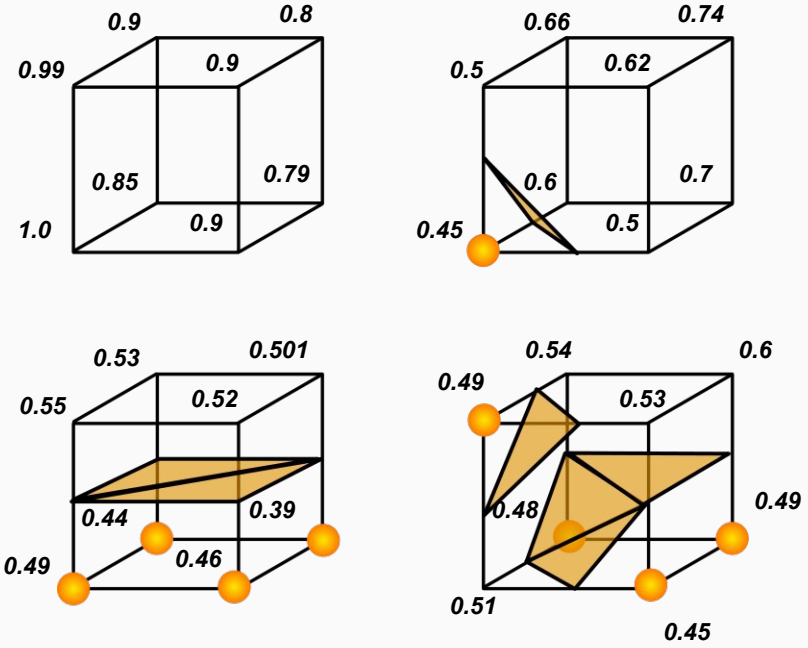
A Simple Overview of Marching Cubes

- Once we understand this concept, Marching Cubes just consists of 15 cases that could occur. A look up table!
- Steps:
  - For ***every*** “voxel” (set of 8 vertices) do:
    - Check which vertices are inside terrain
    - Look up case in table
    - Add new set of triangles to mesh
  - Output mesh to Unity!
  - Calculate normals, UVs, and other vertex data



Example:

- Consider the different examples shown to the right.
  - Look at each CORNERS of the cube
  - If any of the values at the corners are BELOW 0.5, mark them as ground.
  - Now, look at the EDGES of the cube
  - If the edge goes from a ground node to a sky node (unmarked), place a vertex on that edge
  - Combine the triangles.
- There are one or two arbitrary cases...

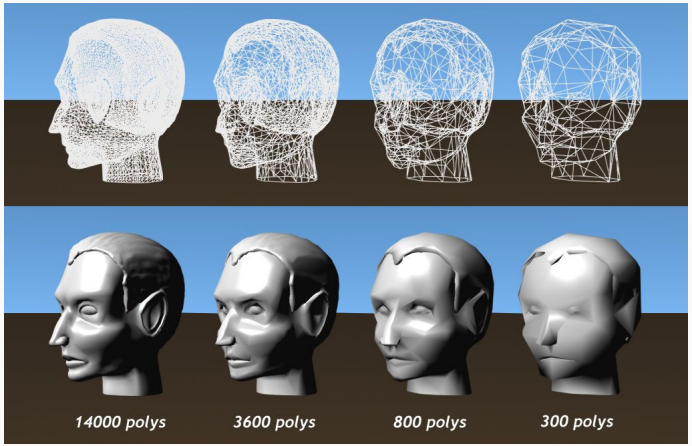


Are We Done?

- The issue is that the largest mesh in Unity can only be ~65000 vertices in size.
- If we have millions of vertices in our generated terrain, how do display it?
- We can separate the world into smaller sectors called “chunks”
- Each chunk is an extremely large mesh that ***together*** make up the environment

Other Technical Considerations

- There are more details to implementing a clean, efficient and realistic looking voxel engine. Those interested should look into the following:
  - Linear Interpolation of Marching Cubes
  - Triplanar Texturing/Shaders
  - LOD System
  - Occlusion Culling
  - GPU Computation



How Do We Store an “Infinite” World?

