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 Coherency (i.e., if you change the input value to the noise function a little, the output value will only change a little)
- Has Repeatability (i.e., if you supply the same inputs, the noise function will always give you back the same output)
- Is Continuous (i.e., it’s smooth with no jarring jumps)

Positional Noise

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

No matter what, you will get a very good plus-or-minus distribution.

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

Coefficients for Cubic and Quintic Forms

<table>
<thead>
<tr>
<th>Noise values</th>
<th>Gradients</th>
<th>Curvatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cubic</td>
<td>Quintic</td>
<td></td>
</tr>
<tr>
<td>$C_{x0} = 1 - 3t^2 + 2t^3$</td>
<td>$C_{x0} = 1 - 10t^5 + 15t^4 - 6t^3$</td>
<td></td>
</tr>
<tr>
<td>$C_{x1} = 3t^2 - 2t^3 = 1 - C_{x0}$</td>
<td>$C_{x1} = 10t^4 - 15t^3 + 6t^2 = 1 - C_{x0}$</td>
<td></td>
</tr>
<tr>
<td>$C_{g0} = t - 2t^2 + t^3$</td>
<td>$C_{g0} = t - 6t^2 + 8t^4 - 3t^3$</td>
<td></td>
</tr>
<tr>
<td>$C_{g1} = -t^2 + t^3$</td>
<td>$C_{g1} = -4t^3 + 7t^4 - 3t^5$</td>
<td></td>
</tr>
<tr>
<td>$C_{c0} = 0$</td>
<td>$C_{c0} = \frac{1}{2}t^2 - \frac{3}{2}t^3 + \frac{3}{2}t^4 - \frac{1}{2}t^5$</td>
<td></td>
</tr>
<tr>
<td>$C_{c1} = 0$</td>
<td>$C_{c1} = \frac{1}{2}t^4 - t^5 + \frac{1}{2}t^6$</td>
<td></td>
</tr>
</tbody>
</table>
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

1 Octave

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

S* = Mid-value

4 Octaves

Examples

Color Blending for Marble
Color Blending for Clouds

Deciding when to Discard for Erosion
### Turbulence

**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 use of the term as fluid "turbulence".

#### Turbulence Example

<table>
<thead>
<tr>
<th>Normal</th>
<th>Turbulent</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Normal" /></td>
<td><img src="image" alt="Turbulent" /></td>
</tr>
</tbody>
</table>

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

<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>(nv.r)</td>
<td>0.5 ± .5000</td>
<td>0.0000 → 1.0000</td>
</tr>
<tr>
<td>1</td>
<td>(nv.g)</td>
<td>0.5 ± .2500</td>
<td>0.2500 → 0.7500</td>
</tr>
<tr>
<td>2</td>
<td>(nv.b)</td>
<td>0.5 ± .1250</td>
<td>0.3750 → 0.6250</td>
</tr>
<tr>
<td>3</td>
<td>(nv.a)</td>
<td>0.5 ± .0625</td>
<td>0.4375 → 0.5625</td>
</tr>
<tr>
<td>sum</td>
<td>(2.0 ± )</td>
<td></td>
<td>1.0 → 3.0</td>
</tr>
<tr>
<td>((\text{sum} - 1) / 2)</td>
<td>(1.0 ± )</td>
<td></td>
<td>0.0 → 2.0</td>
</tr>
<tr>
<td>((\text{sum} - 2))</td>
<td>(0.0 ± )</td>
<td></td>
<td>-1.0 → 1.0</td>
</tr>
</tbody>
</table>

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
uniform sampler3D Noise3;
...
vec4 nv = texture( Noise3, uNoiseFreq * vMCposition );
```

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.

#### A Noise Texture in Glman

The `glman` tool method 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
uniform sampler3D Noise3;
vec4 nv = texture( Noise3, uNoiseFreq * vMCposition );
```

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.

