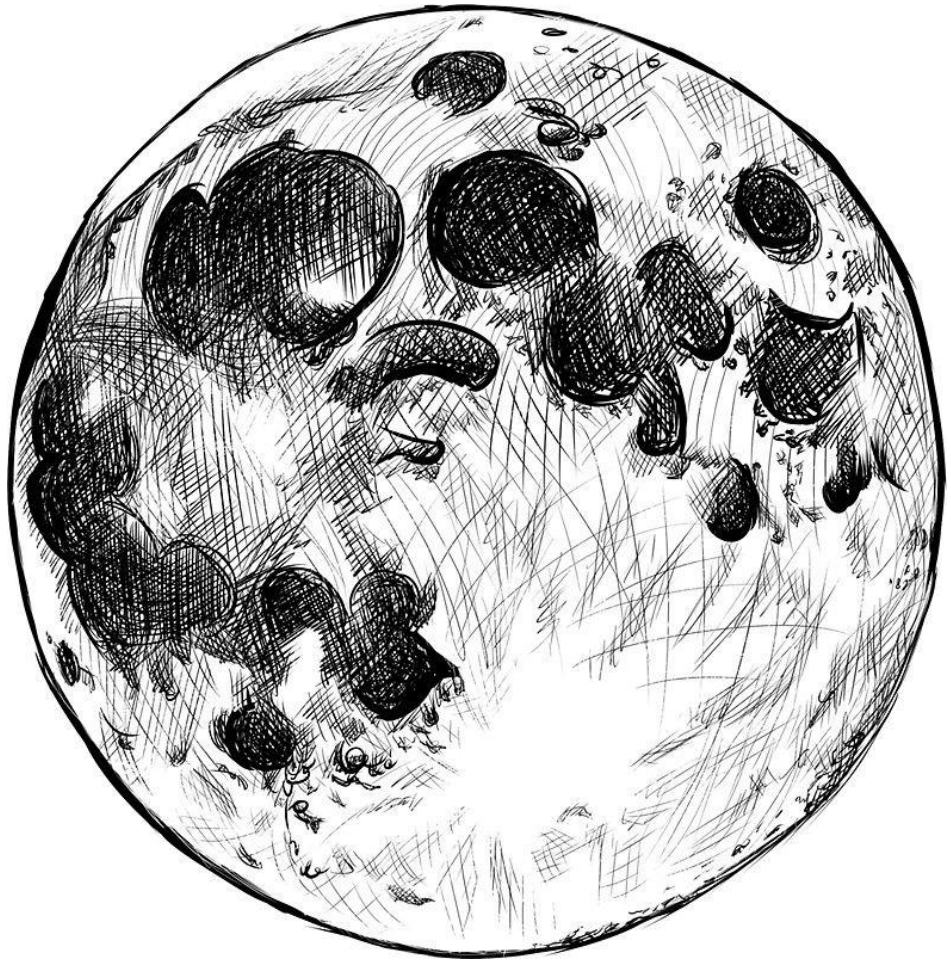
How to draw the Moon

A not-fancy approach

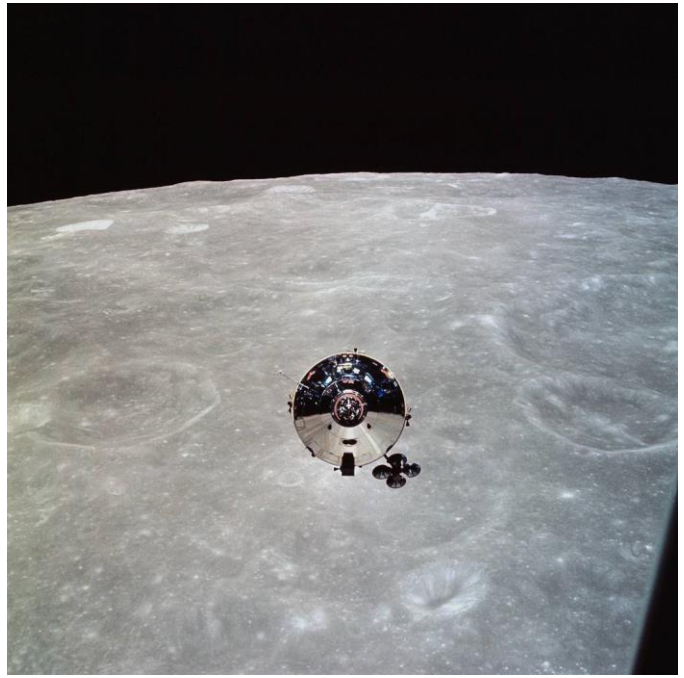


How to ~~draw~~ render the Moon

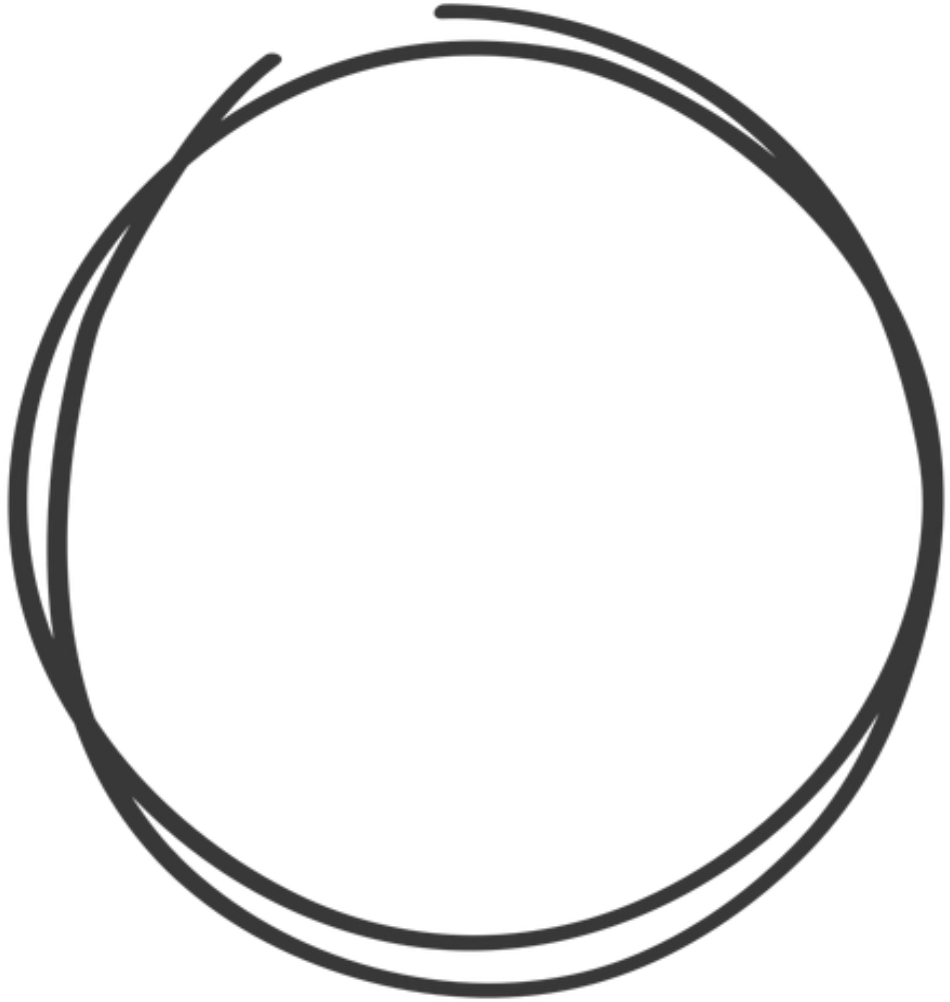
A not-fancy approach

Requirements

- Unified 3D view
- Arbitrary level of detail
  - Dust on ground (3D, not painted)