- Games like minecraft have large scale worlds that can’t be entirely explored without cheats
- With finite storage space how can we store an infinite world?
- Given the same seed, we can generate the same chunk
- Delete chunks far from the player and restore when they come back

How Do We Store an “Infinite” World?

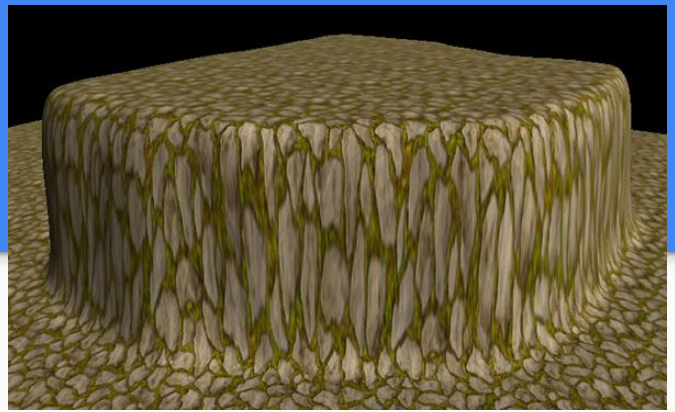
- Storage only requires:
  - The seed to generate
  - Player positioning
  - Player modifications to the terrain
- The world can be so large that the player could never truly explore it all
- Cheats to increase character speed or modify the terrain quickly could overload the hard drive

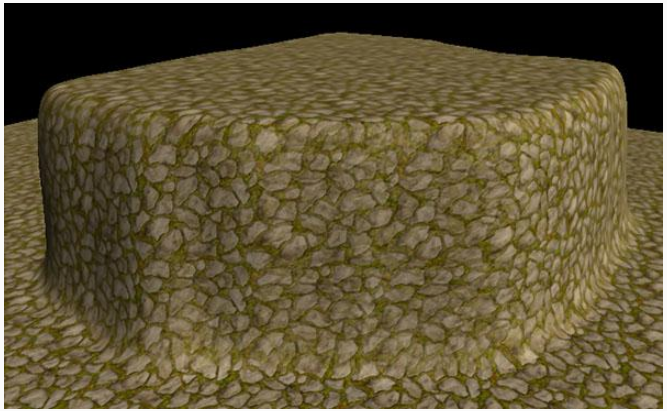
Terrain Generation Techniques

- Heightmaps
  - Relatively low memory usage
  - Fast Implementation
  - Efficient physics
- Voxels
  - Computationally slow
  - Expensive Algorithms
  - Slows with higher LOD

Computational Expenses

- Perlin Noise
  - Computing initial data points in 3D space
- Marching Cubes
  - Creating new vertices
- Mesh Creation
  - Mesh size limitations
- Triplanar Texturing (3 texture lookups)



