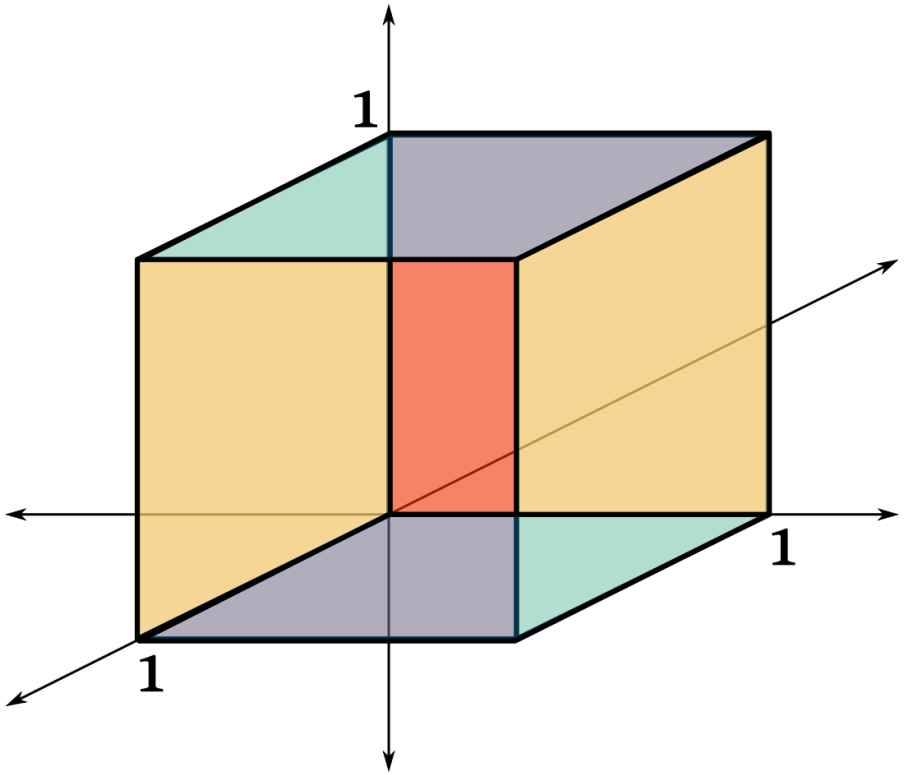
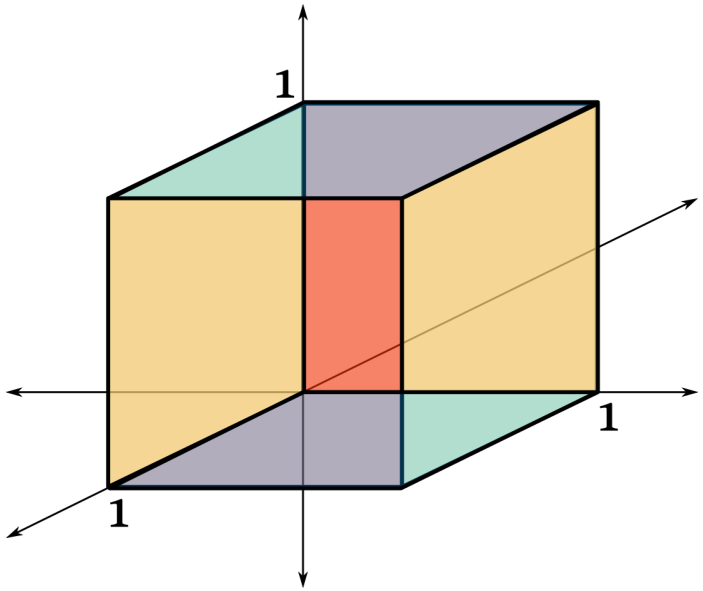
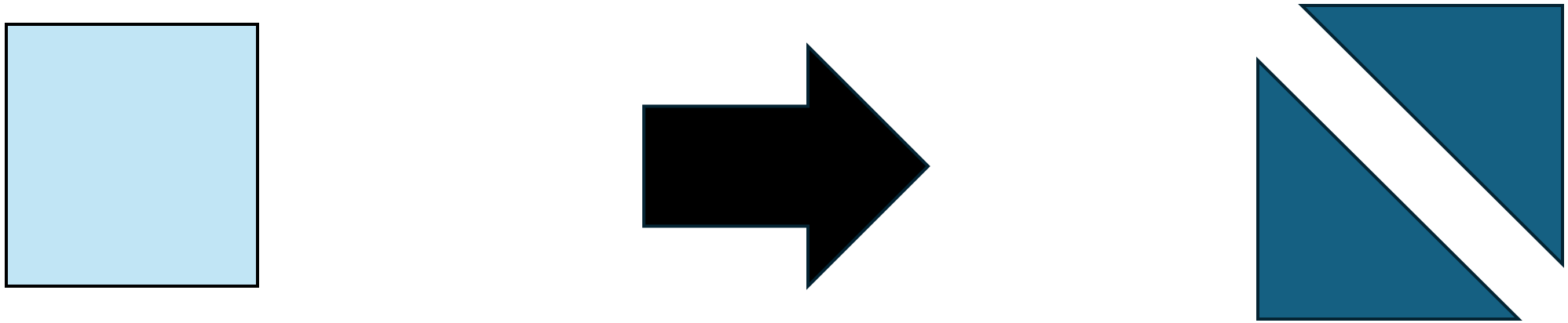
Voxels