  - View from orbit
  - Everything in between
- 2 weeks to build it



First, draw a circle



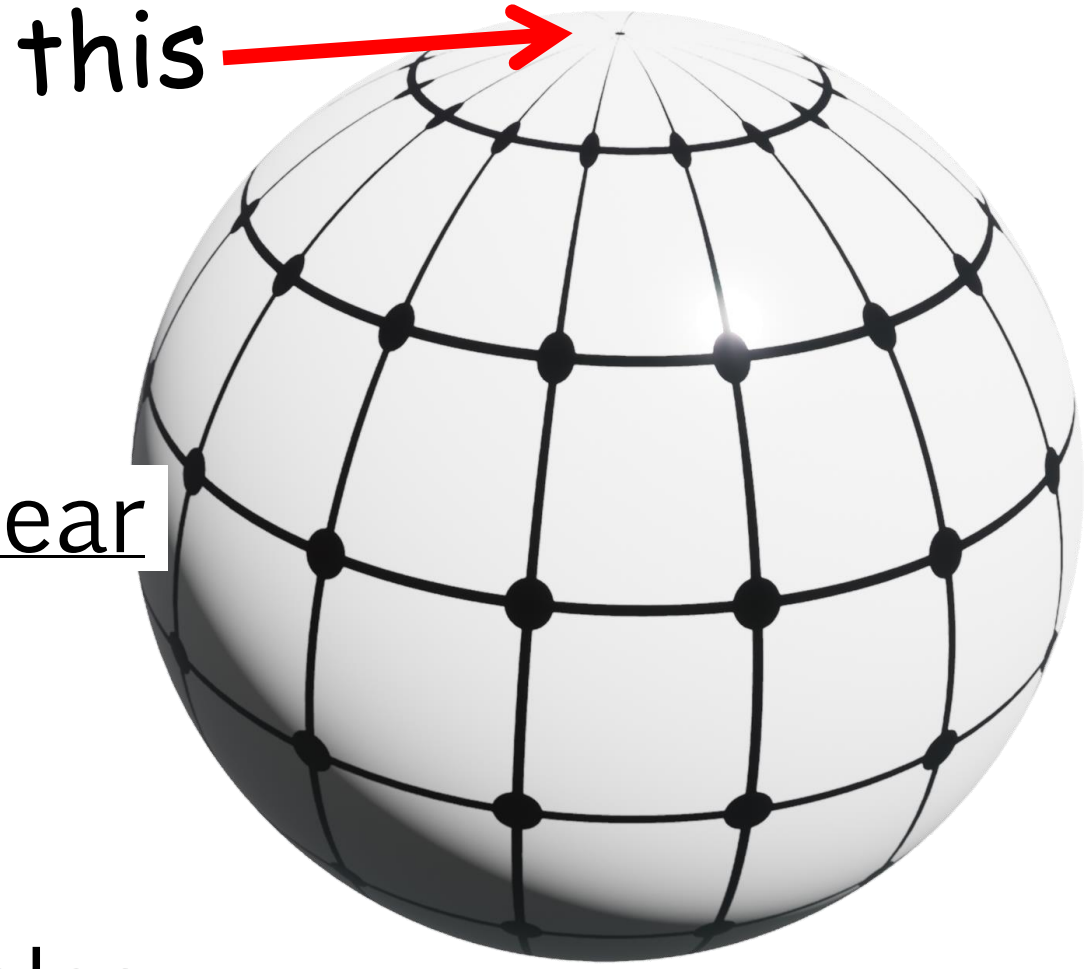
render

First, ~~draw~~ a

~~circle~~

sphere

The problem with spheres



- Computer memory is linear
  - ∴ Want linear (or rectangular) data layout
- Spheres are bad rectangles
- **Use a cube instead**