#### A Noise Texture in Glman

The first time `glman` runs, it creates 2D and 3D noise textures for you, it will take a few seconds. But, `glman` then writes them to a local file, so that the next time this 2D or 3D texture is needed, it is read from the file, which is a lot faster.

A 2D noise texture works the same way as a 3D noise texture, except you get at it with:

```glsl
uniform sampler2D Noise2;
vec4 nv = texture( Noise2, uNoiseFreq * vST );
```

The only difference is that a 2D noise texture is indexed by a `vec2` (such as the s-t coordinates) while the 3D noise texture is indexed by a `vec3` (such as the model x-y-z coordinates). But, both return a `vec4`. 

### Noise3D 128

to get dimension 128 × 128 × 128, or choose whatever resolution you want (up to around 400 × 400 × 400).
A 2D 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.

```cpp
GLuint TexName; // a global

// in InitGraphics:
glGenTextures(1, &TexName);

int nums, numt;
unsigned char * texture = ReadTexture2D( "noise2d.064.tex", &nums, &numt);
If( texture == NULL ) { … }

glBindTexture(GL_TEXTURE_2D, TexName);

Pattern = new GLSLProgram();
bool valid = Pattern->Create( "pattern.vert", "pattern.frag" );
If ( !valid )
```

A 3D 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.

```cpp
GLuint TexName; // a global

// in InitGraphics:
glGenTextures(1, &TexName);

int nums, numt, nump;
unsigned char * texture = ReadTexture3D( "noise3d.064.tex", &nums, &numt, &nump);
If( texture == NULL ) { … }

Pattern = new GLSLProgram();
bool valid = Pattern->Create( "pattern.vert", "pattern.frag" );
If ( !valid )
```

A Noise Texture in Your C++ Program

```cpp
void Display()
{
  ... 
  glBindTexture(GL_TEXTURE_3D, TexName);
  Pattern->Use();
  Pattern->SetUniformVariable( "uTexUnit", 3 );
  ... 
  << Draw something >>
  ... 
  Pattern->Use(0);
}
```

How to Use Noise

Idea: The graphics system will display "here", using display parameters as if you were "over there".

1. Have actual input values of where we are right now
2. Add Noise to the actual input values to produce new "fake" input values
3. Have an equation that relates some input value (x,y,z or s,t) to output values (color, height)
4. Use those new "fake" input values in the original equation
How to Use Noise

Coordinates where this fragment is

How much to magnify the noise effect

How much to increase the sampling rate

In the vertex shader:

```glsl
out vec3 vMCposition;

vMCposition = gl_Vertex.xyz;
```

How to Use Noise

Coordinates where this fragment is

How much to magnify the noise effect

How much to increase the sampling rate

In the fragment shader:

```glsl
uniform float uNoiseFreq, uNoiseMag;
in vec3 vMCposition;

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

out vec3 vMCposition;
```

Elliptical Dots with Tolerance

Add noise to the actual (s,t) location. Compute the
effect at that “fake” location, but apply it at the actual location.

`1 - uTol \leq \left( \frac{s-c}{A_s} \right)^2 + \left( \frac{t-c}{B_t} \right)^2 \leq 1 + uTol`

```glsl
float d = \frac{(s-c)^2}{A_s^2} + \frac{(t-c)^2}{B_t^2};
float t = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 color = mix( ORANGE, WHITE, t );
```

Elliptical Dots with Tolerance and Noise

Compute the

effect at that “fake” location, but apply it at the actual location.

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

Elliptical Dots with Tolerance and Noise

Compute the

effect at that “fake” location, but apply it at the actual location.

```glsl
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
ds /= Ar; // ellipse equation
dt *= scale; // scale by noise factor
dt /= Br; // ellipse equation
float d = ds*ds + dt*dt;
float t = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 theColor = mix( ORANGE, WHITE, t );
```

Color Only
If You Didn’t Have the Labels, Could You Tell Which of These Two Images is Displacement-Mapped and Which is Bump-Mapped?