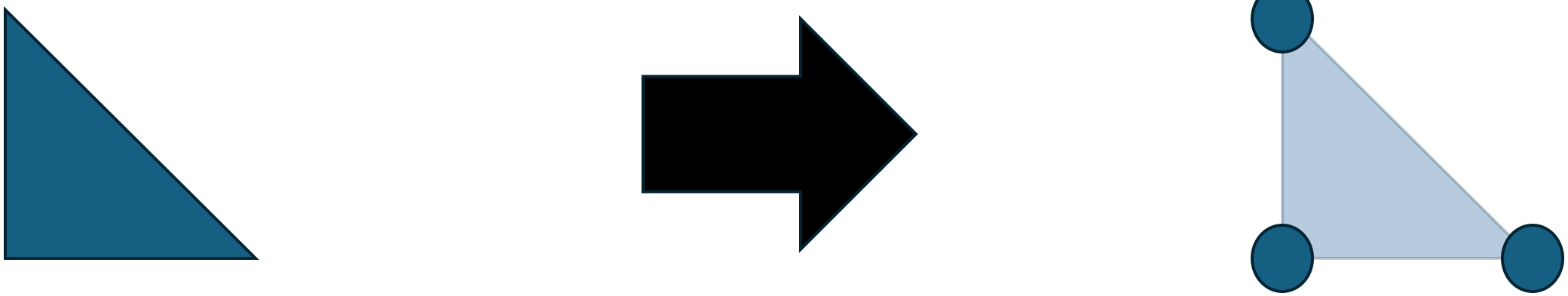
Nick Horton

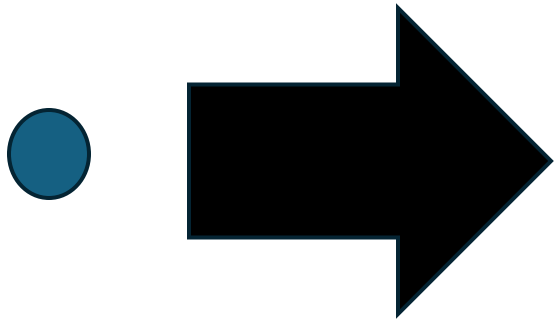
The Humble Cube




1
1
1



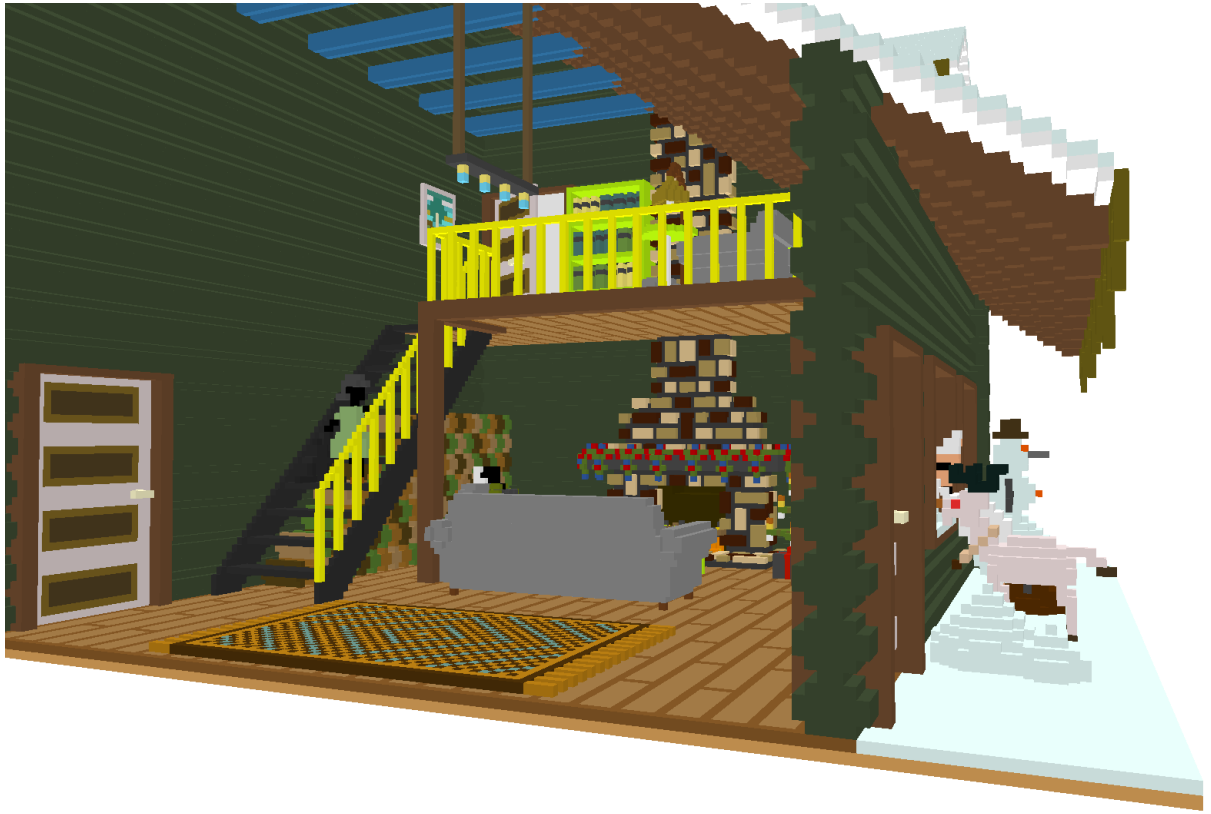




Data Name	Type	Size (bytes)
position	float[3]	12
color	float[3]	12
total		24

24 bytes per vertex
3 vertex per triangle
2 triangles per square
6 squares per cube
230,503 cubes per model
16 models per scene

