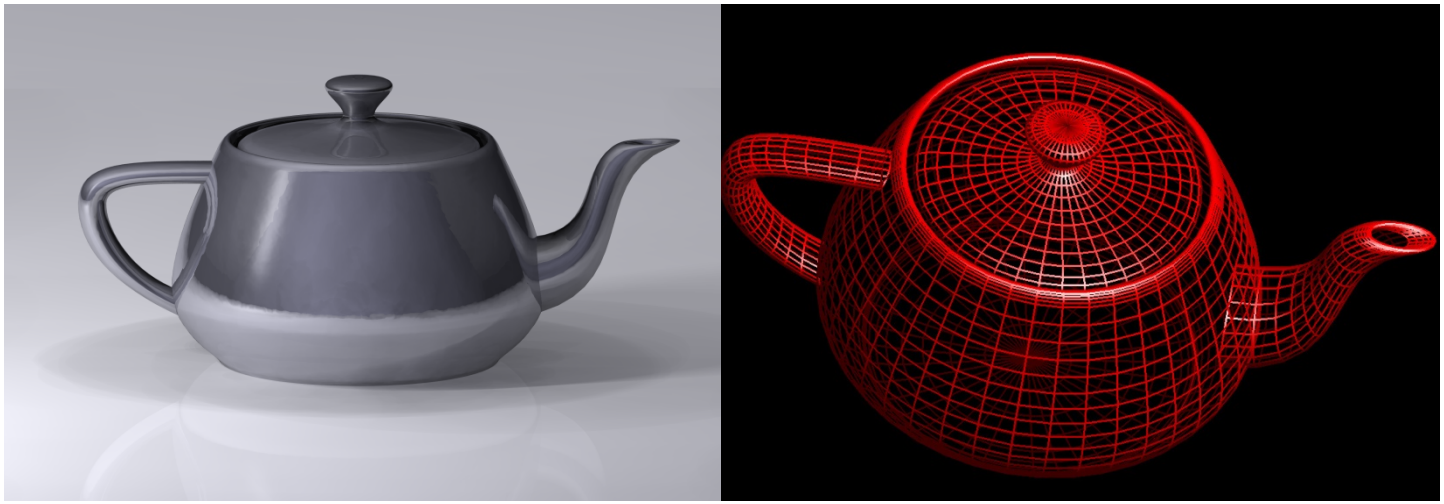


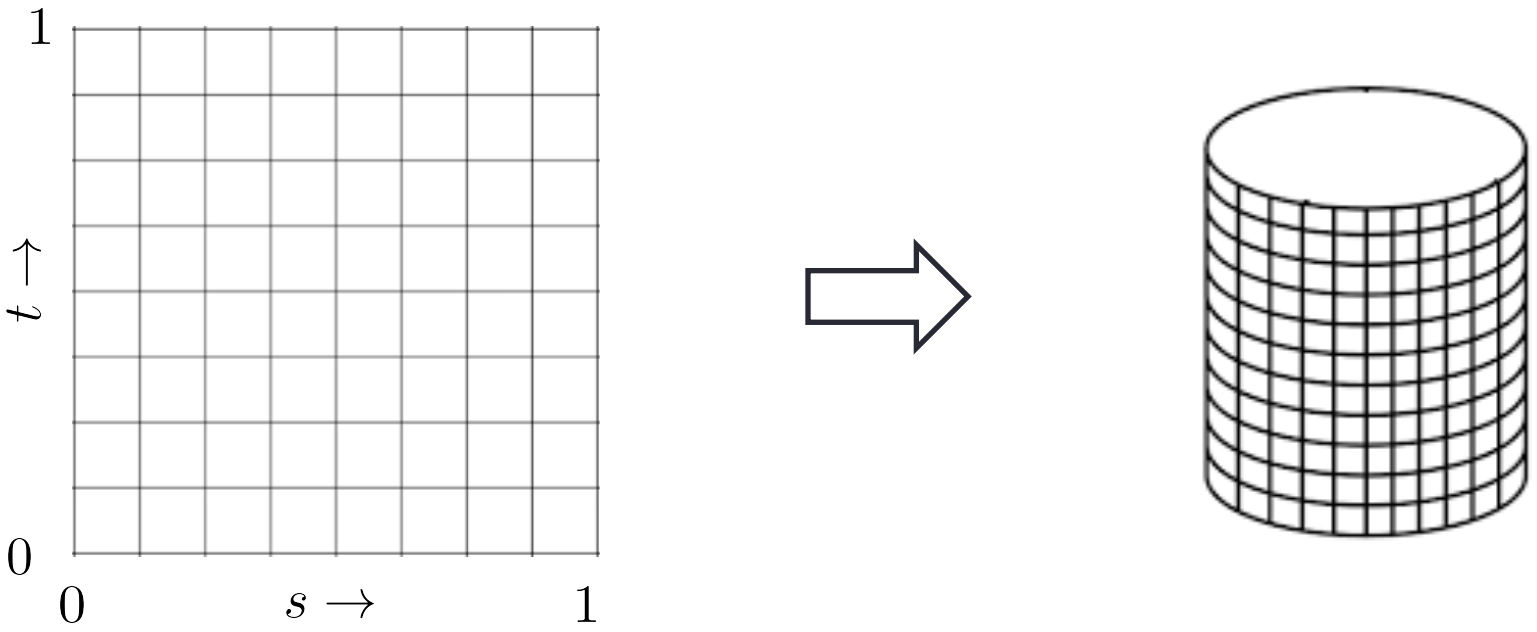
Foundations of Computer Graphics

Saurabh Ray

We will start in a few minutes.

Tuesday, April 7 instead of April 2.

Parametric Surfaces



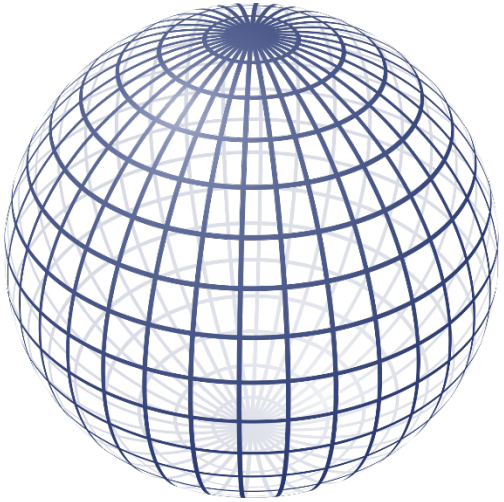
Suppose that we want to map the grid into a mesh for a cylinder.

