Bremen

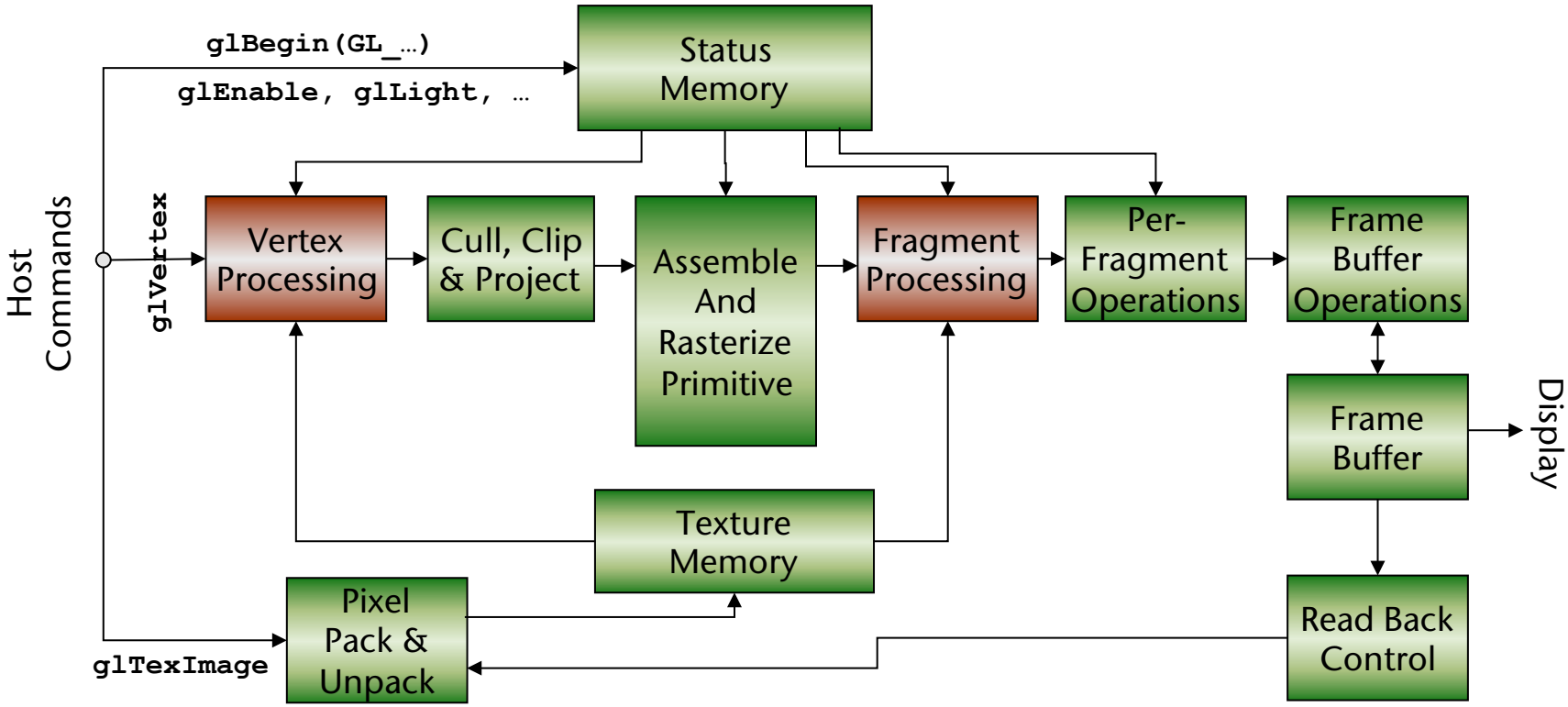
CG VR

Advanced Computer Graphics
Advanced Shader Programming

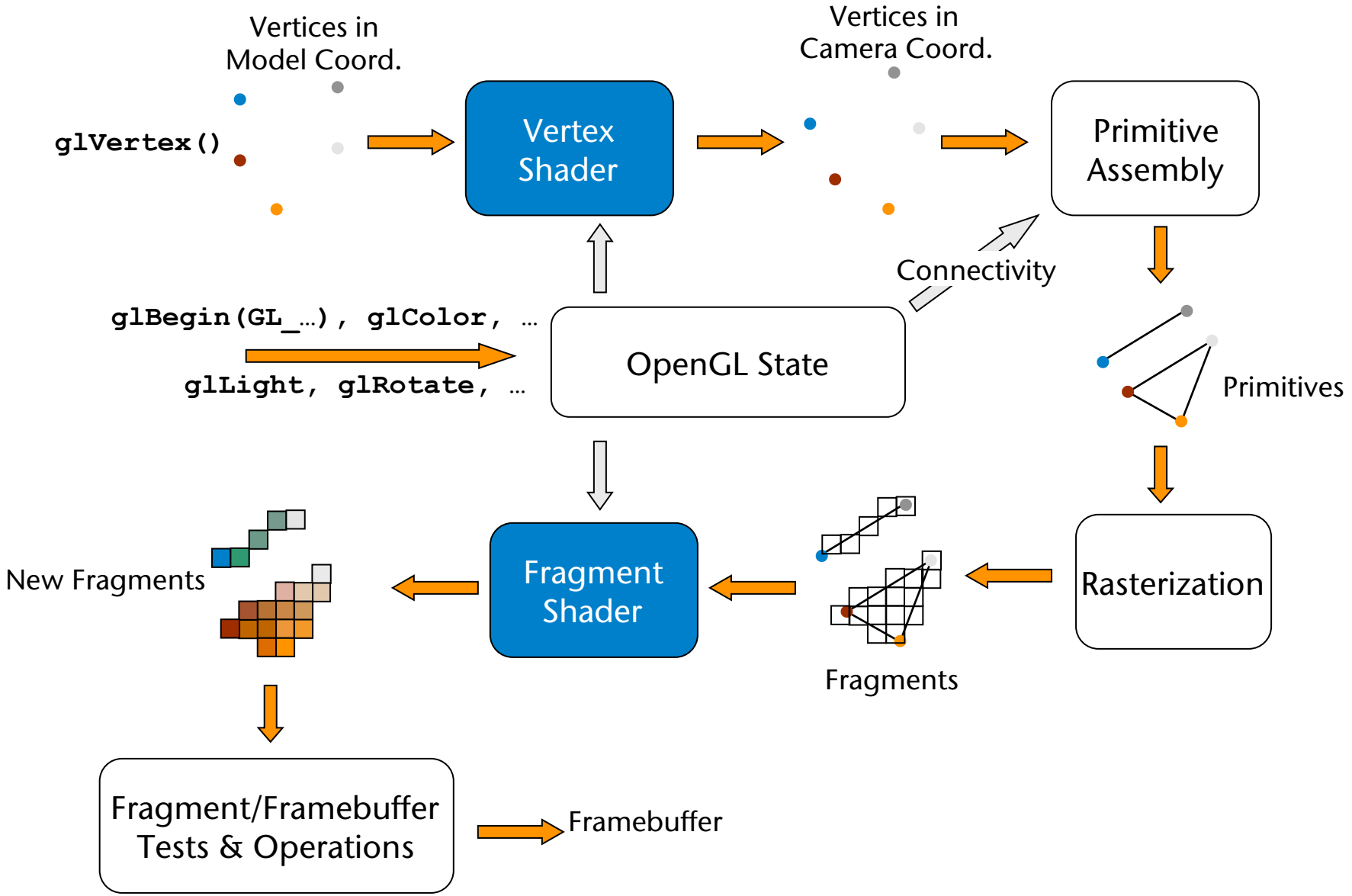
G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Recap

- Programmable *vertex und fragment processors*
  - Expose that which was already there anyway
- Texture memory = now general storage for any data



A More Abstract Overview of the Programmable Pipeline



More Versatile Texturing by Shader Programming

- Declare texture in the shader (vertex or fragment):

```
uniform sampler2D myTex;
```

- Load und bind texture in OpenGL-program as always:

```
glBindTexture( GL_TEXTURE_2D, myTexture );
glTexImage2D(...);
```

- Establish a connection between the two:

```
uint mytex = glGetUniformLocation( prog, "myTex" );
glUniform1i( mytex, 0 );  // 0 = texture unit, not ID
```

- Access in fragment shader:

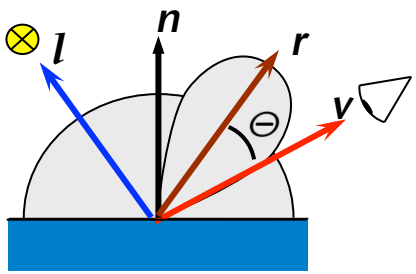
```
vec4 c = texture2D( myTex, gl_TexCoord[0].xy );
```