32*3*2*6*230,503*16
= ~3 billion bytes aka 3 gigabytes of geometry data.



The Scene

On my (bad) laptop this renders at ~6 FPS.

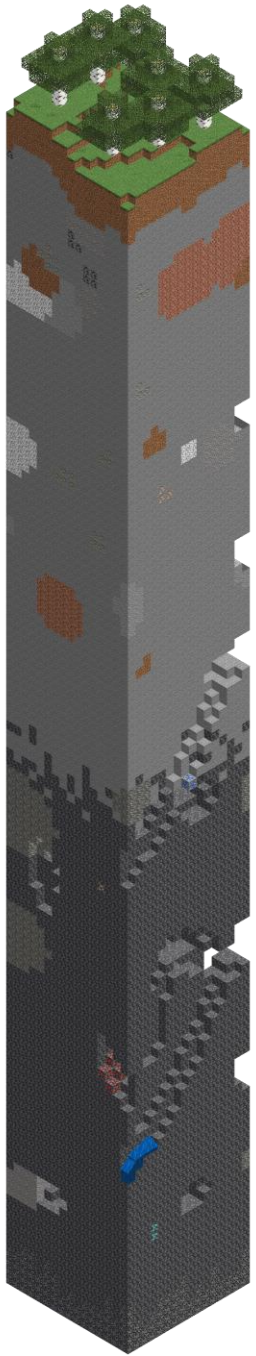
How can we optimize this (memory and FPS)?

Topics

- Chunking
- Memory
- Restrictions
- Instancing
- Culling
- Greedy Meshing

Chunking

- Seperation of concerns
- Finite remeshing size
- Makes frustum culling possible