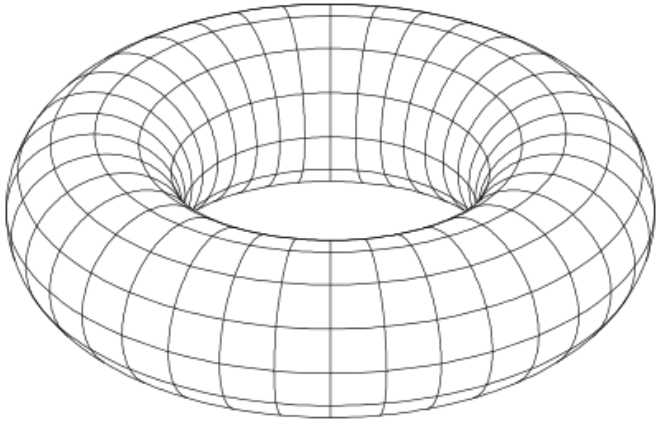
Each point (s, t) is mapped to a point on the surface of the cylinder.

What is the mapping function?

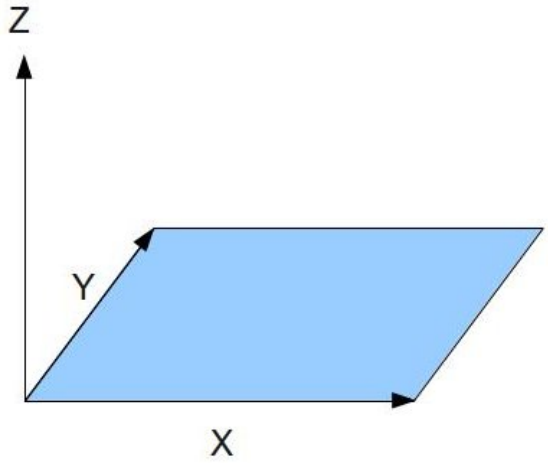
Parametric Surfaces

What about other shapes like cone, sphere, torus?

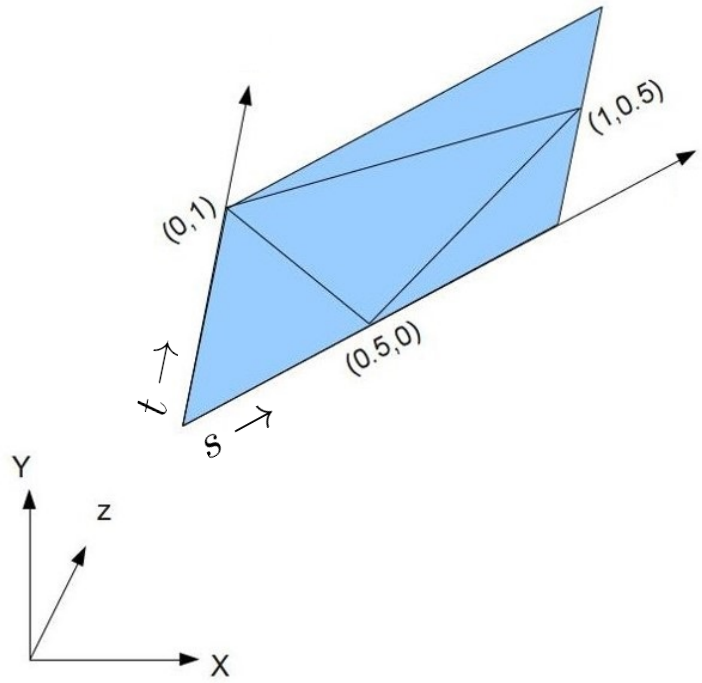




Normal Mapping



Coordinate system in which normals are defined.

