# NOISE

Richard Stump

# 1) PROBLEM

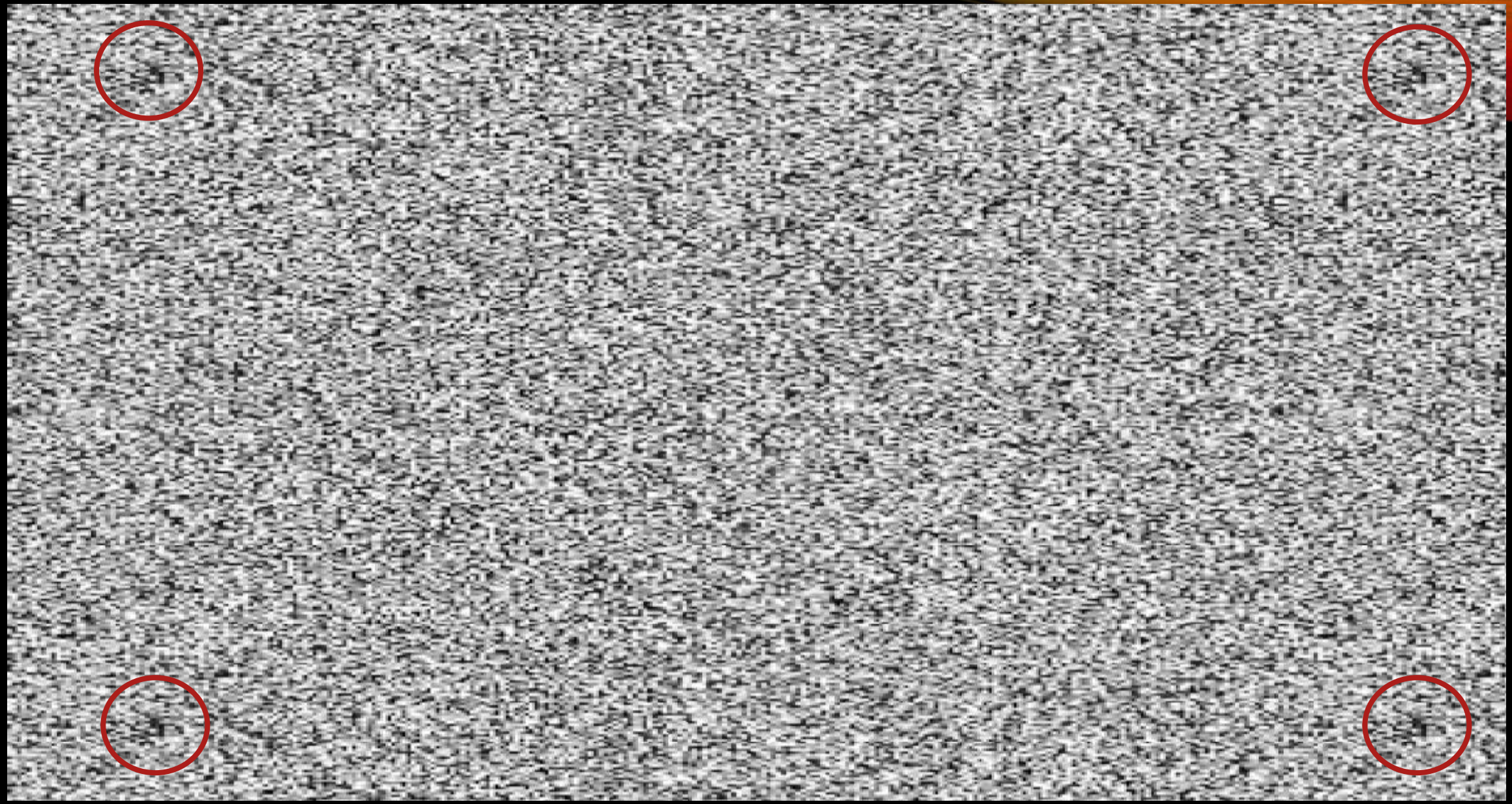
# 1) PROBLEM:

- Implement Noise Functions
- Visualize them in a cool way
- Make it run in real time