render

First, ~~draw~~ a

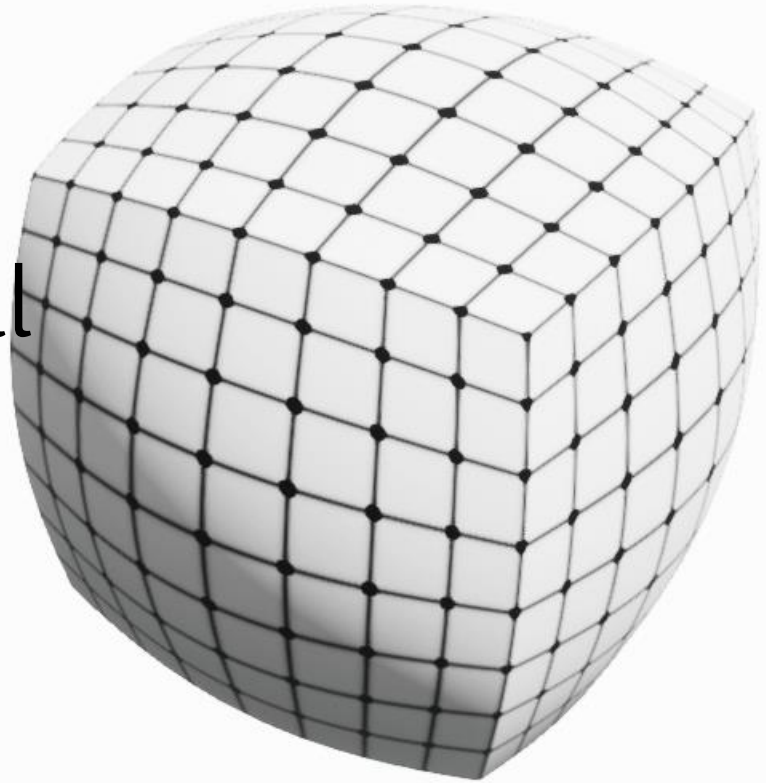
~~circle~~

~~sphere~~

cube

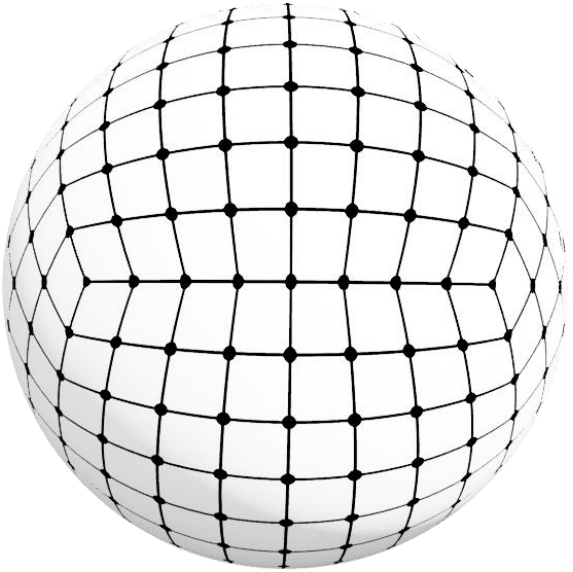
A cube?

- But cubes are… not spherical
- So, spherise our cube
  - Split edges, adding more sections
  - Pull the vertices inward to where a sphere *would* be



Better Uniformity

- Use Equi-Angular Cubemaps[1] technique
- Take $\tan\frac{point * \pi}{4}$ before pulling vertices in to sphere



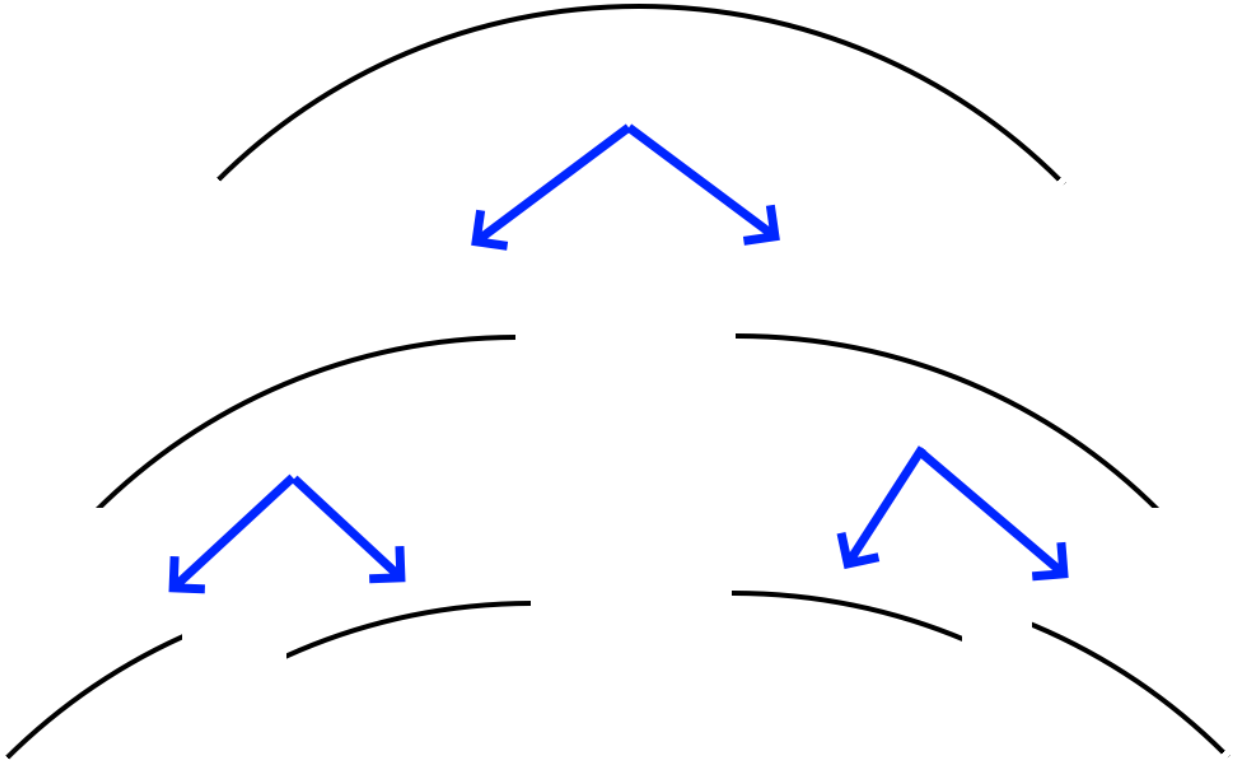
[1] https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/