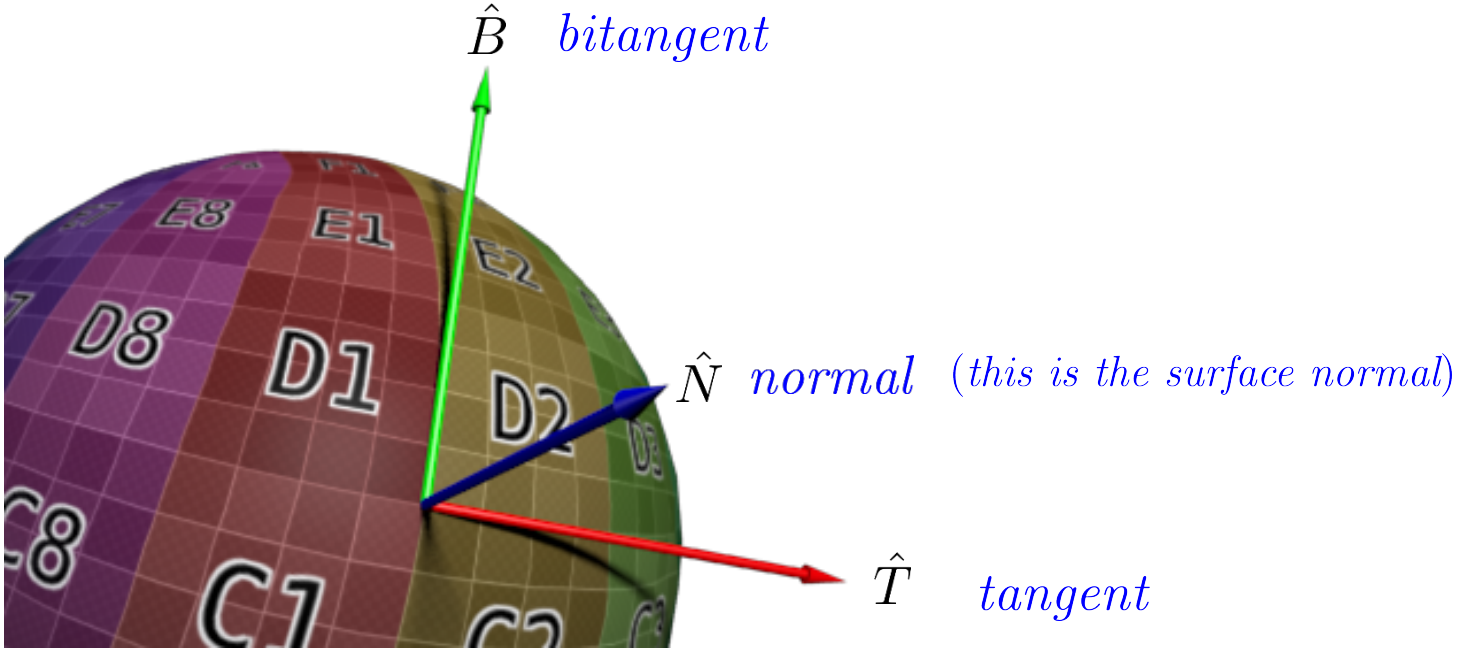


Placing the normal map on a triangle whose vertices have texture coordinates $(0.5, 0)$, $(1, 0.5)$ *and* $(0, 1)$.

The normal we get from the texture needs to be modified according to the orientation of surface.

Normal Mapping

In general, we use a "local" coordinate frame to interpret the coordinates.



The "modified" normal we get from the texture is described in this frame.

We convert it to the coordinate frame where we do lighting.

Normal Mapping

The tangent frame can be easily computed for surfaces described by an equation using partial derivatives.

How do we compute the tangent frame for triangular meshes?

- For each triangle, compute the tangent, bitangent and normal

- For each vertex, average the values at adjacent triangles

- For fragments, we get the vectors by linear interpolation

Note: This leads to slight inaccuracy since after the averaging and interpolation, the three vectors are not perpendicular to each other.

We ignore this since if the triangles are small, the error is small.

Normal Mapping

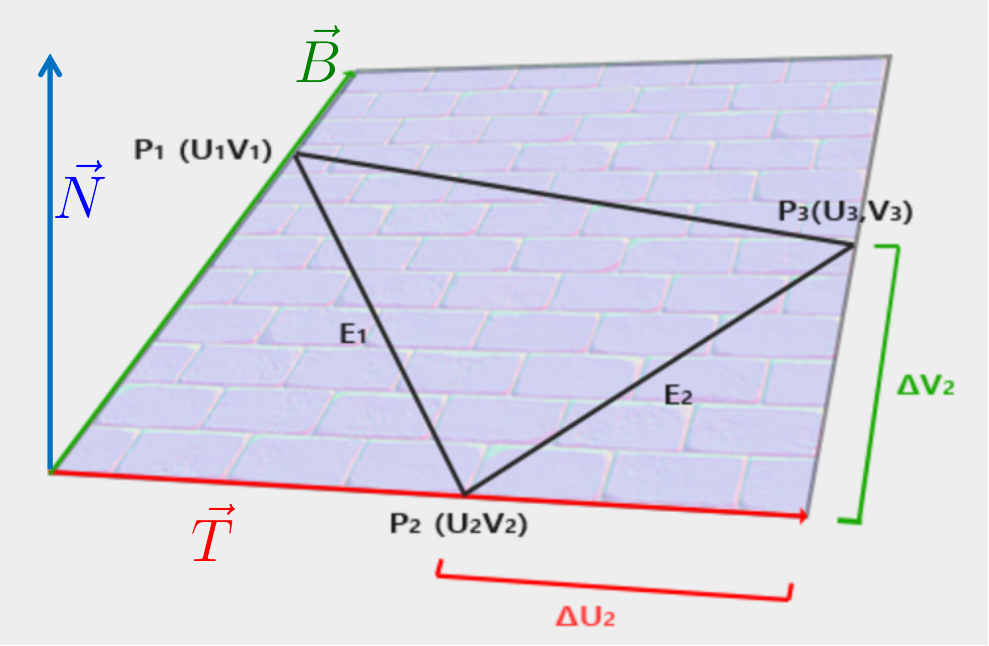
How do we compute tangents and bitangent on a triangle?

We have texture coordinates at each vertex.

These are along $\vec{T}$ and $\vec{B}$.

We also know the positions of the vertices.

From this we can figure out $\vec{T}$ and $\vec{B}$.



This is how it would look if we put the entire texture in the plane of the triangle so that the texture coordinates at the vertices match.

Note that $\vec{T}$ and $\vec{B}$ need not be of unit length.

They are as long as the length and height of the texture respectively.