# 2) SOLUTIONS

# GENERATING

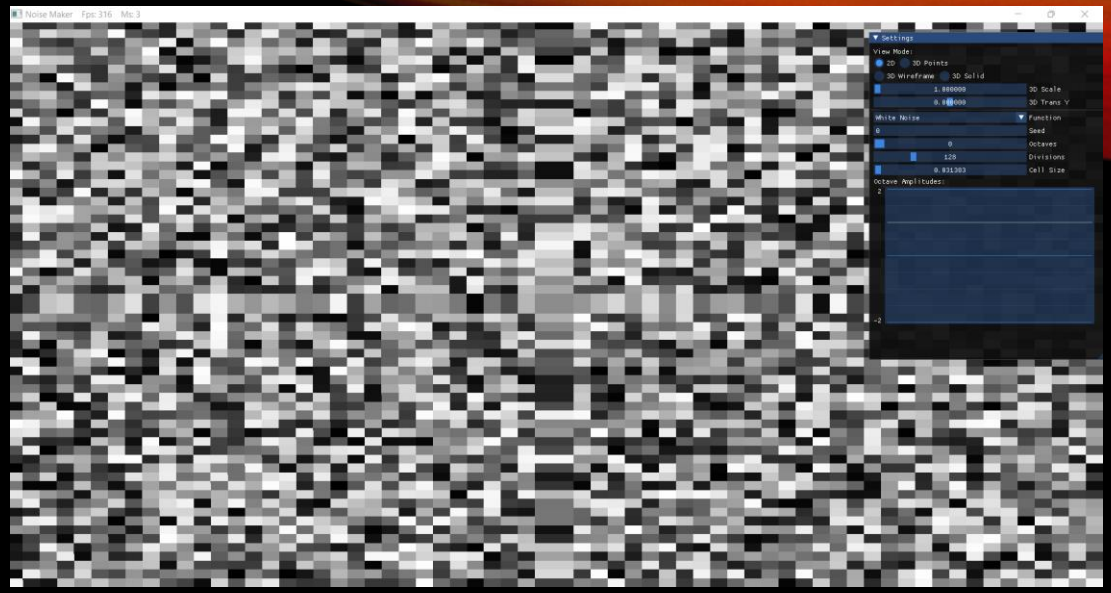
1) Pre-calculate lookup tables
2) For each sample, calculate its integer coordinates and use the lookup table
3) Transform coordinates by octaves and scale
4) Take weighted sum of octaves

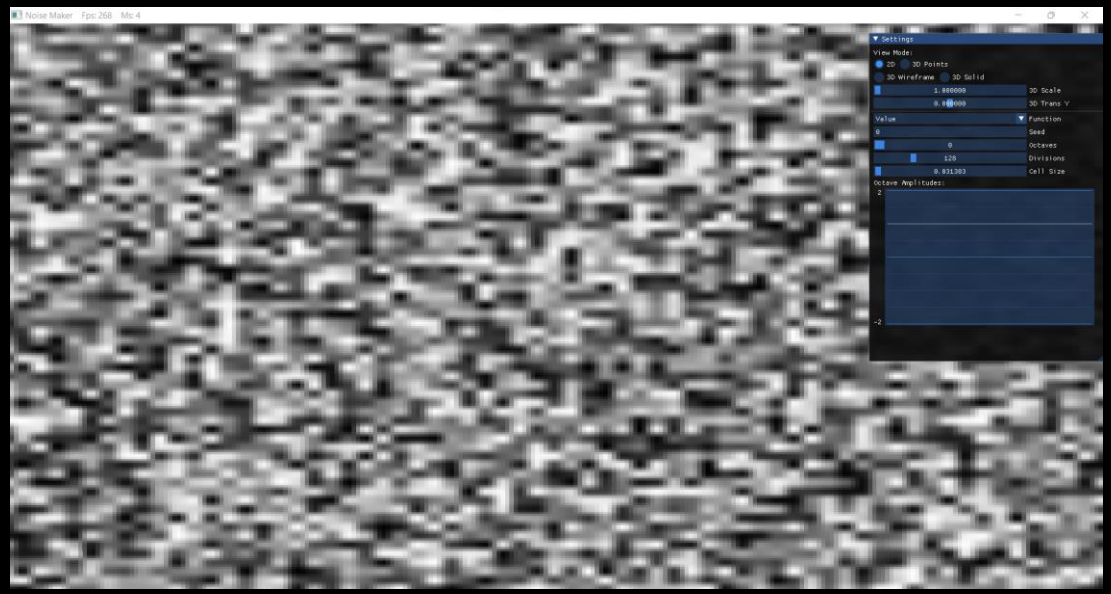


# FUNCTIONS

1) White Noise:
    1) Give each integer coordinate a random value
    2) Use value of nearest integer coordinate