Example: A Simple "Gloss" Texture

- Idea: expand the conventional Phong lighting by introducing a *specular reflection coefficient* that is mapped from a texture on the surface

$$I_{\text{out}} = \left(r_d \cos \Phi + r_s \cos^p \Theta\right) \cdot I_{\text{in}}$$

$$r_s = r_s(u, v)$$

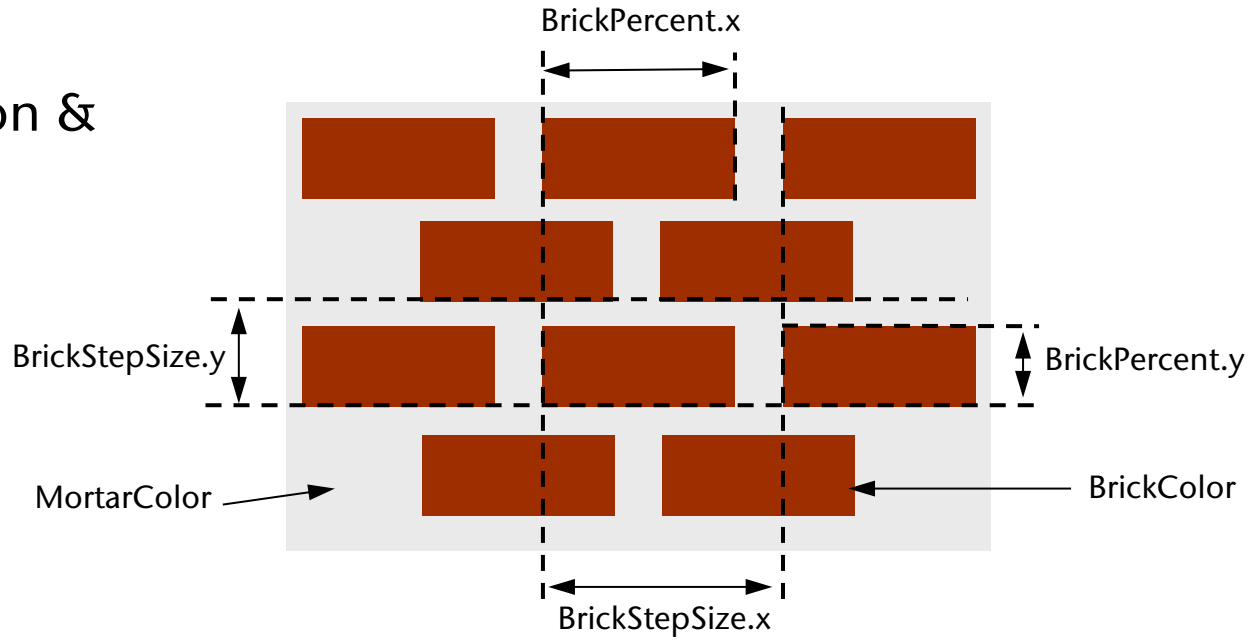


`demos/shader/vorlesung_demos/gloss.{frag,vert}`

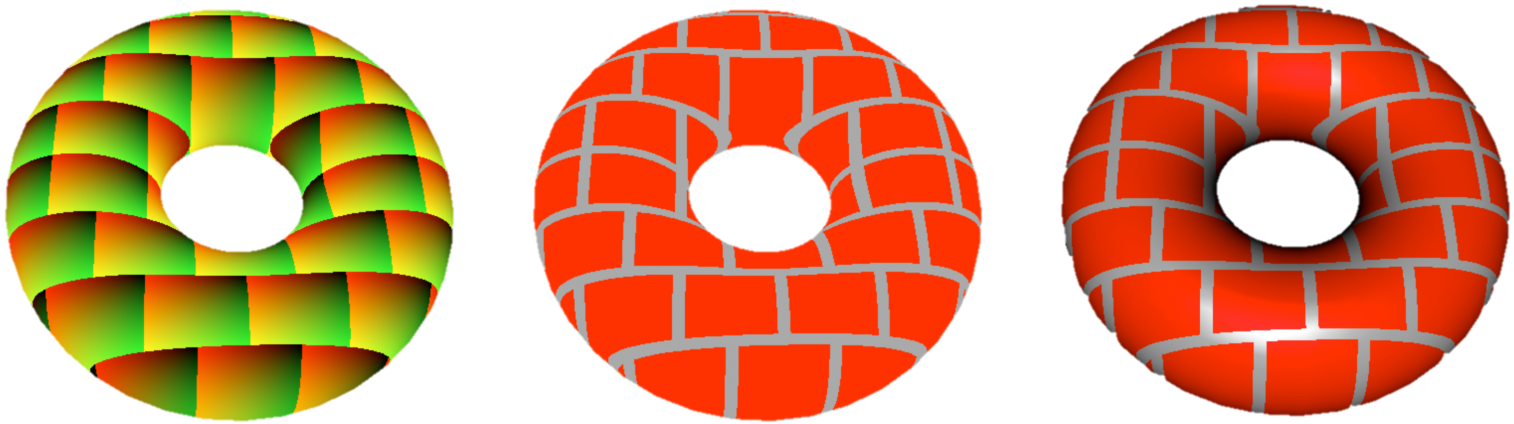
Procedural Textures Using Shader Programming