Normal Mapping

Let $\vec{E}_1 = P_2 - P_1$

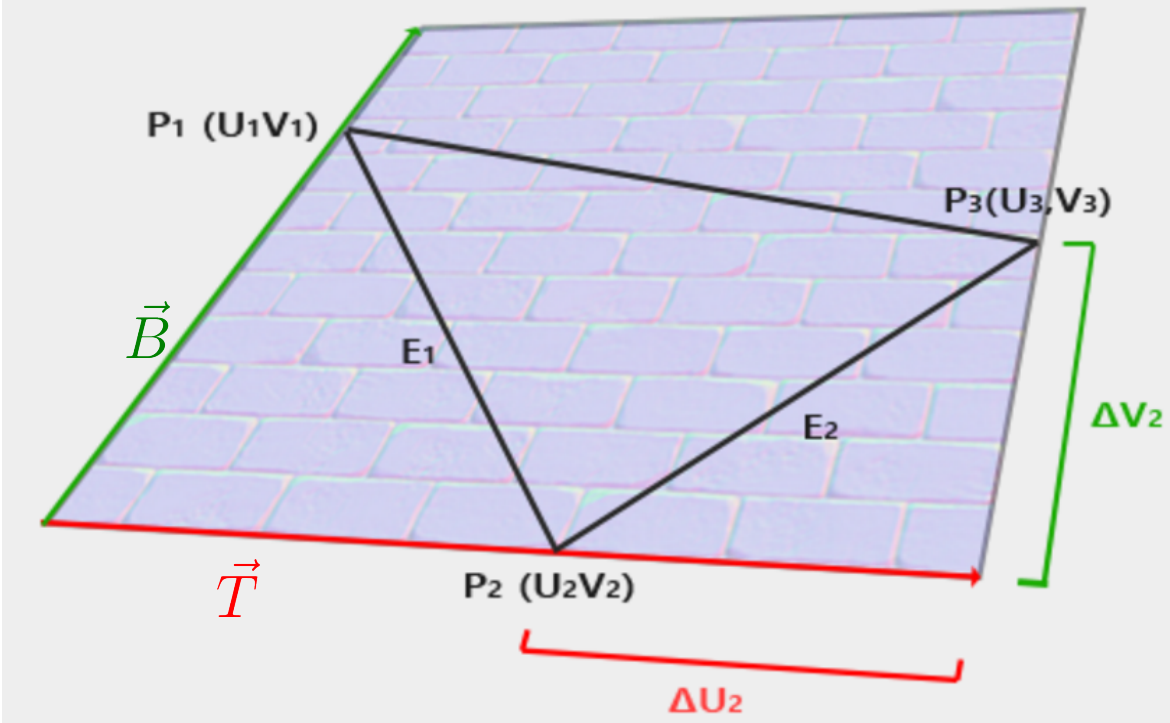
$\vec{E}_2 = P_3 - P_2$

$\Delta U_1 = U_2 - U_1$

$\Delta V_1 = V_2 - V_1$

$\Delta U_2 = U_3 - U_2$

$\Delta V_2 = V_3 - V_2$



$\vec{E}_1 = \Delta U_1 \vec{T} + \Delta V_1 \vec{B}$

$\vec{E}_2 = \Delta U_2 \vec{T} + \Delta V_2 \vec{B}$

two equations in two variables

Normal Mapping

$$\vec{E}_1 = \Delta U_1 \vec{T} + \Delta V_1 \vec{B}$$

$$\vec{E}_2 = \Delta U_2 \vec{T} + \Delta V_2 \vec{B}$$

Expanding to three coordinates:

$$(E_{1x}, E_{1y}, E_{1z}) = \Delta U_1 (T_x, T_y, T_z) + \Delta V_1 (B_x, B_y, B_z)$$

$$(E_{2x}, E_{2y}, E_{2z}) = \Delta U_2 (T_x, T_y, T_z) + \Delta V_2 (B_x, B_y, B_z)$$

In matrix notation:

$$\begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

Normal Mapping

$$\begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix}^{-1} \begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} = \frac{1}{\Delta U_1 \Delta V_2 - \Delta U_2 \Delta V_1} \begin{bmatrix} \Delta V_2 & -\Delta V_1 \\ -\Delta U_2 & \Delta U_1 \end{bmatrix} \begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix}$$

Normal Mapping

We now need two more vertex attributes: **tangent** and **bitangent**.

Just like normals, these are interpolated for each fragment so that we get the normal, tangent and bitangent at each fragment.

Now, we use the texture coordinates to read the color (r, g, b) from the normal map texture.

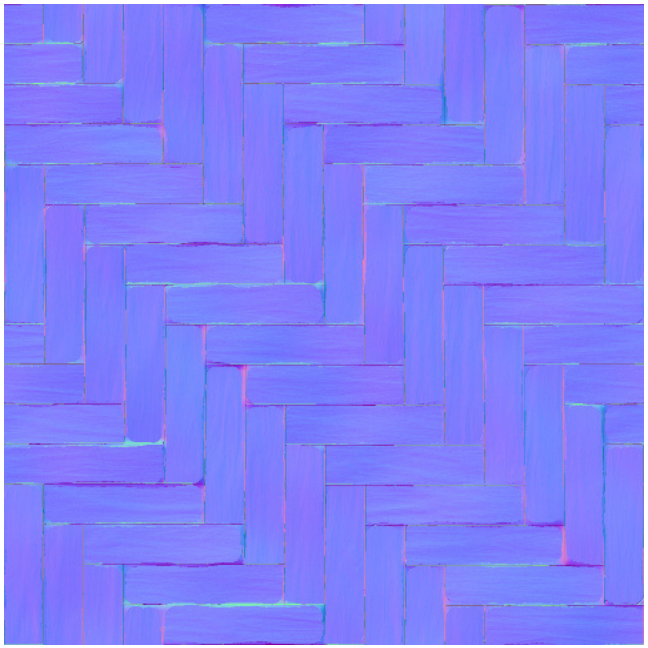
Convert color to normal in the tangent frame: $(2r-1, 2g-1, 2b-1)$

This is converted to world frame for lighting:

$$\vec{N}' = (2r-1)\hat{T} + (2g-1)\hat{B} + (2b-1)\hat{N}$$

where $\hat{T} = \frac{\vec{T}}{\|\vec{T}\|}$, $\hat{B} = \frac{\vec{B}}{\|\vec{B}\|}$ and $\hat{N} = \frac{\vec{N}}{\|\vec{N}\|}$.

Problem 3 (10 points)**.**
Display a unit square with the texture `wood-diffuse.jpg` and using `wood-normal.jpg` as the normal map. Use phong lighting and set the shininess value so the square looks like a polished wood surface.

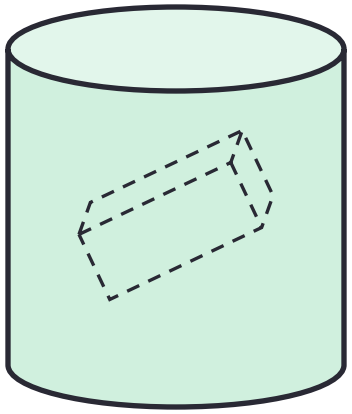


Solid (3d) Textures

Looks better than a 2d texture pasted on the surface of the model.

Storing the texture in a file limits resolution and requires more space, but is faster.

Procedural generation is slower but requires less storage space and has potentially unlimited resolution.