How to scale?

- Insufficient compute + memory to use high-resolution data everywhere at once
  - Moon surface area ≈ 37,936,695,000,000,000,000 mm²
- Want max detail near camera, less in distance
- **Render hierarchically**

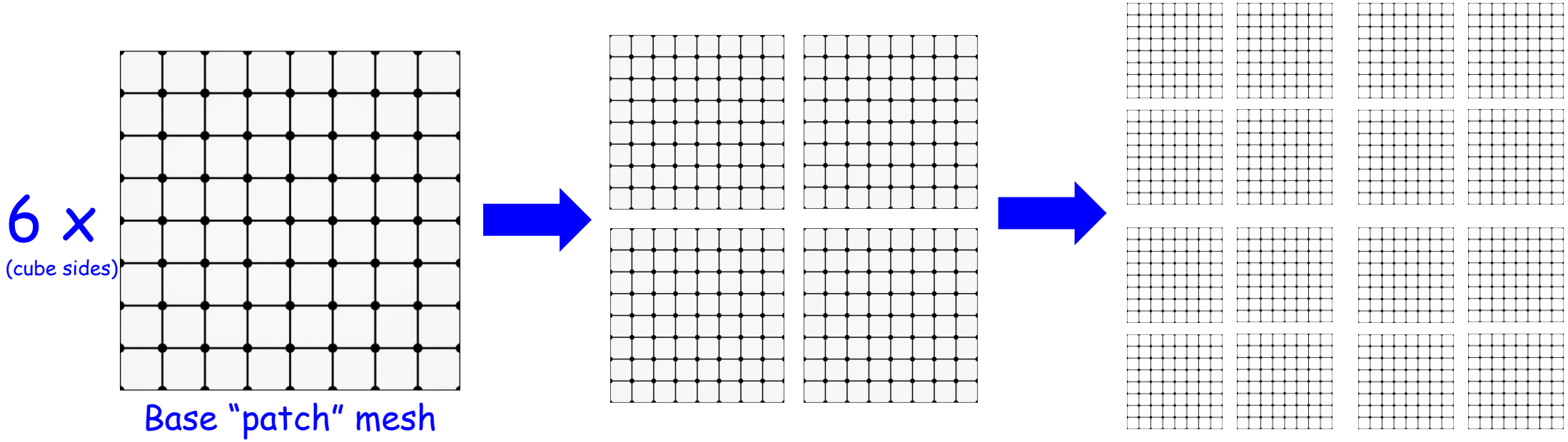
Hierarchical Rendering

(2D analogy)



