Noise!

- Can be 1D, 2D, or 3D
- Is a function of input value(s)
- Ranges from -1. to +1. or from 0. to 1.
- Might look random, but really isn’t
- Has continuity (i.e., if you change the input value to the noise function a little, the output value will only change a little)
- Is repeatable (i.e., if you supply the same inputs, the noise function will always give you back the same output)

Positional Noise

Idea: Pick a random number at the whole-number input values and then fit a piecewise smooth curve through those points.

Gradient Noise

Idea: Place points at the mid-line at the whole-number input values and use random numbers to pick gradients (slopes) there, and then fit a piecewise smooth curve through those points with those slopes.

Quintic (5th order) Interpolation Creates More Continuity Than Cubic

Cubic: \( C^{(1)} \) continuity at the whole-number values
Quintic: \( C^{(2)} \) continuity at the whole-number values

Coefficients for Cubic and Quintic Forms

Cubic
- \( C_{N0} = 1 - 3t^2 + 2t^3 \)
- \( C_{N1} = 3t^2 - 2t^3 = 1 - C_{N0} \)
- \( C_{G0} = t - 2t^2 + t^3 \)
- \( C_{G1} = -t^2 + t^3 \)
- \( C_{C0} = 0 \)
- \( C_{C1} = 0 \)

Quintic
- \( C_{X0} = 1 - 10t^2 + 15t^4 - 6t^5 \)
- \( C_{X1} = 10t^2 - 15t^4 + 6t^5 = 1 - C_{X0} \)
- \( C_{G0} = t - 6t^2 + 8t^3 - 3t^4 \)
- \( C_{G1} = -4t^2 + 7t^3 - 3t^4 \)
- \( C_{C0} = \frac{1}{2}t^2 - \frac{3}{2}t^3 + \frac{3}{4}t^4 - \frac{1}{2}t^5 \)
- \( C_{C1} = \frac{1}{2}t^2 - \frac{1}{2}t^3 + \frac{1}{2}t^4 \)
Noise Octaves

Idea: Add multiple noise waves, each one twice the frequency and half the amplitude of the previous one.

1 Octave  4 Octaves

Image Representation of 2D Noise

4 Octaves

1 Octave

3D Surface Representation of 2D Noise

3D Volume Rendering of 3D Noise

Low ------- Mid ------ High
Blue ------ Green ------ Red
Has continuity in X, Y, and Z.

Volume Isosurfaces of 3D Noise

S* = Mid-value

The low half of the noise values are on one side of the surface, the high half are on the other.

Examples

Color Blending for Marble
Color Blending for Clouds
Deciding when to Discard for Erosion
Idea: Take the absolute value of the noise about the centerline, giving the noise a “sharper” appearance and creating “creases”. **Warning:** this is not the same as fluid “turbulence.”

Normal

Turbulent

Remember Noise Octaves? What if we create a lookup table of noise octaves and hide it in a texture?

1 Octave 4 Octaves

A Noise Texture in Glman

The glman tool automatically creates a 3D noise texture and places it into Texture Unit 3. Your shaders can access it through the pre-created uniform variable called *Noise3*. You just declare it in your shader as:

```glsl
texture3D Noise3;
```

The “noise vector” texture *nv* is a vec4 whose components have separate meanings. The *r* component is the low frequency noise. The *g* component is twice the frequency and half the amplitude of the *r* component, and so on for the *b* and *a* components. Each component is centered around the middle value of 0.5.

<table>
<thead>
<tr>
<th>Component</th>
<th>Term</th>
<th>Term Range</th>
<th>Term Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><em>r</em></td>
<td>0.5 ± .5000</td>
<td>0.0000 → 1.0000</td>
</tr>
<tr>
<td>1</td>
<td><em>g</em></td>
<td>0.5 ± .2500</td>
<td>0.2500 → 0.7500</td>
</tr>
<tr>
<td>2</td>
<td><em>b</em></td>
<td>0.5 ± .1250</td>
<td>0.3750 → 0.6250</td>
</tr>
<tr>
<td>3</td>
<td><em>a</em></td>
<td>0.5 ± .0625</td>
<td>0.4375 → 0.5625</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td>2.0 ± ~ 1.0</td>
<td>~1.0 → 3.0</td>
</tr>
<tr>
<td>(sum – 2)</td>
<td></td>
<td>0.0 ± ~ 1.0</td>
<td>~1.0 → 1.0</td>
</tr>
</tbody>
</table>

The first time glman runs, it creates a 3D noise texture for you, it will take a few seconds. But, glman then writes it to a local file, so that the next time this 3D texture is needed, it is read from the file, which is a lot faster.