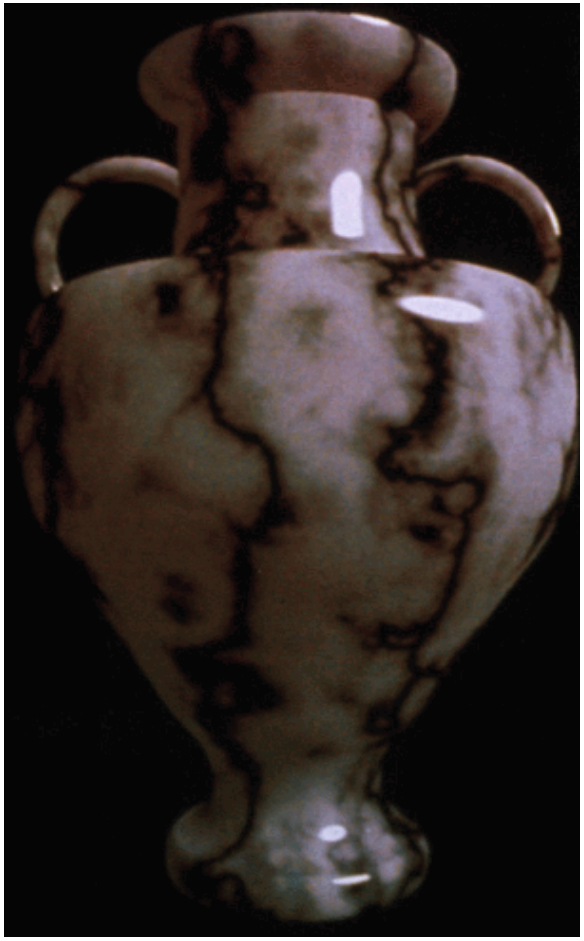


We want a function that given any point in the cylinder outputs the wood color at that point.

Creating a function so that each horizontal cross section has perfectly circular rings with two colors is easy.

Can be made more realistic using perturbations.

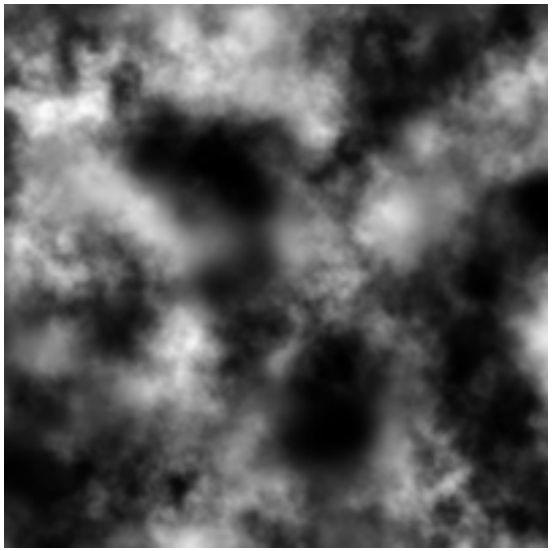
Procedural Generation



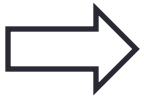
Many natural phenomena can be simulated procedurally.

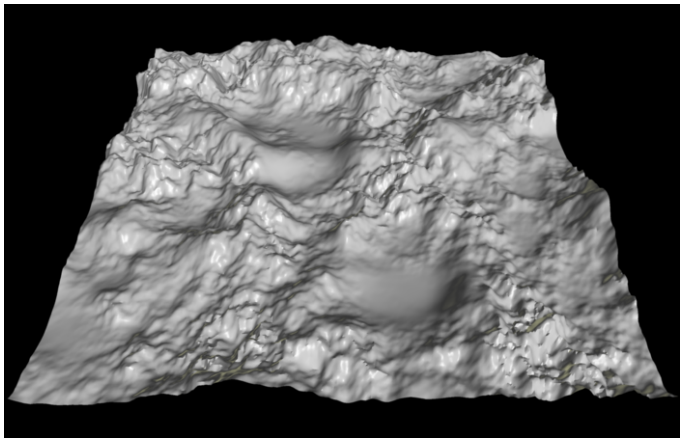
Key ingredient: Randomness.

Terrains using Height Maps



Height Map





Generated Terrain

We can use a terrain texture like this:

Procedural Noise

Natural phenomena contain a mixture of structure and randomness.

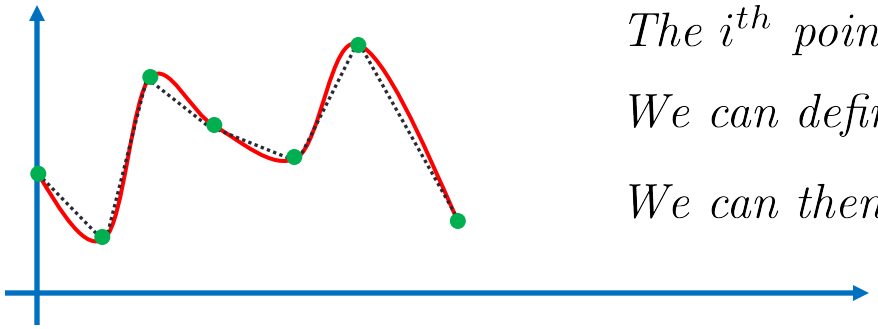
For example, altitude in a terrain isn't totally random.

There is random but gradual change as we move around in the terrain.

And this happens at all scales.

How do we obtain a smooth random function using a pseudo random number generator (PRNG)?

Idea. Interpolation of randomly generated points.



The i^{th} point has coordinates $(i, rand(i))$.

We can define the function on $[0, n]$ s.t. $f(0) = f(n)$.

We can then make it periodic: $g(t) = f(t \bmod n)$.

Procedural Noise

How do we achieve randomness at every scale?

$$N(t) = g(t) + \frac{1}{2}\, g(2t) + \frac{1}{4}\, g(4t) + \frac{1}{8}\, g(8t) + \cdots$$

We add terms with exponetially increasing frequency and exponentially decreasing amplitude.

This also gives us *self similarity*.

We could also add an *offset* to each scaled copy.

Procedural Noise

Natural phenomena contain a mixture of structure and randomness.