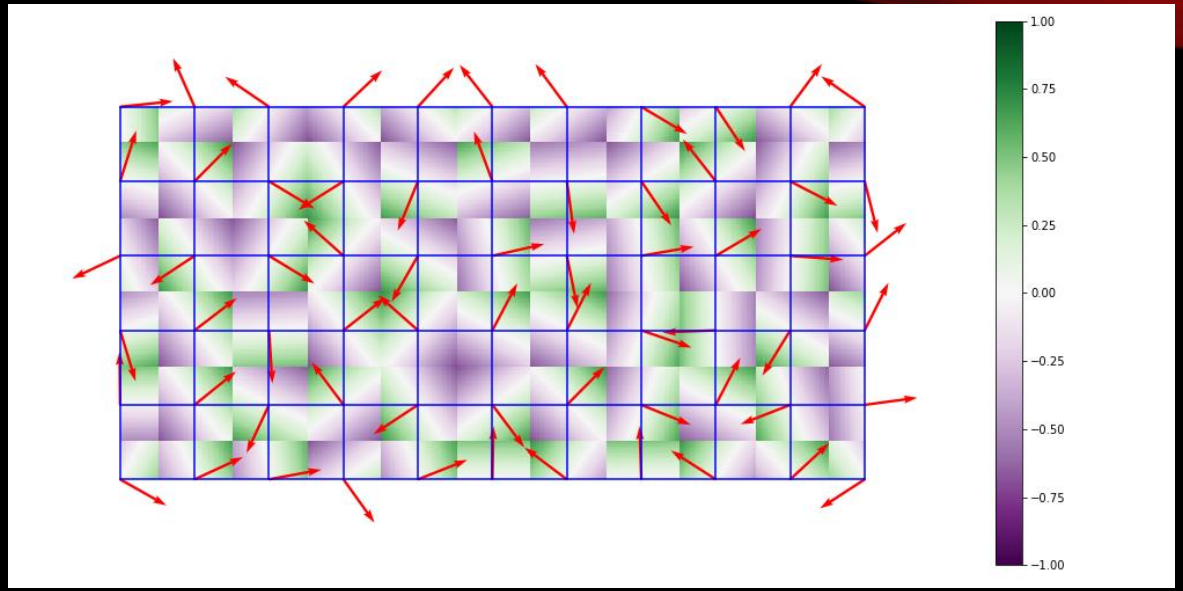
2) Value Noise:
    1) Same as white noise, except values are interpolated between nearest grid points





# FUNCTIONS

1) Perlin Noise:
    1) Give each grid point a random direction (gradient)
    2) Take dot product
    3) Interpolate between dot products