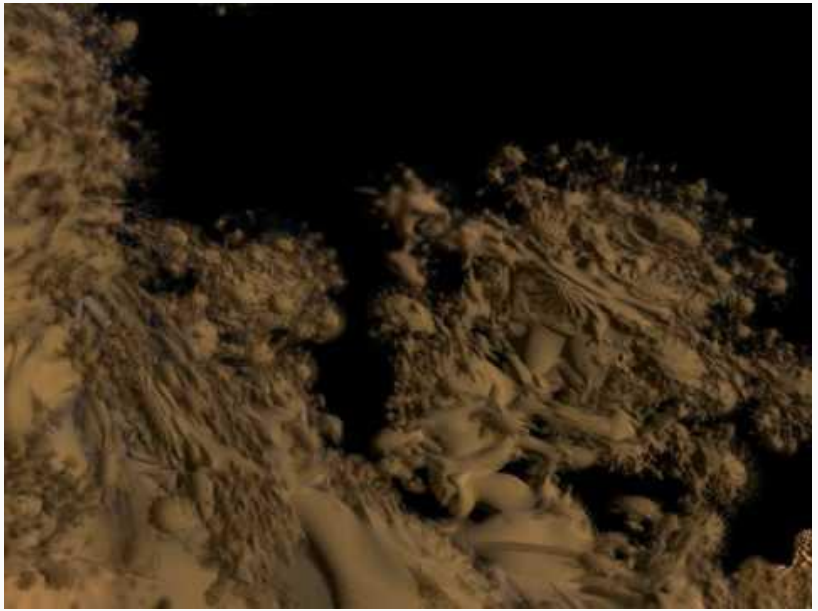


Scalability

- Balance Size, Detail, and Performance for game
- Less mesh = less computation
- Resolution (perlin noise density)
    - Marching cubes with massive number of vertices
- Lower Level of Detail

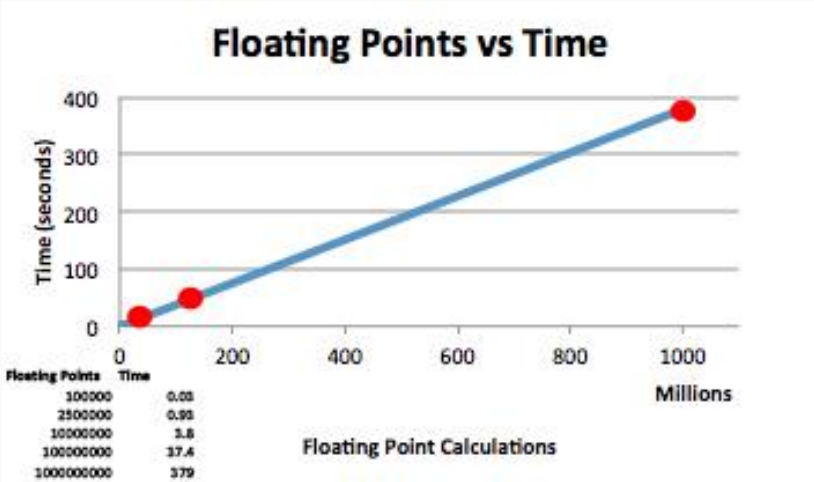
Level Of Detail (LOD)

- Multiple pre-constructed meshes
- Switch out meshes as distance changes
- Lower detail takes less time to render



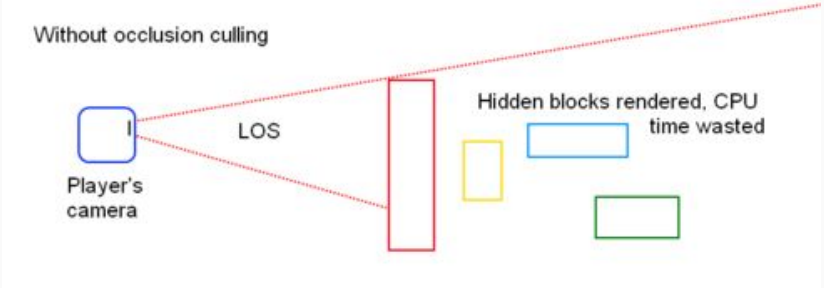
Modeled Expenses

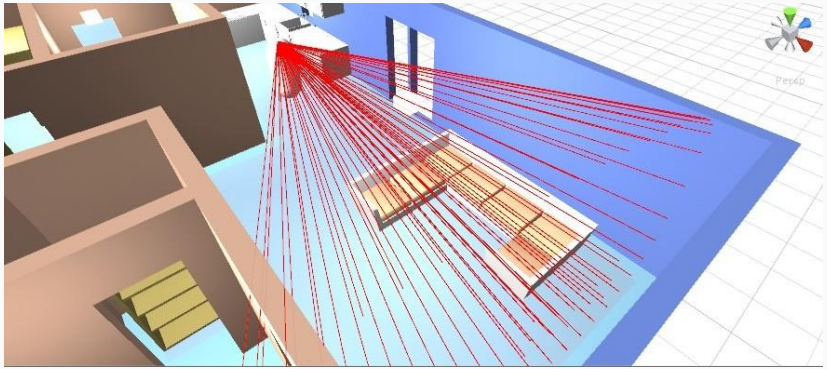
- Unity terrain where player can’t easily see end of world
  - 2048*2048*2048 = 8.5 billion
- 1 billion takes 6.32 minutes
- Multiple Levels of Detail
- Threading not considered