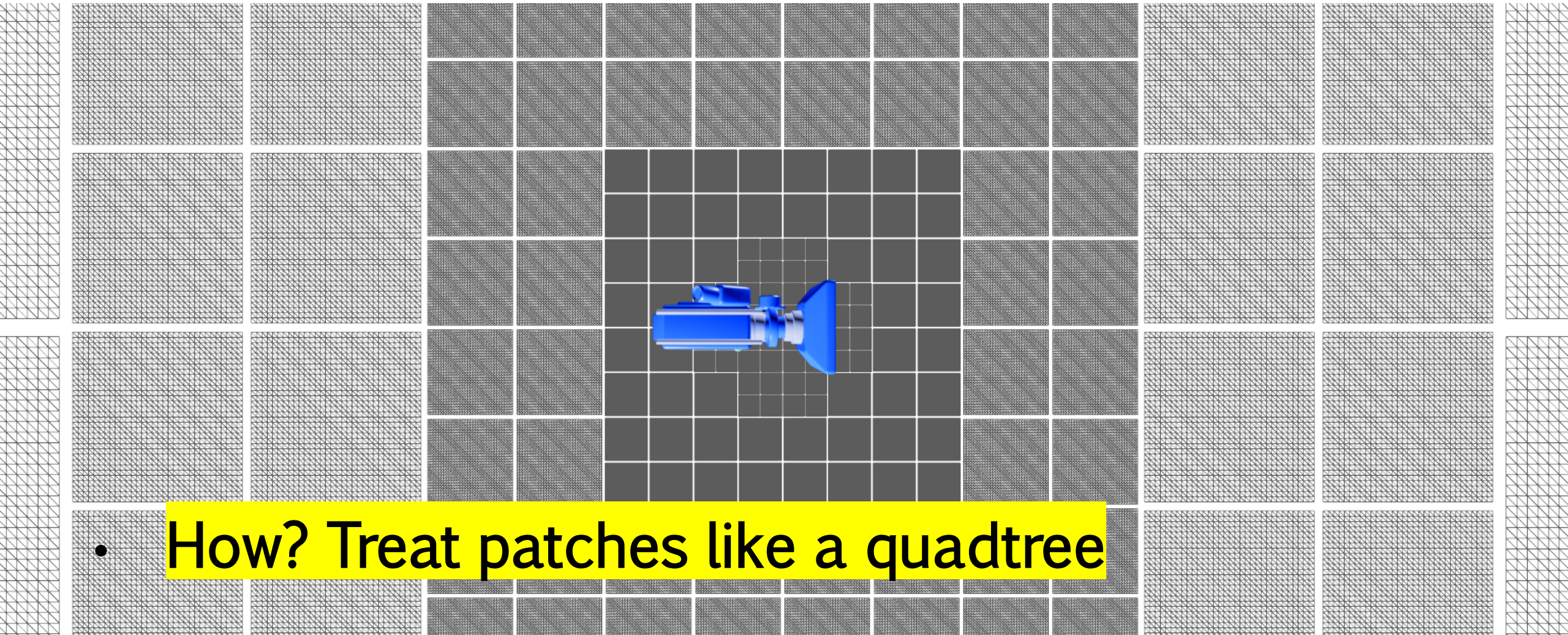
- Select size based on distance from camera

Hierarchical Rendering



- Same base mesh, repeated at different scales
- Very GPU friendly

Hierarchical Rendering

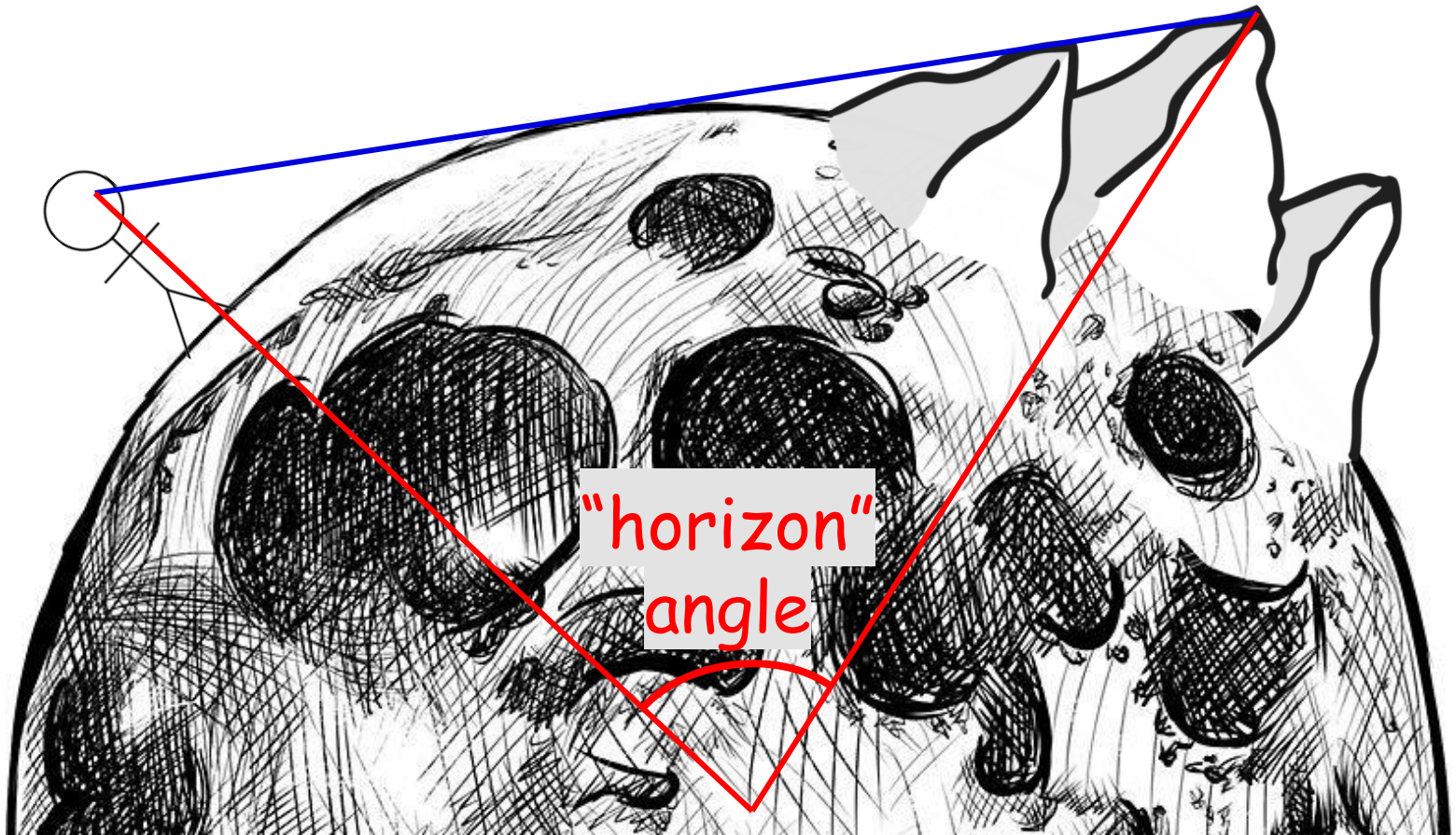


- How? Treat patches like a quadtree

Quadtree DFS (CPU)

```
void SearchQuadTree(float2 center, float size, Map& output) {
    if (Distance to Camera < Threshold * size) {
        output[center] = size
        return
    }

    SearchQuadTree(center + size * (-0.5, -0.5), size * 0.5, output)
    SearchQuadTree(center + size * (+0.5, -0.5), size * 0.5, output)
    SearchQuadTree(center + size * (-0.5, +0.5), size * 0.5, output)
    SearchQuadTree(center + size * (+0.5, +0.5), size * 0.5, output)
}
```