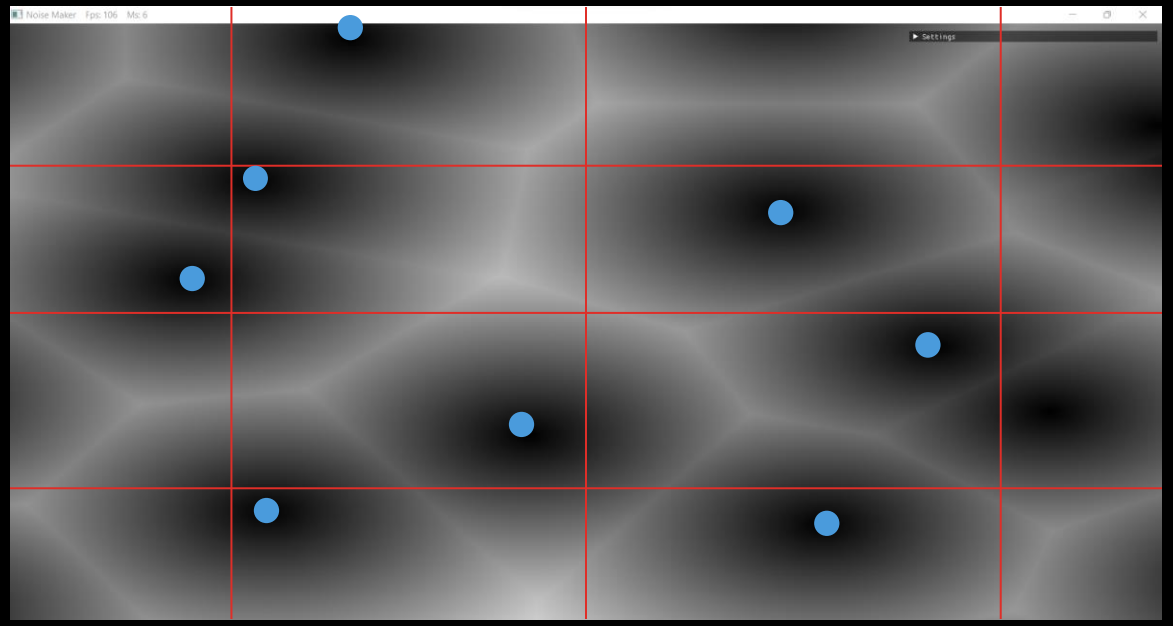



https://en.wikipedia.org/wiki/Perlin_noise

# FUNCTIONS

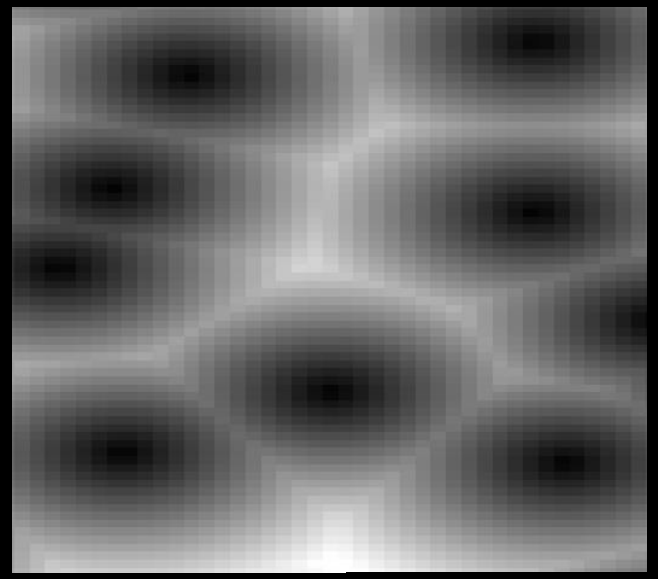
1) Worley
    1) Give each cell a random point
    2) For a sample point, result is the distance to the closes point

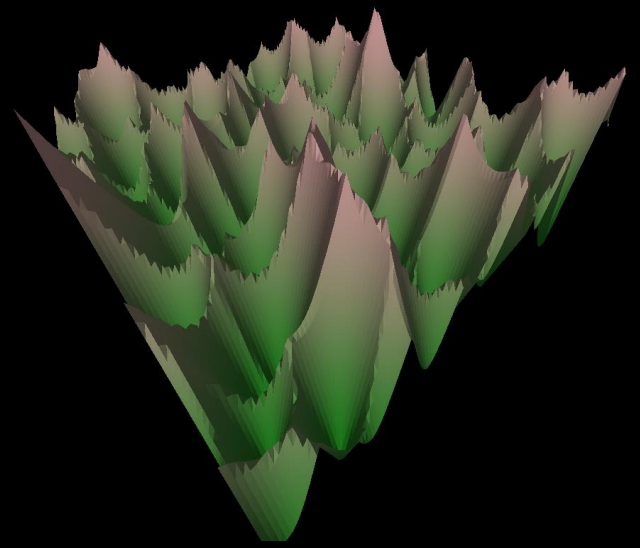
2) Worley F2
    1) Same as Worley, but using 2nd closest point



# VISUALIZING

1) Sample noise to a buffer
2) Use single buffer to visualize it different ways
    1) Grayscale texture
    2) Sample points
    3) Wireframe
    4) Filled Triangles