Computational Improvements

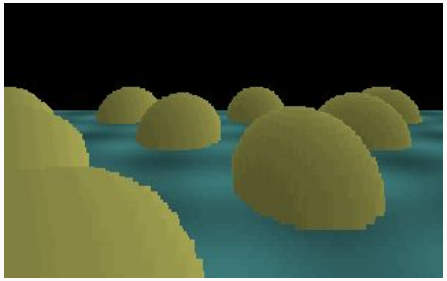
- Occlusion Culling
  - Unity has it for pre-constructed meshes (Umbra)
  - Unreal Engine has support built in
- GPU utilization
  - Divide up computations into thousands of small portions
- Threading
  - Math computations





The Restrictions of Yesteryear

- The size of the information and complexity of the processing of Voxels prevented global use
  - However, advancements in parallel processing & availability of GPU's helped to bring Voxels into more widespread use
  - Voxlap, an early voxel engine made in 2000, supported about 268 million voxels in a scene (w/ some lag reported)
  - Modern simulators take advantage of billions of voxels per scene and able to still hold up in real-time tests



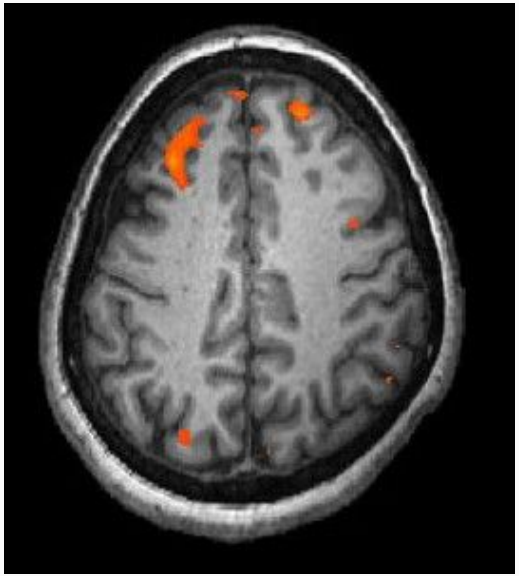
The Voxel Engines of Tomorrow

- The Atomontage Engine, which is currently under development, simulates the world with atom-based representation through the use of voxels
- Features elements such as destructible environments & objects, fluid simulation, and soft voxel bodies
- Fluidity in the nature of the voxels(a voxel can go from simulating a rigid structure to a soft body in real-time)
- Artificial Intelligence at the core overseeing the optimal rendering performance is being achieved at every tick
- Voxelizer able to convert most polygonal models into voxels

