A Problem

One of the early criticisms of Computer Graphics is that it was too good, that is, everything was too perfect. Spheres were too perfectly round. And so on. Computer Graphics needed a way to add imperfections. It seemed like random numbers could be used here. But pure random numbers are rather jarring:

and that's not what we want. What we want is randomness, but controlled randomness. In Computer Graphics, this became known as Noise.

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

- $C_{N0} = 1 - 3t^2 + 2t^3$
- $C_{N1} = 3t^2 - 2t^3 = 1 - C_{N0}$
- $C_{G0} = t - 2t^2 + t^3$
- $C_{G1} = -t^2 + t^3$
- $C_{C0} = 0$
- $C_{C1} = 0$

Quintic

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

---

**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

4 Octaves

3D Surface Representation of 2D Noise

4 Octaves

3D Volume Rendering of 3D Noise

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

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

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
uniform sampler3D Noise3;
```

```glsl
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>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>
</tbody>
</table>

- `sum`: 2.0 ± 1.0
- `(sum - 1) / 2`: 0.5 ± 0.5
- `(sum - 2)`: 0.0 ± 1.0

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
GLuint Noise3; // a global
GLSLProgram Pattern; // a global

// in InitGraphics:
glGenTextures(1, &Noise3);
int nums, numt, nump;
unsigned char * texture = ReadTexture3D( "noise3d.064.tex", &nums, &numt, &nump);
if( texture == NULL ) { … }

glBindTexture(GL_TEXTURE_3D, Noise3);
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_RGBA, nums, numt, nump, 0, GL_RGBA, GL_UNSIGNED_BYTE, texture);
```

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

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

```glsl
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. → 3.
```

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

```glsl
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. → 3.
```

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).

Getting a noise value from a 2D quantity (such as vST) works the same way as a 3D noise texture, except you get at it with:

```glsl
uniform sampler3D Noise3;
```
unsigned char *
ReadTexture3D (char *filename, int *width, int *height, int *depth)
{
    FILE *fp = fopen(filename, "rb");
    if (fp == NULL)
        return NULL;
    /*
     * Get texture size
     */
    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;
}

A Noise Texture in Your C++ Program

void
Display()
{
    ...
    glActiveTexture(GL_TEXTURE_3);
    // set to use texture unit 3
    glBindTexture(GL_TEXTURE_3D, Noise3);
    Pattern.Use();
    Pattern.SetUniformVariable("Noise3", 3);
    ...
    // Draw something
    ...
    Pattern.UnUse;
}

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

Use those new "fake" input values in the original equation

How much to amplify the noise effect

How much to increase the sampling rate

Model coordinates where this fragment is

In the vertex shader:

out vec3 vMCposition;
    vMCposition = gl_Vertex.xyz;

How to Index Noise from 3D Model Coordinates

In the fragment shader:

uniform sampler3D Noise3;
uniform float uNoiseFreq, uNoiseAmp;
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 *= uNoiseAmp;

    out vec3 vMCposition;
        vMCposition = gl_Vertex.xyz;

Now add the noise value, n, to the actual 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.
How to Index Noise from 2D Texture Coordinates

In the vertex shader:

```glsl
out vec2 vST;
... vST = gl_MultiTexCoord0.st;
```

How much to amplify the noise effect

How much to increase the sampling rate

In the fragment shader:

```glsl
uniform sampler3D Noise3;
uniform float uNoiseFreq, uNoiseAmp;
in vec2 vST;
...
vec4 nv = texture( Noise3, uNoiseFreq * vec3( vST, 0. ) );
float n = nv.r + nv.g + nv.b + nv.a; // range is 1. -> 3.
n = n - 2.; // range is now -1. -> 1.
n *= uNoiseAmp;
out vec2 vST;
... vST = gl_MultiTexCoord0.st;
```

Texture coordinates where this fragment is

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

Elliptical Dots with Tolerance and Noise

Now add the noise value, n, to the actual location. Compute the effect at that "fake" location but apply it at the actual location.

In the vertex shader:

```glsl
uNoiseAmp = 0.
```

In the fragment shader:

```glsl
uNoiseAmp > 0.
```

Elliptical Dots with Tolerance

```
1 - uTol ≤ \left( \frac{s - s_c}{A_c} \right)^2 + \left( l - l_c \right)^2 ≤ 1 + uTol

float d = \left( \frac{s - s_c}{A_c} \right)^2 + \left( l - l_c \right)^2

float t = smoothstep( 1-uTol, 1+uTol, d );
vec3 color = mix( ORANGE, WHITE, t );
```
\[ N = \text{NoiseAmp} \times \text{noise(NoiseFreq \times PP)} \]
If you didn't have the labels, could you tell which of these two images is displacement-mapped and which is bump-mapped?