A Noise Texture in Glman

So, if you would like to have a four-octave noise function that ranges from 0 to 1, then do this:

```
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. -> 3.
        n = ( n - 1. ) / 2.; // range is now 0. -> 1.
```

If you would like to have a four-octave noise function that ranges from -1 to 1, then do this instead:

```
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. -> 3.
        n = ( n - 2. ); // range is now -1. -> 1.
```

By default, the glman 3D noise texture has dimensions 64 × 64 × 64. You can change this by putting a command in your GLIB file of the form

```glsl
Noise3D 128
```

A Noise Texture in Glman

To get dimension 128 × 128 × 128, or choose whatever resolution you want (up to around 400 × 400 × 400).
A Noise Texture in Your C/C++ Program

The easiest way to read a noise texture into your C/C++ program is to get one of the noise textures from glman and know how to read it in. These pages will tell you how:

```c
// in InitGraphics:
glGenTextures(1, &TexName);
int nums, numt, nump;
unsigned char * texture = 
ReadTexture3D("noise3d.064.tex", &nums, &numt, &nump);
If( texture == NULL ) { … }
```

```c
void Display()
{
    …
    glActiveTexture(GL_TEXTURE0 + 3); // set to use texture unit 3
    glBindTexture(GL_TEXTURE_3D, TexName);
    Pattern->Use();
    Pattern->SetUniformVariable("uTexUnit", 3);
    …
    << Draw something >>
    Pattern->Use(0);
}
```

A Noise Texture in Your C++ Program

```c
float n = NoiseMag * noise( NoiseFreq * PP );
float t = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 color = mix( ORANGE, WHITE, t );
```

### How to Use Noise

Have an equation that relates some input value (x, y, z or s, t) to output values (color, height)

Add Noise to the actual input values to produce new "fake" input values

Have actual input values of where we are right now

How much to amplify the noise effect

Coordinates where you are now

How much to increase the sampling rate

Now add the noise value, N, to the actual location. Compute the effect at that new location, but apply it at the actual location.

```
vec4 nv = texture( Noise3, uNoiseFreq * vMCposition );
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. -> 3.
    n = ( n - 1. ) / 2.; // range is now 0. -> 1.
    n *= uNoiseMag;
```

### Elliptical Dots with Tolerance

```
float d = ( x - x0 )^2 / ( x0 ).^2 + ( y - y0 )^2 / ( y0 ).^2 ≤ 1 - uTol

float d = ( x - x0 )^2 / ( x0 ).^2 + ( y - y0 )^2 / ( y0 ).^2 ≤ 1 + uTol
```

vec3 color = mix( ORANGE, WHITE, d );
float n = nv.r + nv.g + nv.b + nv.a; // 1. -> 3.
    n = ( n - 2. ); // -1. -> 1.
    n *= uNoiseAmp;
    . . .
float ds = st.s - sc; // wrt ellipse center
float dt = st.t - tc; // wrt ellipse center
float oldDist = sqrt( ds*ds + dt*dt );
float newDist = oldDist + n;
float scale = newDist / oldDist; // this could be < 1., = 1., or > 1.
ds *= scale; // scale by noise factor
dt *= scale; // scale by noise factor
ds /= Ar; // ellipse equation
dt /= Br; // ellipse equation
float d = ds*ds + dt*dt;
float t = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 theColor = mix( ORANGE, WHITE, t );
. . .
uNoiseAmp = 0.
uNoiseAmp > 0.

N = NoiseMag * noise( NoiseFreq * PP );

N = NoiseMag * noise( NoiseFreq * PP );
If you didn’t have the labels, could you tell which of these two images is displacement-mapped and which is bump-mapped?