FPS: 6.6
Mem: 2.9G
DC: 1

FPS: 6.6
Mem: 2.9G
DC: 16

Memory

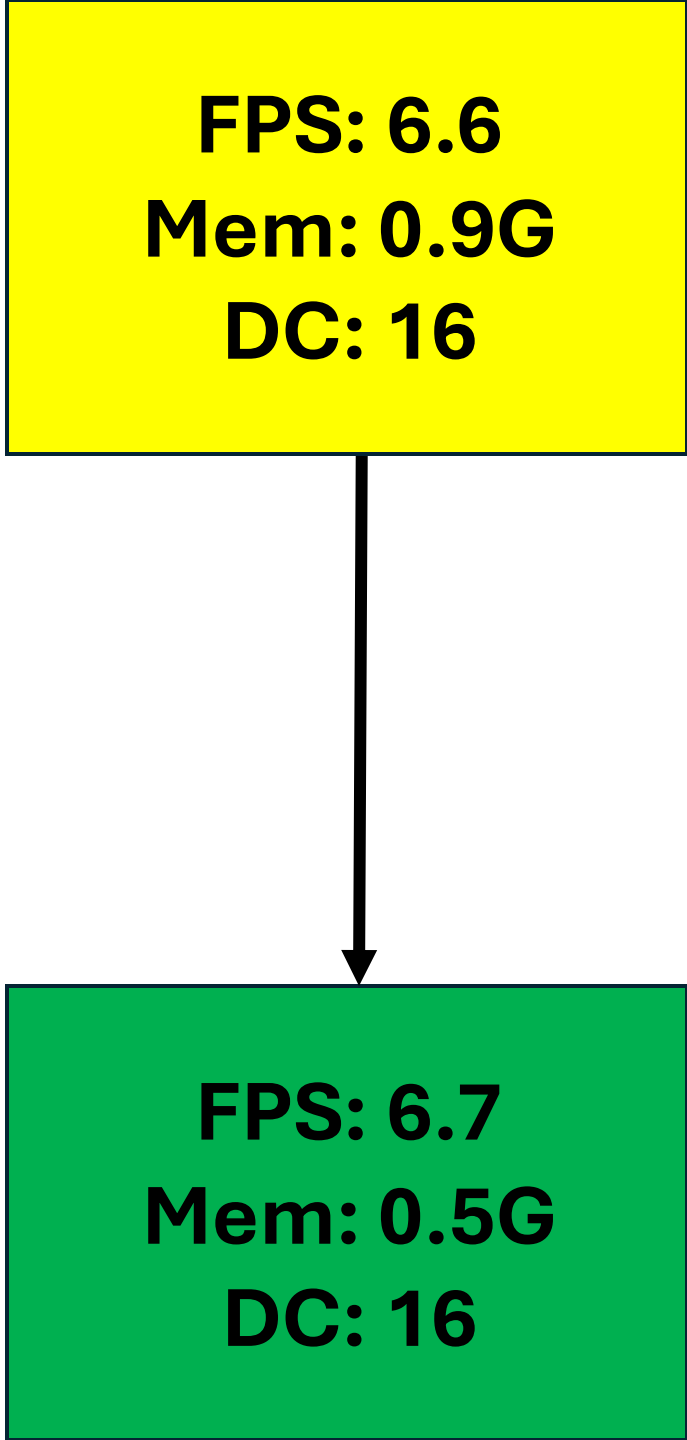
- (most) voxels are grid aligned
- Depending on chunk size you need log2(chsz) bits
- Uniform for chunk offset
- For our case chunking at 256 means 1 uint can contain x,y,z with 8 bits to spare
- Same with color