Horizon Optimization



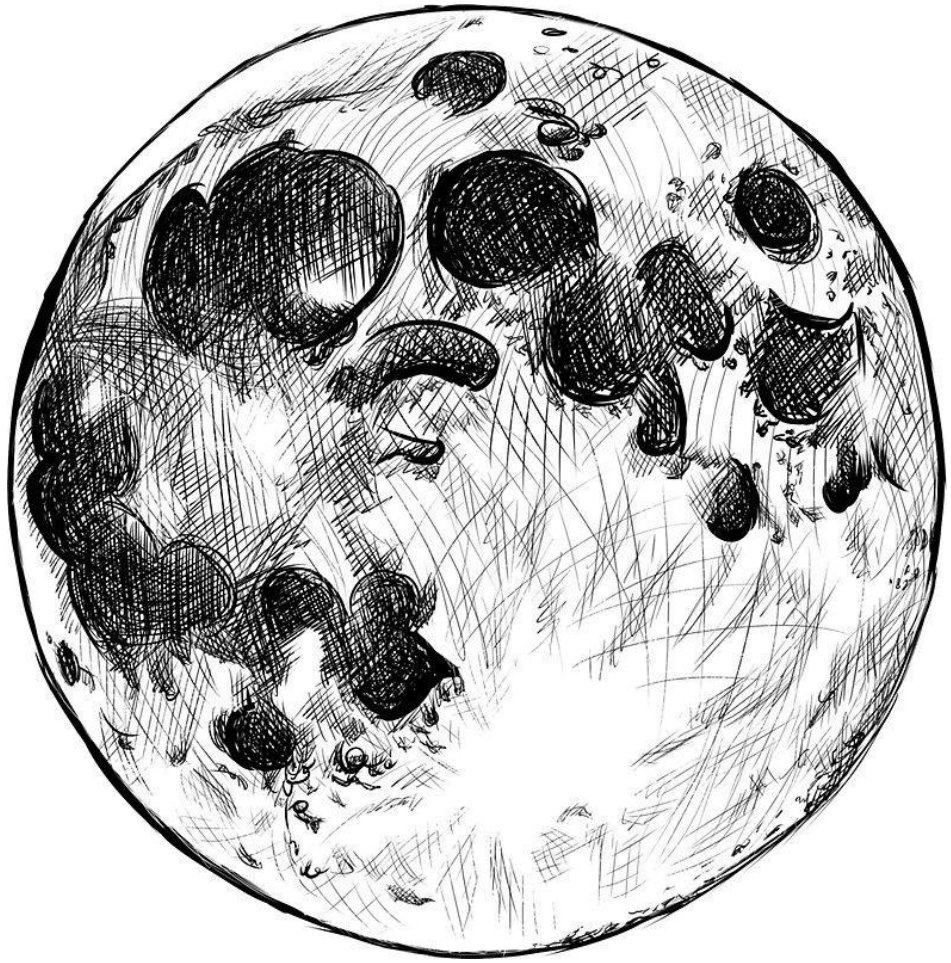
Horizon Optimization (CPU)