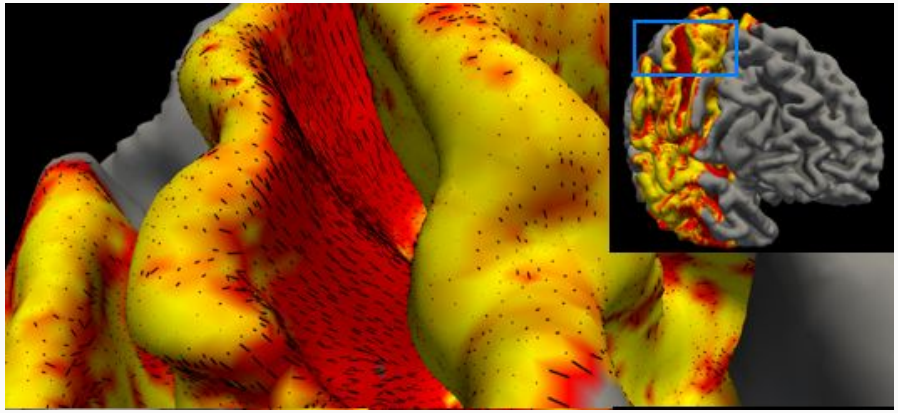
Atomontage

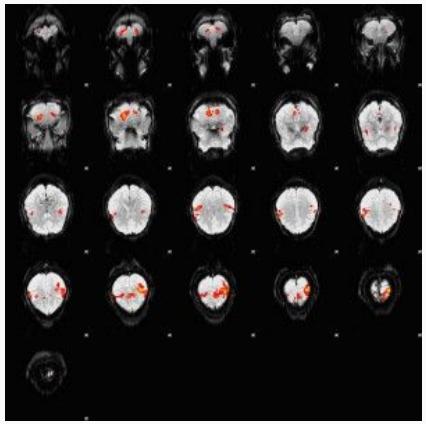
Applications in Medicine & Modeling

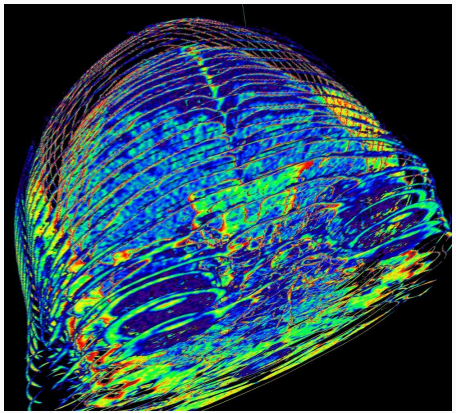
- Voxels use is vastly wide-spread in modelling and is expected to continue to grow in the field of medicine
- Brain Scans
  - Voxels are currently being used in several scanning technologies such as fMRI, MRI, and PET.
  - However, issues exist with the use of voxels in scanning such as the resolution of changes are equivalent to the size of the voxel being used