FPS: 6.6
Mem: 2.9G
DC: 16

FPS: 6.6
Mem: 0.9G
DC: 16

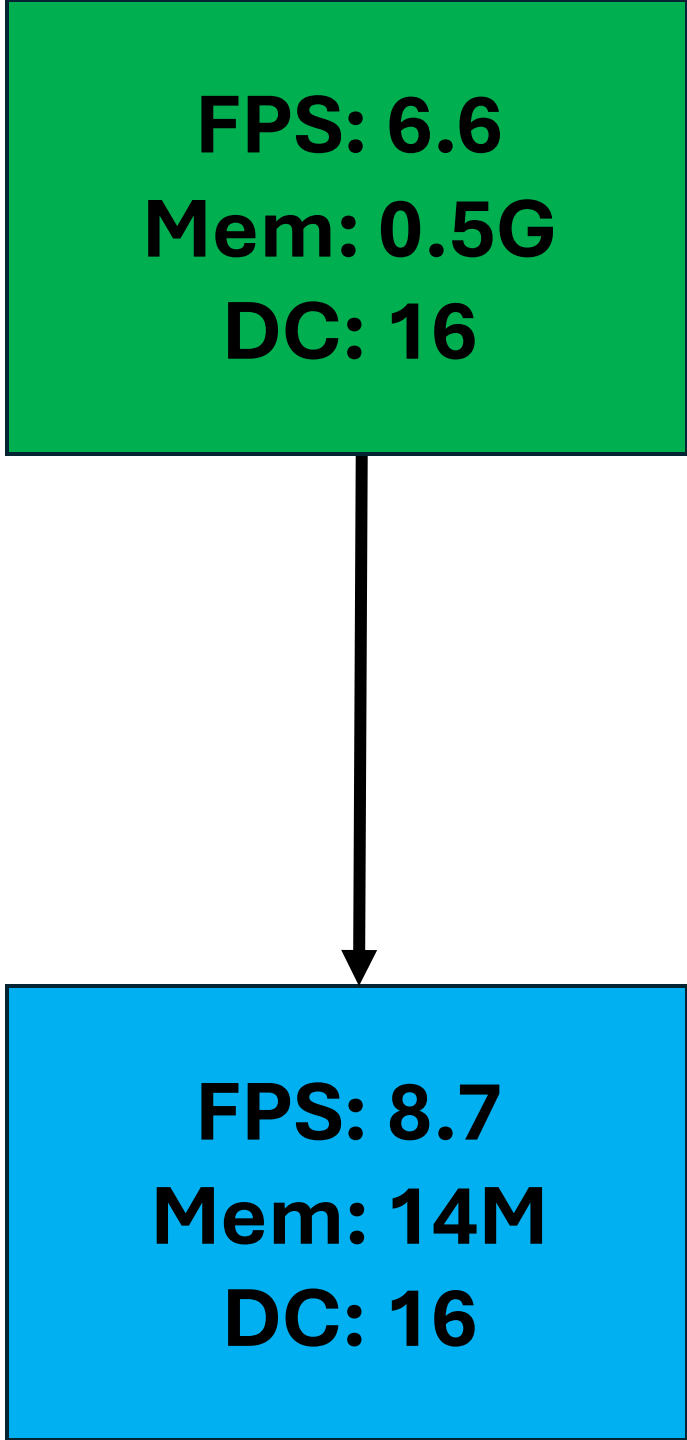
Restrictions

- Most game optimizations make concessions for performance
- Often these restrictions are placed on the artists/designers
- For example (in our case) using a color palette instead of rgb color for each voxel
- Index into palette requires 8 bits luckily, we have an extra byte in the position uint to store this.



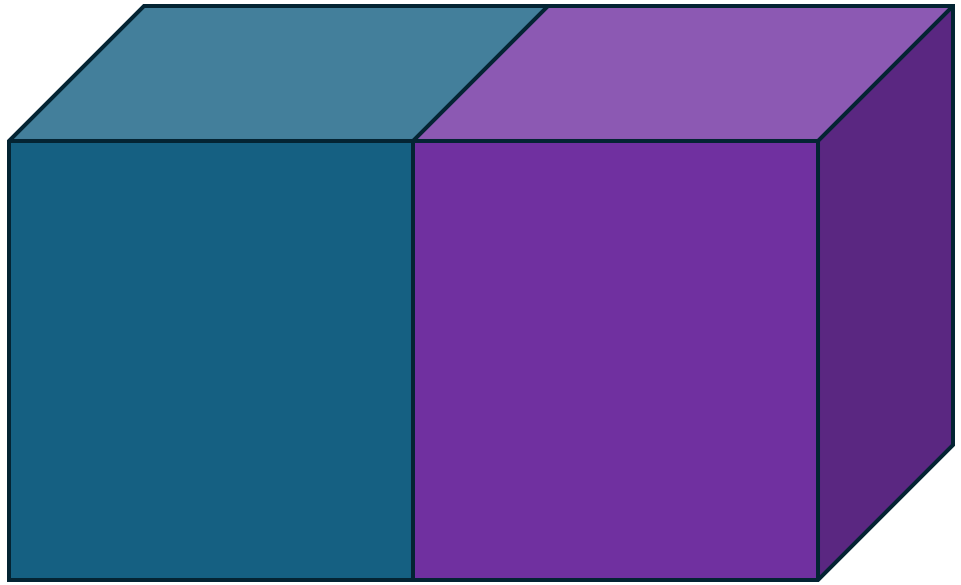
Instancing

- We only have 6 (4) vertexes of real geometry data, that of a square.
- Everything else is per square or per cube
  - Direction of square
  - pos and col of cube
- We have A LOT of duplicate data



Face Culling

- Most faces we render will never be seen.
- Removing these faces is an expensive calculation but chunking makes it possible in realtime.



FPS: 8.7
Mem: 14M
DC: 16

FPS: 43.7
Mem: 14M
DC: 16

Greedy Meshing

- Extremly expensive calculation
- Mathy to implement
- So so so fast for static meshes