- Goal:
  Brick texture

- Simplification & parameters:



- General mechanics:
  - Vertex shader: normal lighting calculation
  - Fragment shader:
    - For each fragment, determine if the point lies in the brick or in the mortar on the basis of the x/y coordinates of the corresponding point in the object's space
    - After that, multiply the corresponding color with intensity from lighting model
- First 3 steps towards a complete shader program:



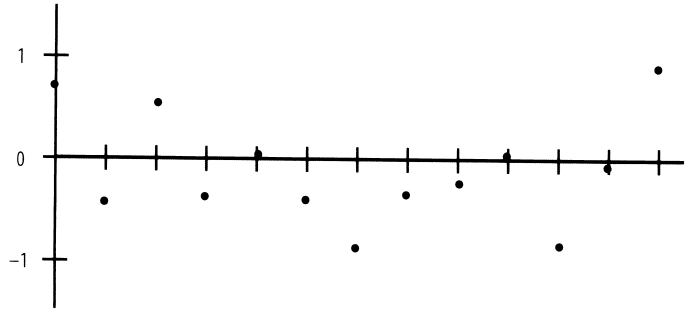
vorlesung_demos/brick.vert and brick[1-3].frag

Noise

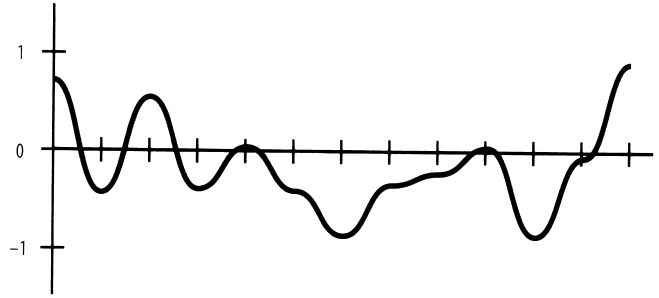
- Most procedural textures look too "clean"
- Idea: add all sorts of noise
  - Dirt, grime, random irregularities, etc., for a more realistic appearance
- Ideal qualities of a noise function:
  - At least C^2-continuous
  - It's sufficient if it looks random
  - No obvious patterns or repetitions
  - Repeatable (same output with the same input)
  - Convenient domain, e.g. [-1,1]
  - Can be defined for 1-4 dimensions
  - Isotropic (invariant under rotation)

- Simple idea, demonstrated by a 1-dimensional example:

  1. Choose random y-values from [-1,1] at the integer points:

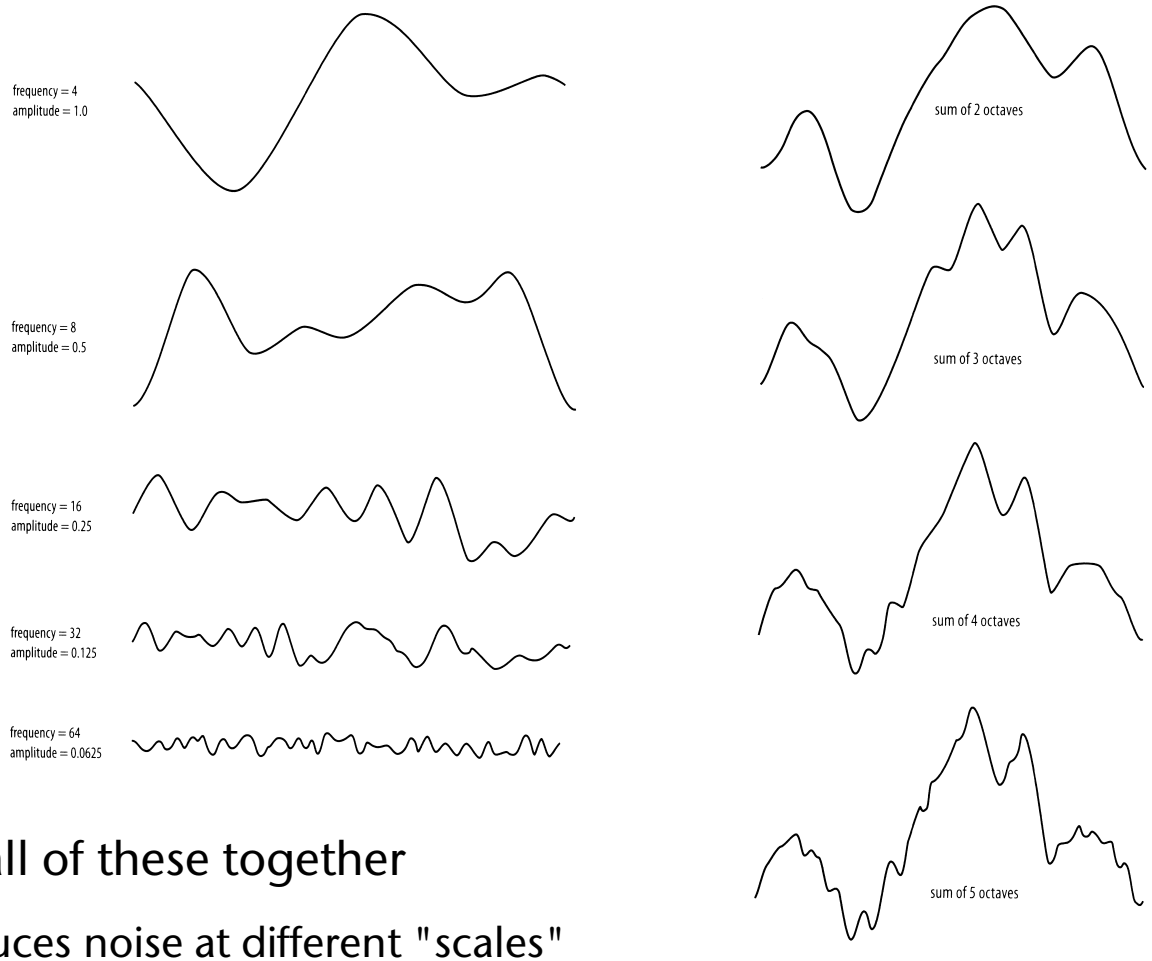


  2. Interpolate in between, e.g. cubically (linearly isn't sufficient):



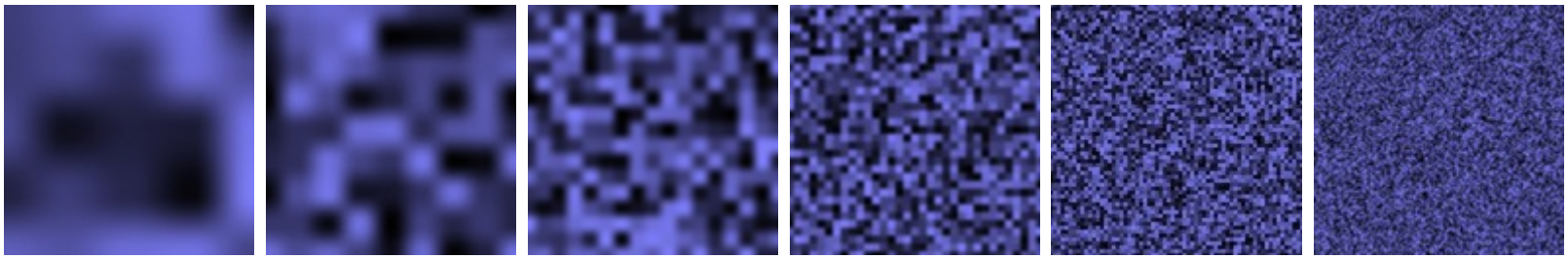
- This kind of noise function is called *"value noise"*

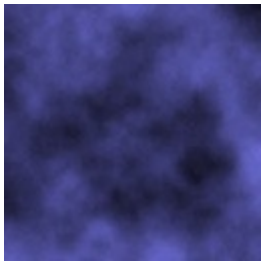
3. Generate multiple noise functions with different frequencies:



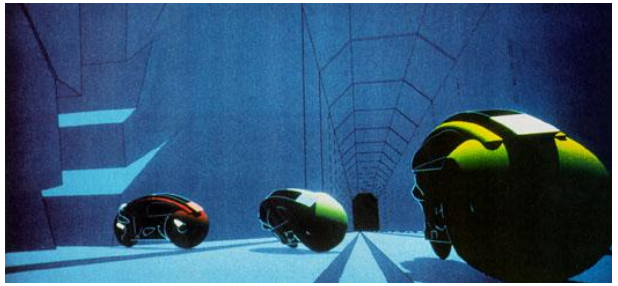
4. Add all of these together
   - Produces noise at different "scales"

- The same thing in 2D:





Result



- Easily allows itself to be generalized into higher dimensions
- Also called *Perlin noise, pink noise,* or *fractal noise*
  - Ken Perlin first dealt with this during his work on TRON