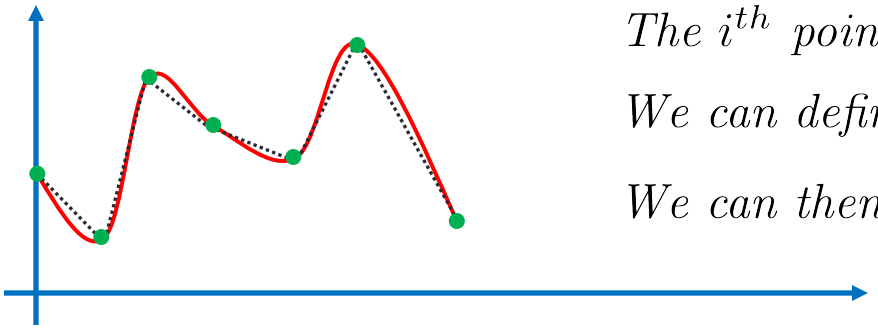
For example, altitude in a terrain isn't totally random.

There is random but gradual change as we move around in the terrain.

And this happens at all scales.

How do we obtain a "smooth" random function using a pseudo random number generator (PRNG)?

Idea. Interpolation of randomly generated points.



The i^{th} point has coordinates $(i, rand(i))$.

We can define the function on $[0, n]$ s.t. $f(0) = f(n)$.

We can then make it periodic: $g(t) = f(t \bmod n)$.

This is called **Value Noise**.

Procedural Noise

How do we achieve randomness at every scale?

$$N(t) = g(t) + \frac{1}{2}\, g(2t) + \frac{1}{4}\, g(4t) + \frac{1}{8}\, g(8t) + \cdots$$

We add terms with exponetially increasing frequency and exponentially decreasing amplitude.

This also gives us *self similarity.*

We could also add an *offset* to each scaled copy.

The functions $g(t), g(2t), \ldots$ are often called **octaves**.

In practice, we don't need an infinite sum since the amplitudes decrease exponentially.

This technique is often referred to as **fractional brownian motion.**

Closely related: **Turbulence.** $N(t) = |g(t)| + \frac{1}{2}\, |g(2t)| + \frac{1}{4}\, |g(4t)| + \cdots$

Gradient Noise.

We change how we obtain the initial function $f(t)$.

For each integer $t \in [0, n]$:

- we prescribe $f(t)$ using a PRNG

- we prescribe the slope $f'(t)$ using a PRNG

How do we define a "smooth" function $f(t)$ on $[0, n]$ to match the prescriptions at each integer in $[0, n]$?

Is there a unique way to construct such a function? **No!**

Procedural Noise

Here is one solution. For $t \in [i, i+1]$, define

$$f(t) = \begin{aligned}&(1-(t-i)) \cdot [\, f(i) + f'(i) \cdot (t-i)\,] \\ &+(t-i) \cdot [\, f(i+1) - f'(i+1) \cdot (i+1-t)\,]\end{aligned}$$

Instead of these, we can use $(1-\alpha(t-i))$ *and* $\alpha(t-i)$ *where* $\alpha(x)$ *is a smooth 'fading' function that goes from* 0 *to* 1 *as* x *goes from* 0 *to* 1.

e.g. $\alpha(x) = 3x^2 - 2x^3$ *or* $\alpha(x) = 6x^5 - 15x^4 + 10x^3$

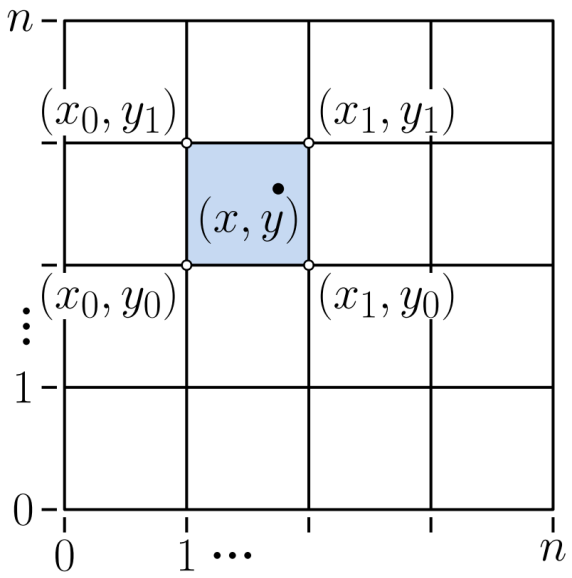
$$\begin{aligned}f(t) &= (1-\alpha(t-i)) \cdot (\, f(i) + f'(i) \cdot (t-i)\,) \\ &\quad + \alpha(t-i) \cdot (\, f(i+1) - f'(i+1) \cdot (i+1-t)\,)\end{aligned}$$

$$\begin{aligned}&= (1-\alpha(t-\lfloor t \rfloor)) \cdot (\, f(\lfloor t \rfloor) + f'(\lfloor t \rfloor) \cdot (t-\lfloor t \rfloor)\,) \\ &\quad + \alpha(t-\lfloor t \rfloor) \cdot (\, f(\lceil t \rceil) - f'(\lceil t \rceil) \cdot (\lceil t \rceil - t)\,)\end{aligned}$$

since $i = \lfloor t \rfloor$ *and* $i+1 = \lceil t \rceil$.

Procedural Noise

How do we generalize value noise to two dimensions?



At each grid point in $[0, n-1] \times [0, n-1]$, *the values are randomly generated.*

For any other point $p = (x, y)$, *we bilinearly interpolate the values at the four corners of the grid cell containing* p.