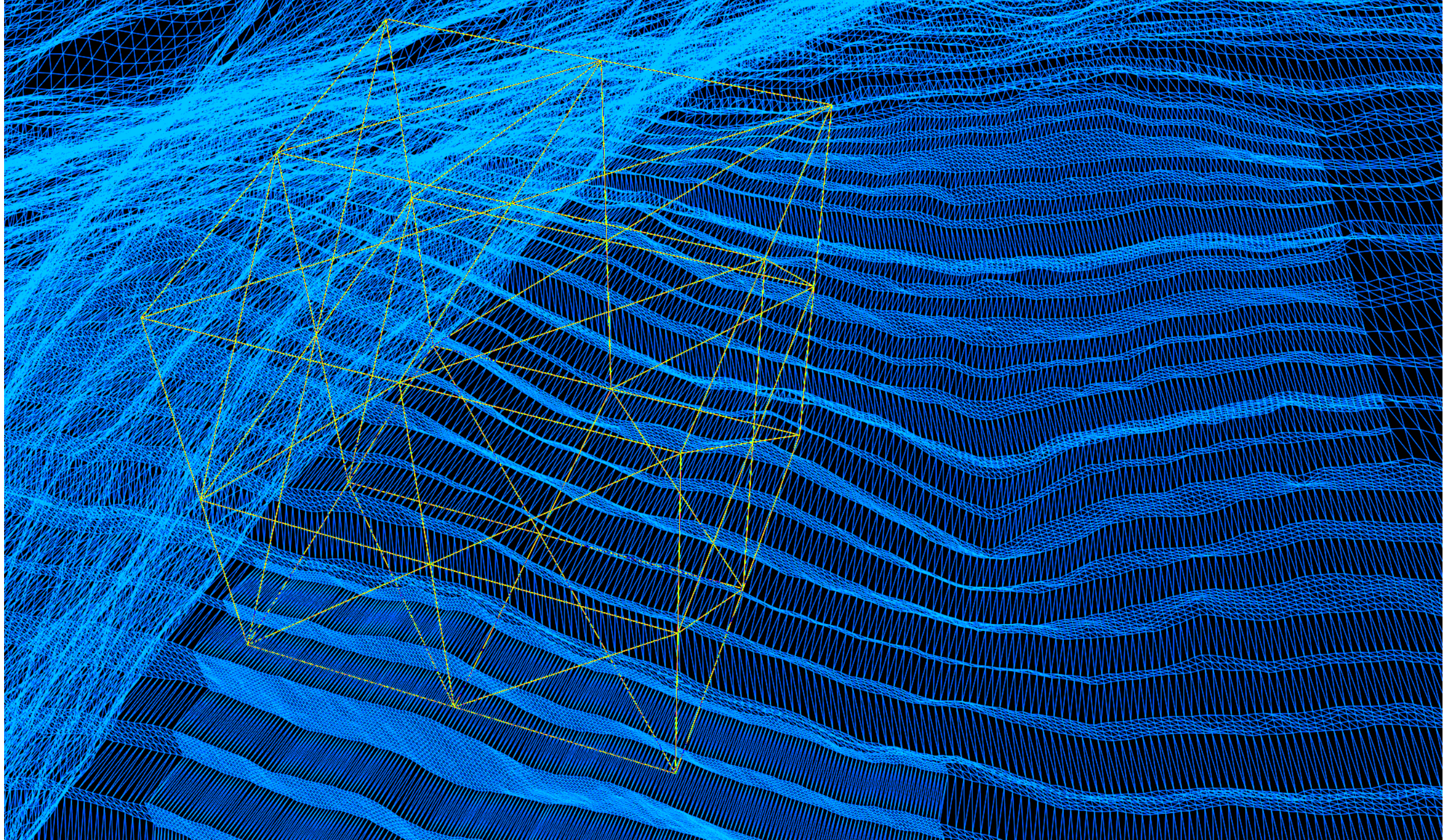
```
void SearchQuadTree(float2 center, float size, horizon, Map& output) {
+    if (Angle to Camera > horizon) {
+        return
+    }

    if (Distance to Camera < Threshold * size) {
        output[center] = size
        return
    }

    SearchQuadTree(center + size * (-0.5, -0.5), size * 0.5, horizon, output)
```

Render a spherised cube

… we’re done?

Not quite!





Floating-Point Precision

- 32-bit floats are precise to at most 1 in 2^{24}
- The radius of the Moon is 1737.4km
  - 1 part in 2^{24} is ~10cm
  - Not precise enough for our needs
- **Emulate doubles using float pairs**

Double-Float Representation

- Use two floats to represent (nearly) a double
- How to turn a double into two floats?

```
double d = 3.141592653589793
float hi = (float)d
float low = (float)(d - hi)
doublefloat df = { hi, low }
```

- The double is the **sum** of the two floats

Controlled Overflow

- Add: float a + float b, producing doublefloat

```
float sum = a + b
float error = b - (sum - a)
return { sum, error }
```

- Multiply: float a * float b, producing doublefloat

```
float product = a * b
float error = fma(a, b, -product) // a * b – product
return { product, error }
```

Double-Float Arithmetic (GPU)

- + and * below are operations from previous slide
- Addition is straightforward

```
df1 + df2 = df1.hi + df2.hi + df1.low + df2.low
```

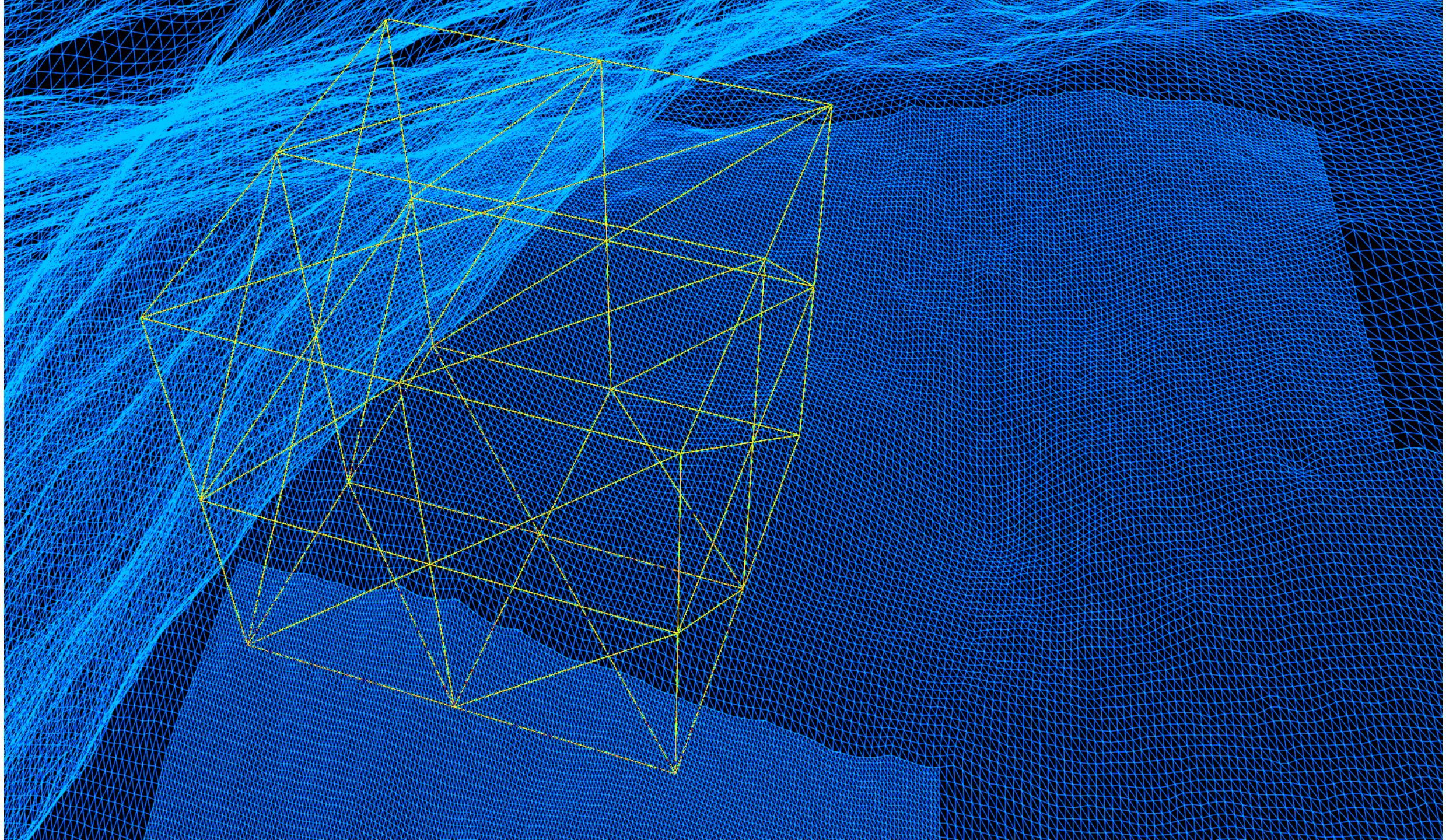
- Multiplication works like algebra

```
df1 * df2 = (df1.hi + df1.low) * (df2.hi + df2.low) // Conceptually
          = (df1.hi * df2.hi) + (df1.hi * df2.low) + (df2.hi * df2.low)
```