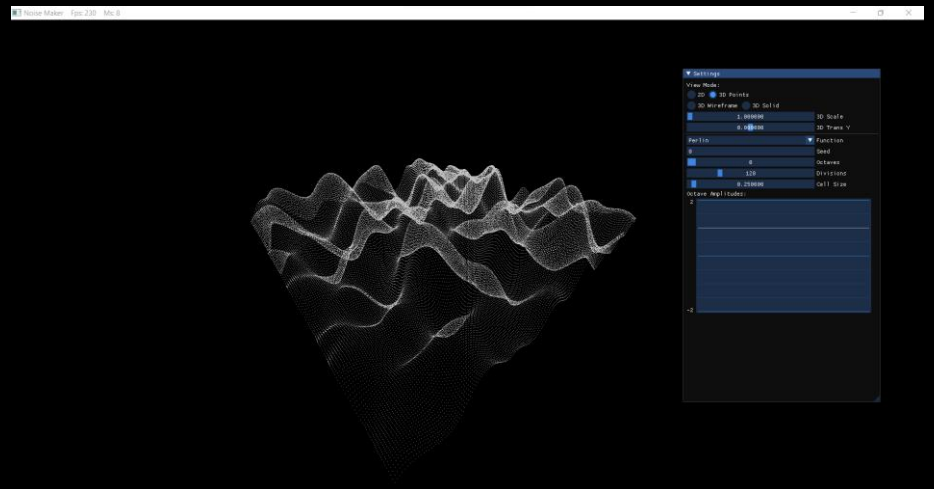
# 3) CODE DESCRIPTION

# FOR EACH NOISE FUNCTION:

1) Inherit from a common base class:
    1) float sample(float x, float y)
    2) void reseed(int seed)
2) Pre-calculate 256x256 table of random values in constructor
3) Use coordinates of samples
    1) Use closes integer coordinates to index table
    2) Combine results

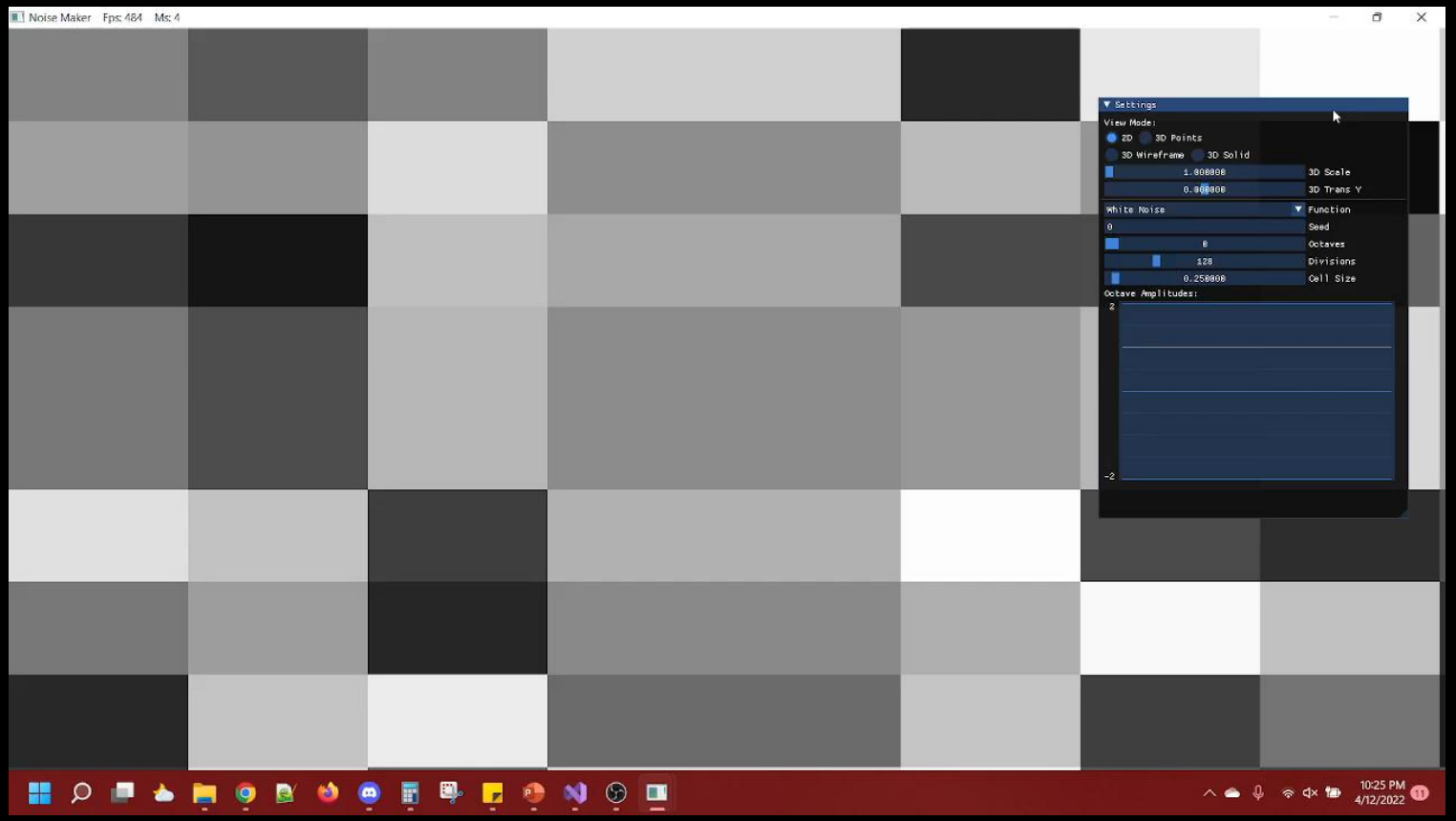
# MAIN LOOP:

1) Handle UI
    1) If the seed changes, regenerate the lookup table
    2) If any other parameter changes, resample
2) Render according to the selected method
    1) 2D – The buffer is a texture
    2) 3D – Generate points on the fly



# 4) LIVE DEMO

# BACKUP VIDEO




https://youtu.be/c2gzPOT0tbk

# 5) LIMITATIONS AND PROBLEMS

# LIMITATIONS:

- One function at a time
- Octaves have set frequency
- No turbulence
- Texture output is grayscale

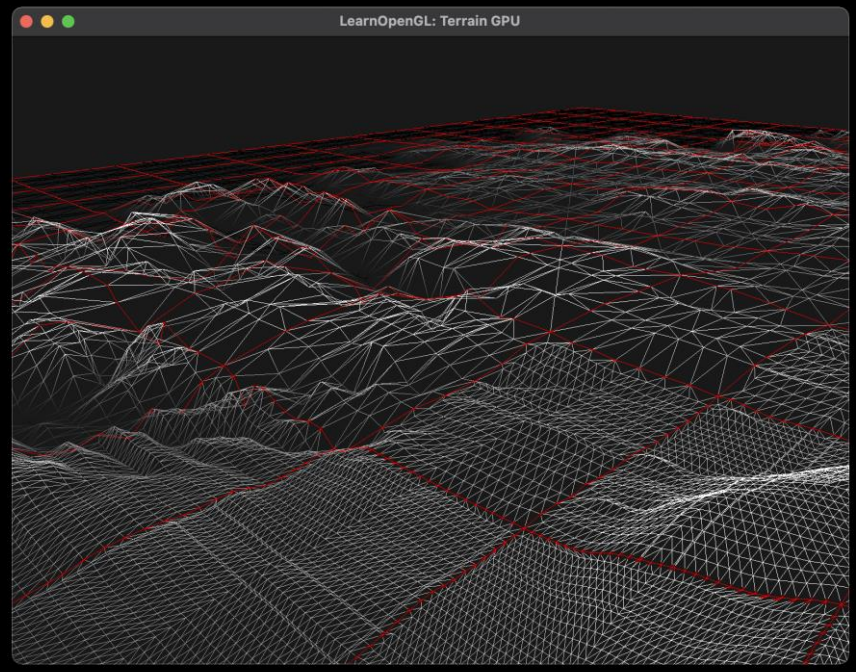
# PROBLEMS:

- Memory efficiency could be improved
- 3D Views are very GPU intensive at higher detail levels