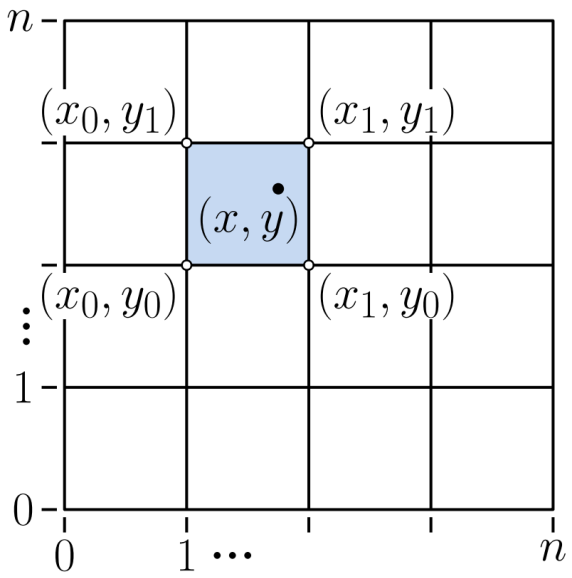
Let $x_0 = \lfloor x \rfloor$, $x_1 = (x_0 + 1) \bmod n$,
$y_0 = \lfloor y \rfloor$, $y_1 = (y_0 + 1) \bmod n$,
$a = x - x_0$, $b = y - y_0$.

$$\begin{aligned} f(x,y) = \quad & (1-a)(1-b) \cdot f(x_0, y_0) \\ & +(1-a)b \cdot f(x_0, y_1) \\ & +a(1-b) \cdot f(x_1, y_0) \\ & +ab \cdot f(x_1, y_1). \end{aligned}$$

As before, we can use a smooth fading function.

Procedural Noise

How do we generalize value noise to two dimensions?



At each grid point in $[0, n-1] \times [0, n-1]$, *the values are randomly generated.*

For any other point $p = (x, y)$, *we bilinearly interpolate the values at the four corners of the grid cell containing* p.

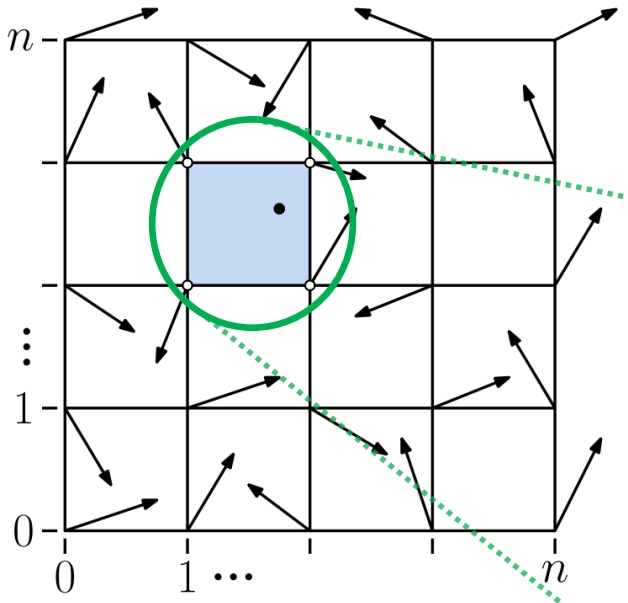
Let $x_0 = \lfloor x \rfloor$, $x_1 = (x_0 + 1) \bmod n$,
$y_0 = \lfloor y \rfloor$, $y_1 = (y_0 + 1) \bmod n$,
$a = x - x_0$, $b = y - y_0$.

$$\begin{aligned} f(x,y) = \quad & \alpha(1-a)\,\alpha(1-b) \cdot f(x_0, y_0) \\ & +\alpha(1-a)\,\alpha(b) \cdot f(x_0, y_1) \\ & +\alpha(a)\,\alpha(1-b) \cdot f(x_1, y_0) \\ & +\alpha(a)\,\alpha(b) \cdot f(x_1, y_1). \end{aligned}$$

Using a smooth fading function α.

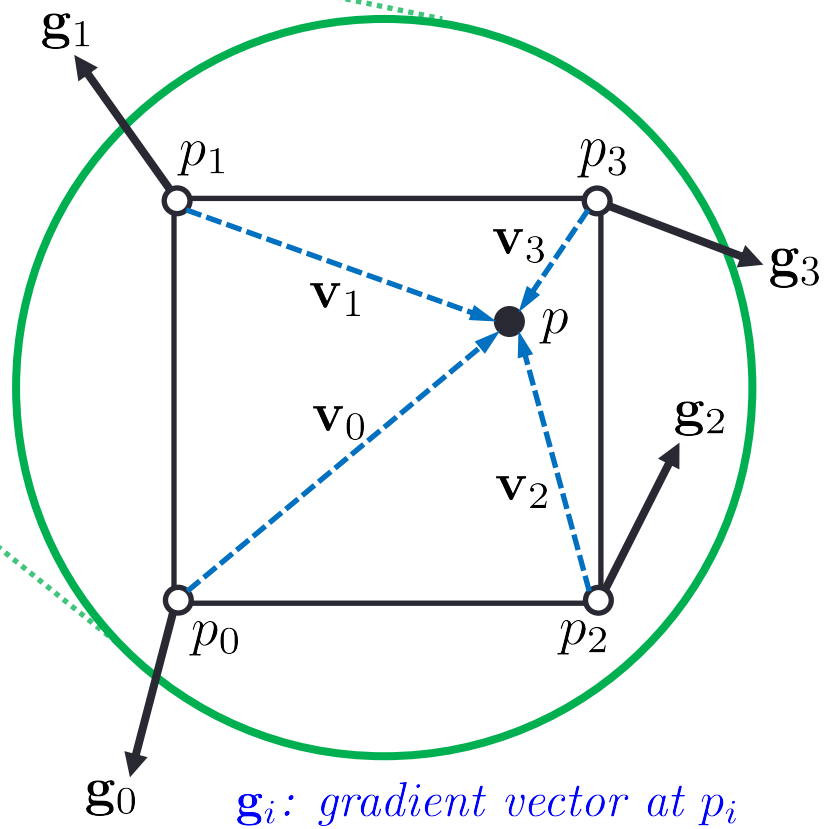
Procedural Noise

How do we generalize gradient noise to two dimensions?



For each grid point in $[0, n-1] \times [0, n-1]$, *we also prescribe a random gradient vector.*

We then construct a "smooth" function matching the prescribed values and gradients at each point.