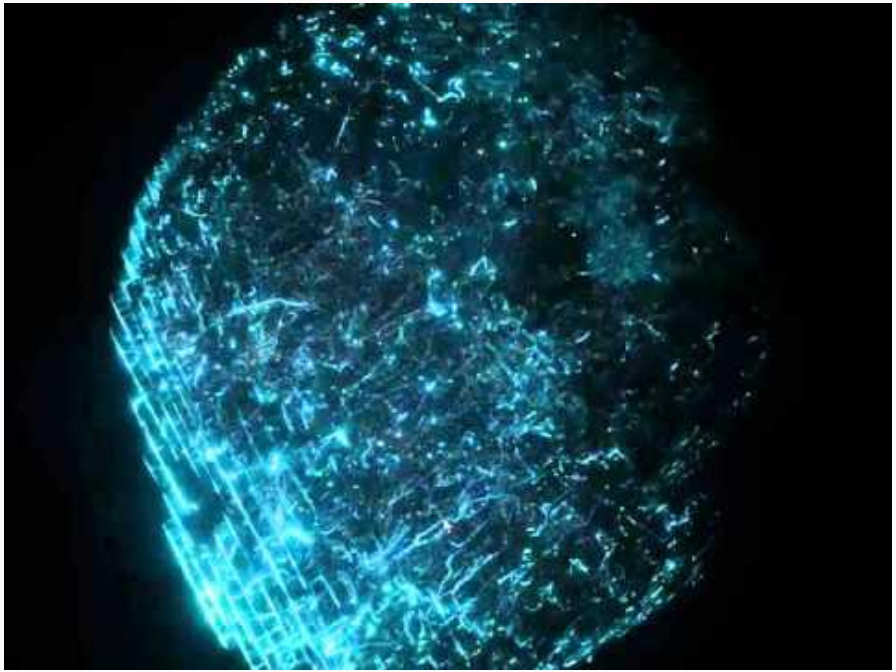


Voxel Use in Brain modelling





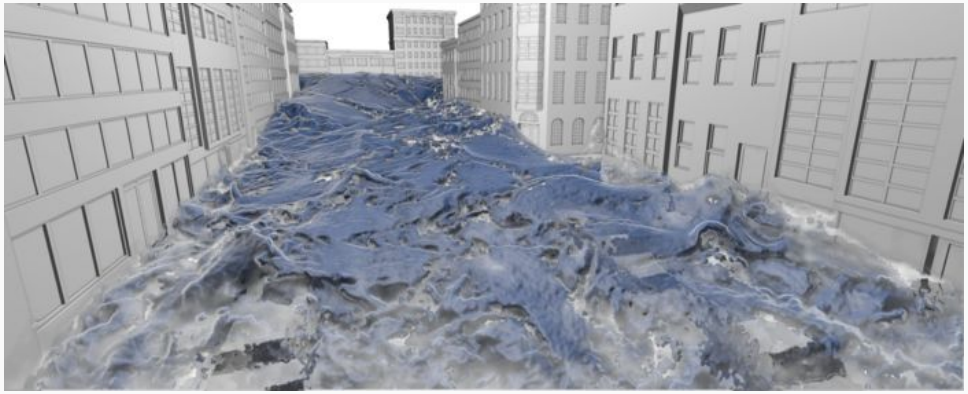


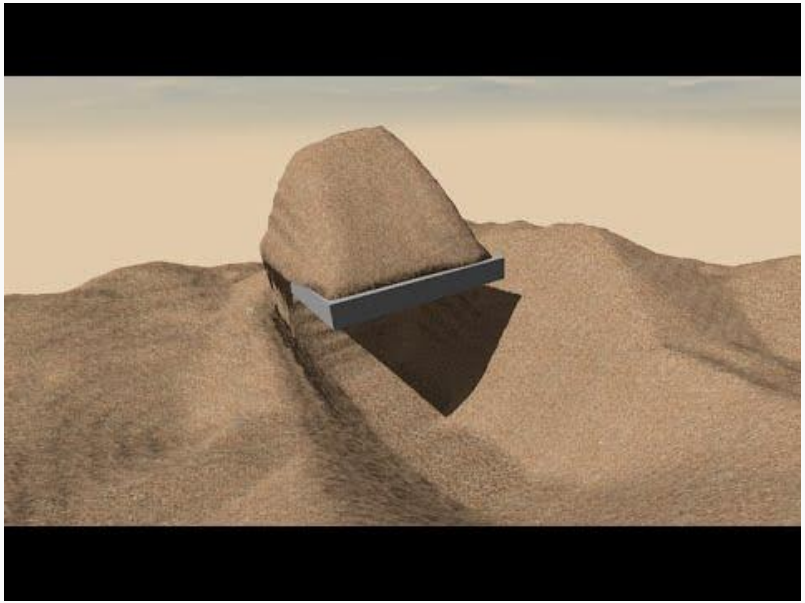


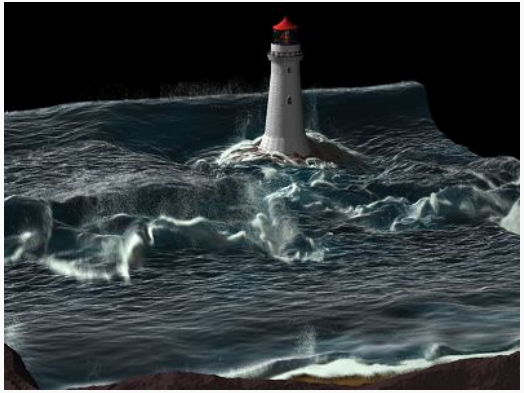
Applications in Physics & Simulation

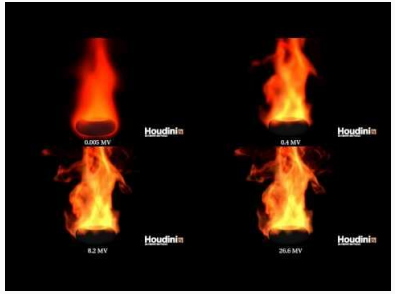
- Voxels, due to the 3 dimension properties, allow for better simulation of many more features of the real world and are starting to become more widely spread for these purposes
- Light
  - The use of voxels in simulation have shown a greater quality and computationally cheaper way of displaying emitted light, shading, and transparency/opacity in a more realistic sense.
- Fluidity
  - The use of voxels has also spread to cinematography as it widely used to simulate items such as fire and smoke as well as liquid itself for CGI

Simulation Modelling for Fluids









https://vimeo.com/5242989

Questions?