# 6) FUTURE WORK

# TESSELLATION SHADERS

- Level of detail changes with distance
- Reduces GPU workload




https://learnopengl.com/Guest-Articles/2021/Tessellation/Tessellation

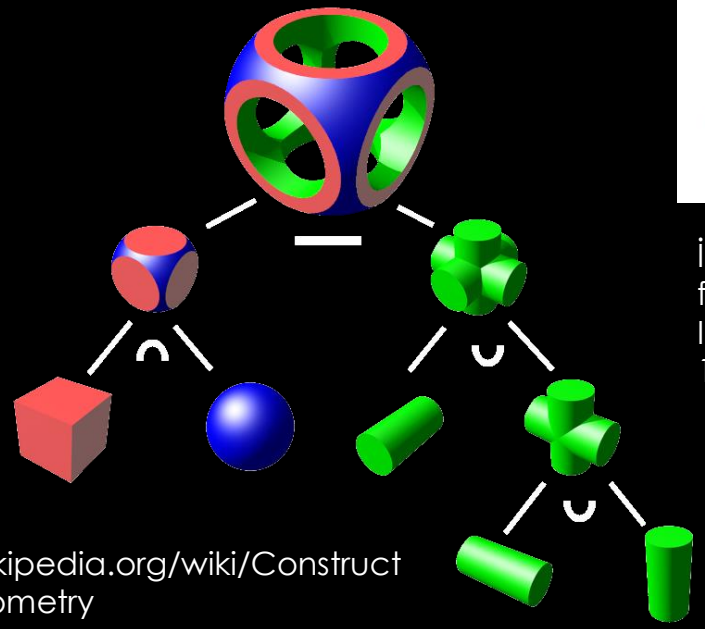
# ADD A SCRIPTING SYSTEM

- Possibly Lua
- Compose and combine in weird ways
- Akin to Ken Perlin's image synthesizer

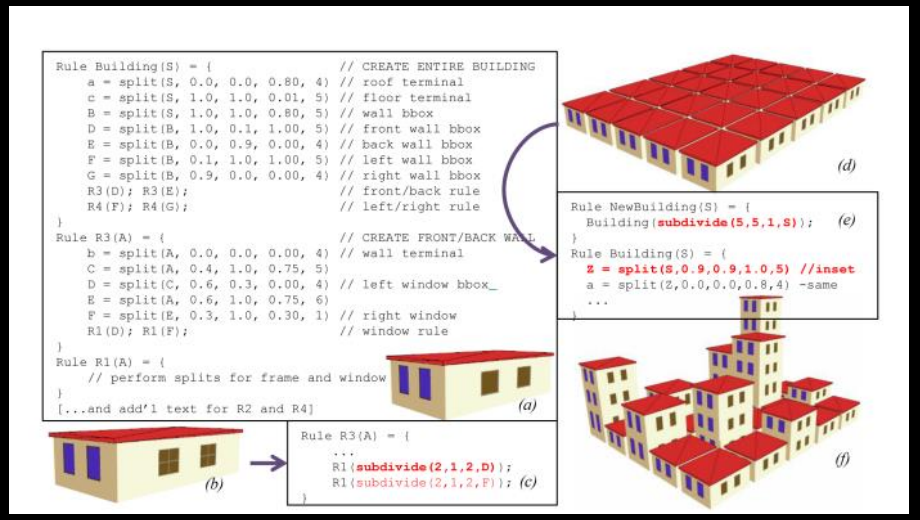
```
lightness = octaves(
        {0.75, 0.2, 0.3, 0.6},
        perlin(u, v, 32, "124as514g")
    );

hue = octaves(
        {0.13, 0.5},
        worley(
            pearlin(u, v, 16, "1324a"),
            perlin(u, v, 16, "sdf123as"),
            8,
            "asdgqe"
        )
    )

hue = clamp(hue, 0, 1)

return hslToRgb(hue, 1, lightness)
```

# PROCEDURAL MODELING

- Split Grammars?
- Guided Procedural Modeling?
- Constructive Solid Geometry?



https://en.wikipedia.org/wiki/Constructive_solid_geometry




İ. Demir, D. G. Aliaga and B. Benes, "Proceduralization for Editing 3D Architectural Models," 2016 Fourth International Conference on 3D Vision (3DV), 2016, pp. 194-202, doi: 10.1109/3DV.2016.28.

# PROCEDURAL WORLD/GAME

- Super small disk footprint
- Reminiscent of the demo scene
- .kkrieger – a 96kb FPS made in 2004


https://commons.wikimedia.org/wiki/File:.kkrieger_1.jpg