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.

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

Cubic

\[ N(t) = C^0_N N_0 + C^1_N N_1 + C^0_G G_0 + C^1_G G_1 + C^0_C C_0 + C^1_C C_1 \]

\[ \begin{align*}
C^0_N &= 1 - 3t^2 + 2t^3 \\
C^1_N &= 3t^2 - 2t^3 = 1 - C^0_N \\
C^0_G &= t - 2t^2 + t^3 \\
C^1_G &= -t^2 + t^3 \\
C^0_C &= 0 \\
C^1_C &= 0
\end{align*} \]

Quintic

\[ \begin{align*}
C^0_N &= 1 - 10t^3 + 15t^4 - 6t^5 \\
C^1_N &= 10t^3 - 15t^4 + 6t^5 = 1 - C^0_N \\
C^0_G &= t - 6t^3 + 8t^4 - 3t^5 \\
C^1_G &= -4t^3 + 7t^4 - 3t^5 \\
C^0_C &= \frac{1}{2} t^2 - \frac{3}{2} t^3 + \frac{3}{2} t^4 - \frac{1}{2} t^5 \\
C^1_C &= \frac{1}{2} t^3 - \frac{3}{2} t^4 + \frac{1}{2} t^5
\end{align*} \]
Noise Octaves

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

Image Representation of 2D Noise

1 Octave

4 Octaves
3D Surface Representation of 2D Noise

1 Octave

Has continuity in X, Y, and Z

3D Volume Rendering of 3D Noise

Low ------- Mid ------ High
Blue ------ Green ------ Red

1 Octave
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 use of the term 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

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:

```
uniform sampler3D Noise3;
```

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

The “noise vector” texture \( nv \) is a \( \text{vec4} \) 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>( nv.r )</td>
<td>( 0.5 \pm 0.5000 )</td>
<td>( 0.0000 \rightarrow 1.0000 )</td>
</tr>
<tr>
<td>1</td>
<td>( nv.g )</td>
<td>( 0.5 \pm 0.2500 )</td>
<td>( 0.2500 \rightarrow 0.7500 )</td>
</tr>
<tr>
<td>2</td>
<td>( nv.b )</td>
<td>( 0.5 \pm 0.1250 )</td>
<td>( 0.3750 \rightarrow 0.6250 )</td>
</tr>
<tr>
<td>3</td>
<td>( nv.a )</td>
<td>( 0.5 \pm 0.0625 )</td>
<td>( 0.4375 \rightarrow 0.5625 )</td>
</tr>
<tr>
<td>sum</td>
<td>( 2.0 \pm 1.0 )</td>
<td>( -1.0 \rightarrow 3.0 )</td>
<td></td>
</tr>
<tr>
<td>sum – 1</td>
<td>( 1.0 \pm 1.0 )</td>
<td>( -0.0 \rightarrow 2.0 )</td>
<td></td>
</tr>
<tr>
<td>(sum – 1) / 2</td>
<td>( 0.5 \pm 0.5 )</td>
<td>( -0.0 \rightarrow 1.0 )</td>
<td></td>
</tr>
<tr>
<td>(sum – 2)</td>
<td>( 0.0 \pm 1.0 )</td>
<td>( -1.0 \rightarrow 1.0 )</td>
<td></td>
</tr>
</tbody>
</table>
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:

```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).

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:

```c
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 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);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, nums, numt, 0, GL_RGBA,
             GL_UNSIGNED_BYTE, texture);

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

A 2D Noise Texture in Your C/C++ Program

```cpp
unsigned char *
ReadTexture2D( char *filename, int *width, int *height )
{
    FILE *fp = fopen(filename, "rb");
    if( fp == NULL )
        return NULL;

    int nums, numt;
    fread(&nums, 4, 1, fp);
    fread(&numt, 4, 1, fp);
    fprintf( stderr, "Texture size = %d x %d\n", nums, numt );

    *width = nums;
    *height = numt;

    unsigned char * texture = new unsigned char[ 4 * nums * numt ];
    fread(texture, 4 * nums * numt, 1, fp);
    fclose(fp);
    return texture;
}
```
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.

```c
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 ) { … }

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);
glTexParameteri(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_RGBA, nums, numt, nump, 0, GL_RGBA,
GL_UNSIGNED_BYTE, texture);

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

A 3D Noise Texture in Your C/C++ Program

```c
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);
    fprintf( stderr, "Texture size = %d x %d x %d\n", nums, numt, nump);

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

How to Use Noise

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

In the vertex shader:

```glsl
out vec3 vMCposition;

vMCposition = gl_Vertex.xyz;
```

How much to magnify the noise effect
How much to increase the sampling rate
Coordinates where this fragment is

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;
```

Now add the noise value, \( n \), to the actual \((s, t)\) location. Compute the effect at that “fake” location, but apply it at the actual location.

We typically do this in Model coordinates so that the pattern sticks to the object.

Elliptical Dots with Tolerance

```glsl
1 + uTol
1 - uTol

float \( d \) = \[ \frac{s - s_c}{A_r} \]^2 + \left[ \frac{t - t_c}{B_r} \right]^2
```

```glsl
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 );
    
    float uNoiseAmp = 0.
    if ( uNoiseAmp > 0. ) ...
\[ N = \text{NoiseMag} \times \text{noise}(\text{NoiseFreq} \times PP); \]
Displacement Only

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