Noise !

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
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)
Idea: Pick a random number at the whole-number input values and then fit a piecewise smooth curve through those points.

The problem is that, due to the uncertainty of random numbers, you might get a good plus-or-minus distribution, or a not-so-good distribution.
**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.

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

\[ N(t) = C_{N0} N_0 + C_{N1} N_1 + C_{G0} G_0 + C_{G1} G_1 + C_{C0} C_0 + C_{C1} C_1 \]

**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**

- \[ C_{N0} = 1 - 10t^3 + 15t^4 - 6t^5 \]
- \[ C_{N1} = 10t^3 - 15t^4 + 6t^5 = 1 - C_{N0} \]
- \[ C_{G0} = t - 6t^3 + 8t^4 - 3t^5 \]
- \[ C_{G1} = -4t^3 + 7t^4 - 3t^5 \]
- \[ C_{C0} = \frac{1}{2}t^2 - \frac{3}{2}t^3 + \frac{3}{2}t^4 - \frac{1}{2}t^5 \]
- \[ C_{C1} = \frac{1}{2}t^3 - t^4 + \frac{1}{2}t^5 \]
**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

4 Octaves
3D Surface Representation of 2D Noise

4 Octaves
3D Volume Rendering of 3D Noise

1 Octave

Has continuity in X, Y, and Z
Volume Isosurfaces of 3D Noise

1 Octave

S* = Mid-value

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

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
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:

```cpp
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 .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>nv.r</em></td>
<td>0.5 ± .5000</td>
<td>0.0000 → 1.0000</td>
</tr>
<tr>
<td>1</td>
<td><em>nv.g</em></td>
<td>0.5 ± .2500</td>
<td>0.2500 → 0.7500</td>
</tr>
<tr>
<td>2</td>
<td><em>nv.b</em></td>
<td>0.5 ± .1250</td>
<td>0.3750 → 0.6250</td>
</tr>
<tr>
<td>3</td>
<td><em>nv.a</em></td>
<td>0.5 ± .0625</td>
<td>0.4375 → 0.5625</td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td>2.0 ± ~ 1.0</td>
<td>~1.0 → 3.0</td>
</tr>
<tr>
<td></td>
<td>sum – 1</td>
<td>1.0 ± ~ 1.0</td>
<td>~0.0 → 2.0</td>
</tr>
<tr>
<td></td>
<td>(sum – 1) / 2</td>
<td>0.5 ± ~ 0.5</td>
<td>~0.0 → 1.0</td>
</tr>
<tr>
<td></td>
<td>(sum – 2)</td>
<td>0.0 ± ~ 1.0</td>
<td>~1.0 → 1.0</td>
</tr>
</tbody>
</table>
So, if you would like to have a four-octave noise function that ranges from 0. to 1, then do this:

```c
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:

```c
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 $64 \times 64 \times 64$. You can change this by putting a command in your GLIB file of the form

**Noise3D 128**

... to get dimension $128 \times 128 \times 128$, or choose whatever resolution you want (up to around $400 \times 400 \times 400$).
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 2D noise texture works the same way, except you get at it with:

```glsl
uniform sampler2D Noise2;
...
vec4 nv = texture( Noise2, uNoiseFreq * vST );
float n = nv.r + nv.g + nv.b + nv.a;       // range is 1. -> 3.
n = ( n - 1. ) / 2.;                       // range is now 0. -> 1.
```

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*. 
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.

```
// 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();
bool valid = Pattern->Create( "pattern.vert", "pattern.frag" );
if (!valid)
... 
```
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

```cpp
void Display() {
    ...
    glEnableTexture(GL_TEXTURE3); // set to use texture unit 3
    glBindTexture(GL_TEXTURE_3D, TexName);
    Pattern->Use();
    Pattern->SetUniformVariable("uTexUnit", 3);
    ...   <<< Draw something >>
    Pattern->Use(0);
}
```
How to Use Noise

Have actual input values of where we are right now

Add Noise to the actual input values to produce new “fake” input values

Use those new “fake” input values in the original equation

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

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

Coordinates where you are now

How much to amplify the noise effect

\[
\text{float } n = \text{uNoiseMag} \times \text{noise}( \text{uNoiseFreq} \times \text{vMCposition} );
\]

Noise frequency

Why would we typically use Model coordinates instead of World coordinates?

Now add the noise value, \( n \), to the actual location. Compute the effect at that new location, but apply it at the actual location.

\[
\text{vec4 } \text{nv} = \text{texture}( \text{Noise3}, \text{uNoiseFreq} \times \text{vMCposition} );
\]

\[
\begin{align*}
\text{float } n &= \text{nv.r} + \text{nv.g} + \text{nv.b} + \text{nv.a}; & \text{// range is 1. -> 3.} \\
n &= (n - 1.)/2.; & \text{// range is now 0. -> 1.} \\
n &= \times \text{uNoiseMag};
\end{align*}
\]
Elliptical Dots with Tolerance

$$1 - uTol \leq \left( \frac{s-s_c}{A_r} \right)^2 + \left( \frac{t-t_c}{B_r} \right)^2 \leq 1 + uTol$$

```c
float t = smoothstep(1.-uTol, 1.+uTol, d);
vec3 color = mix(ORANGE, WHITE, t);
```
Elliptical Dots with Tolerance and Noise

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 );
    ...
Elliptical Dots with Tolerance and Noise
\[ N = \text{NoiseMag} \times \text{noise}(\text{NoiseFreq} \times \text{PP}); \]
Displacement Only
Color and Displacement together
Displacement Only

Surface Only

Surface + Displacement

No Noise

Noise

Or

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