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.

### Noise:
- Noise can be 1D, 2D, or 3D
- Noise output is a function of input value(s)
- Typically, those input values are where you are on the object, but they don’t have to be
- Noise ranges from -1.0 to +1.0 or from 0.0 to 1.0
- Noise might look random, but it really isn’t
- Noise has Coherency (i.e., if you change the input value to the noise function a little, the output value will only change a little)
- Noise has Repeatability (i.e., if you supply the same inputs, the noise function will always give you back the same output)
- Noise 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: C¹ continuity at the whole-number values
Quintic: C² continuity at the whole-number values
### Noise Octaves

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

### Cubic Quintic

#### Noise values

- Cubic: $N_0 = 1 - 3t^2 + 2t^3$
- Cubic: $N_1 = 3t^2 - 2t^3$
- Cubic: $N_2 = t - 2t^2 + t^3$
- Cubic: $N_3 = -t^2 + t^3$
- Cubic: $C_{i=0} = 0$
- Cubic: $C_{i=1} = 0$

- Quintic: $N_0 = 1 - 10t^5 + 15t^4 - 6t^3$
- Quintic: $N_1 = 10t^5 - 15t^4 + 6t^3$
- Quintic: $N_2 = t - 6t^2 + 8t^4 - 3t^5$
- Quintic: $N_3 = -4t^3 + 7t^2 - 3t^3$
- Quintic: $C_{i=0} = \frac{1}{2} t^2 - \frac{3}{2} t^3 + \frac{3}{2} t^4 - \frac{1}{2} t^5$
- Quintic: $C_{i=1} = \frac{1}{2} t^3 - t^2 + \frac{1}{2} t^3$

### Image Representation of 2D Noise

#### 1 Octave

#### 4 Octaves

### 3D Surface Representation of 2D Noise

#### 1 Octave

#### 4 Octaves

### 3D Volume Rendering of 3D Noise

#### 1 Octave

Has continuity in $X$, $Y$, and $Z$.

### Volume Isosurfaces of 3D Noise

#### 1 Octave

- $S^* = \text{Mid-value}$

#### 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?

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;
```

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 ± 0.5000</td>
<td>0.0000 → 1.0000</td>
</tr>
<tr>
<td>1</td>
<td>nv.g</td>
<td>0.25 ± 0.2500</td>
<td>0.0000 → 0.5000</td>
</tr>
<tr>
<td>2</td>
<td>nv.b</td>
<td>0.125 ± 0.1250</td>
<td>0.0000 → 0.2500</td>
</tr>
<tr>
<td>3</td>
<td>nv.a</td>
<td>0.0625 ± 0.0625</td>
<td>0.0000 → 0.1250</td>
</tr>
<tr>
<td>sum</td>
<td>2.0 ± 1.0</td>
<td>~ 1.0 → 3.0</td>
<td></td>
</tr>
<tr>
<td>sum – 1</td>
<td>1.0 ± 1.0</td>
<td>~ 0.0 → 2.0</td>
<td></td>
</tr>
<tr>
<td>(sum – 1)/2</td>
<td>0.5 ± 0.5</td>
<td>~ 0.0 → 1.0</td>
<td></td>
</tr>
<tr>
<td>(sum – 2)</td>
<td>0.0 ± 1.0</td>
<td>~ -1.0 → 1.0</td>
<td></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:

```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 now -1. → 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:

```plaintext
Noise3D 128
```

to get dimension 128 × 128 × 128, or choose whatever resolution you want (up to around 400 × 400 × 400).
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;  // in initGraphics:
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`.

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
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);

    unsigned char * texture = new unsigned char[nums * numt * 4];
    fread(texture, 4 * nums * numt, 1, fp);

    *width = nums;
    *height = numt;
    *depth = numt;

    return texture;
}
```

```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);

    unsigned char * texture = new unsigned char[4 * nums * numt * nump];
    fread(texture, 4 * nums * numt * nump, 1, fp);

    *width = nums;
    *height = numt;
    *depth = nump;

    return texture;
}
```
How to Use Noise

Have an equation that relates some input values (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.

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

Elliptical Dots with Tolerance

Elliptical Dots with Tolerance and Noise

Elliptical Dots with Tolerance and Noise

Elliptical Dots with Tolerance and Noise
If You Didn’t Have the Labels, Could You Tell Which of These Two Images is Displacement-Mapped and Which is Bump-Mapped?