Inverse Square root (GPU)

- We approximate result using hardware float sqrt
- Then perform one iteration of Newton's method

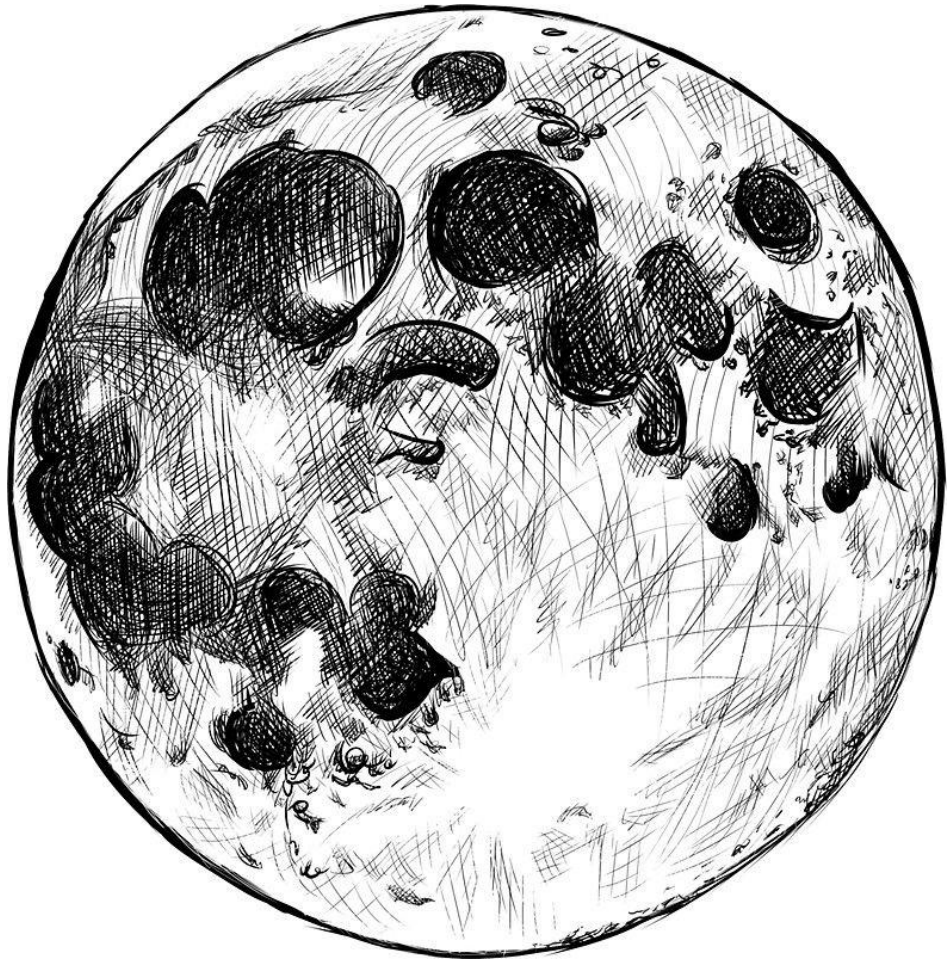
```
float approx = 1.0 / sqrt(df.hi)
// Multiplies by powers two only alter exponents, so can use hardware op
// Therefore, calculating df * 0.5 is cheap
doublefloat halfDf = df * 0.5
return approx * (1.5 - halfDf * (approx * approx))
```



Render a *huge* spherised cube

… we're done???

Yes! 



Result?

- Unified 3D view ✓
- Arbitrary level of detail
  - Dust on ground (not painted on)
  - View from orbit