Noise!

- Can be 1D, 2D, or 3D
- Is a function of input value(s)
- Ranges from -1.0 to +1.0, or from 0.0 to 1.0.
- 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.

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: $C^0$ continuity at the whole-number values
Quintic: $C^2$ continuity at the whole-number values

Coefficients for Cubic and Quintic Forms

<table>
<thead>
<tr>
<th>Cubic</th>
<th>Quintic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{N0} = 1 - 3t^2 + 2t^3$</td>
<td>$C_{N0} = 1 - 10t^2 + 15t^4 - 6t^5$</td>
</tr>
<tr>
<td>$C_{N1} = 3t^2 - 2t^3 = 1 - C_{N0}$</td>
<td>$C_{N1} = 10t^2 - 15t^4 + 6t^5 = 1 - C_{N0}$</td>
</tr>
<tr>
<td>$C_{G0} = t - 2t^2 + t^3$</td>
<td>$C_{G0} = t - 6t^2 + 8t^4 - 3t^5$</td>
</tr>
<tr>
<td>$C_{G1} = -t^2 + t^3$</td>
<td>$C_{G1} = -4t^2 + 7t^4 - 3t^5$</td>
</tr>
<tr>
<td>$C_{G2} = 0$</td>
<td>$C_{G2} = 1$</td>
</tr>
<tr>
<td>$C_{G3} = 0$</td>
<td>$C_{G3} = \frac{1}{2} - \frac{1}{2}t^2 + \frac{3}{2}t^4 - \frac{1}{2}t^5$</td>
</tr>
<tr>
<td>$C_{G4} = 0$</td>
<td>$C_{G4} = \frac{1}{2} - \frac{1}{2}t^2 + \frac{3}{2}t^4$</td>
</tr>
</tbody>
</table>

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

1 Octave

3D Surface Representation of 2D Noise

3D Volume Rendering of 3D Noise

1 Octave

Has continuity in X, Y, and Z.

Volume Isosurfaces of 3D Noise

1 Octave 4 Octaves

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
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 as fluid “turbulence.”

Turbulence Example

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

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:

\[
\text{float } n = \text{nv.r} + \text{nv.g} + \text{nv.b} + \text{nv.a}; \\
n = (n - 1.1) / 2.; \\
\text{if range is 1. } \rightarrow 3, \\
\text{if range is now 0. } \rightarrow 1.
\]

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

\[
\text{float } n = \text{nv.r} + \text{nv.g} + \text{nv.b} + \text{nv.a}; \\
n = (n - 1.2); \\
\text{if range is 1. } \rightarrow 3, \\
\text{if range is now -1. } \rightarrow 1.
\]

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

Noise3D 128

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 ) { … }
glBindTexture(GL_TEXTURE_3D, TexName);
glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);
glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA8, nums, numt, nump, 0, GL_RGBA,
GL_UNSIGNED_BYTE, texture);
Pattern = new GLSLProgram();
if(!valid) . . .
```

A Noise Texture in Your C/C++ Program

unsigned char *
ReadTexture3D(char *filename, int *width, int *height, int *depth)
{
FILE *fp = fopen(filename, "rb");
if( fp == NULL )
return NULL;
int nums, numt, nump;
fread(&nums, 4, 1, fp);
fread(&numt, 4, 1, fp);
fread(&nump, 4, 1, fp);
*width = nums;
*height = numt;
*depth = nump;
unsigned char * texture = new unsigned char [4 * nums * numt * nump];
fread(texture, 4 * nums * numt * nump, 1, fp);
fclose(fp);
return texture;
}

A Noise Texture in Your C++ Program

void
Display()
{
. . .
gActiveTexture(GL_TEXTURE3); // set to use texture unit 3
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”.

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

1. Coordinates where you are now
2. Noise frequency
3. How much to amplify the noise effect

float n = uNoiseMag * noise( uNoiseFreq * vMCposition );
vec4 nv = texture( Noise3, uNoiseFreq * vMCposition );
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. -> 3.
float d = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 color = mix( ORANGE, WHITE, t );

Elliptical Dots with Tolerance

1. Elliptical dots
2. Tolerance
3. Why would we typically use Model coordinates instead of World coordinates?
4. Why would we typically use Model coordinates instead of World coordinates?

Elliptical Dots with Tolerance

float t = smoothstep( 1.-uTol, 1.+uTol, d );
vec3 color = mix( ORANGE, WHITE, t );
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
    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 );
.
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?