Let $\lambda_i = f(p_i) + \mathbf{g}_i \cdot \mathbf{v}_i$,
$\forall i \in \{0, 1, 2, 3\}$.

$$
\begin{aligned}
f(p) = \; & \alpha(1-a)\,\alpha(1-b) \cdot \lambda_0 \\
& + \alpha(1-a)\,\alpha(b) \cdot \lambda_1 \\
& + \alpha(a)\,\alpha(1-b) \cdot \lambda_2 \\
& + \alpha(a)\,\alpha(b) \cdot \lambda_3 .
\end{aligned}
$$

$\mathbf{g}_i$: *gradient vector at* p_i

$\mathbf{v}_i$: $p - p_i$

Procedural Noise

As before, we can add octaves to obtain self similarity at different scales.

$$N(p) = \sum_{i=0}^{k} \frac{1}{r^i} \cdot f(s^i\, p)$$

r and s are some constants > 1.

How do we generalize this to three dimensions?

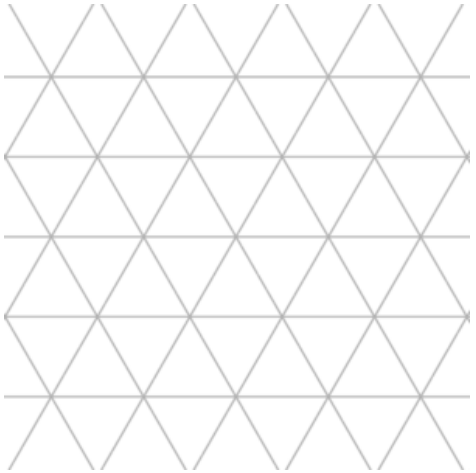
Time complexity in d dimensions? $O(d\, 2^d)$

Can be improved to $O(d^2)$ using **simplex noise**.

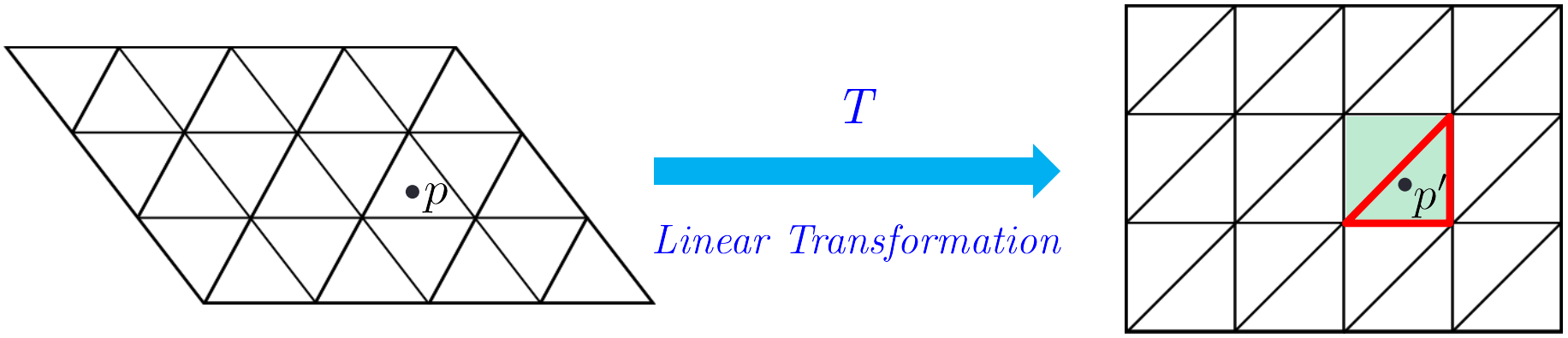
Idea. Tile the space with simplices instead of cubes.

Instead of 2^d corners of a cube, now we need to interpolate only using the values and gradients at the $d+1$ corners of a simplex.

Given a point p, how do we figure out the corners of the simplex containing it?



Procedural Noise

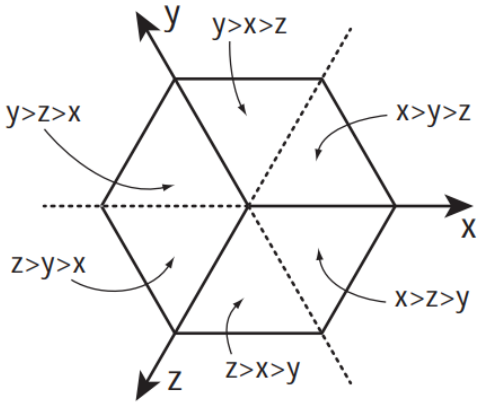


Given a point p in the first grid, we compute $p' = T(p)$.

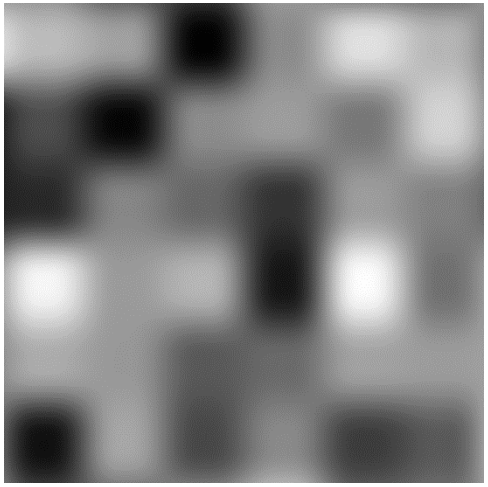
In the second grid, we can figure out the cubic grid cell containing p' using the floor function.

Within that cell, we can figure out the corners of the simplex σ' containing p' using the sorted order of the coordinates of p'. *Non-trivial!*

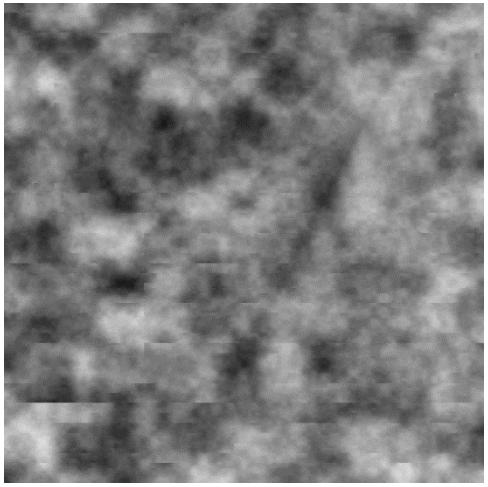
The simplex containing p is $\sigma = T^{-1}(\sigma)$.



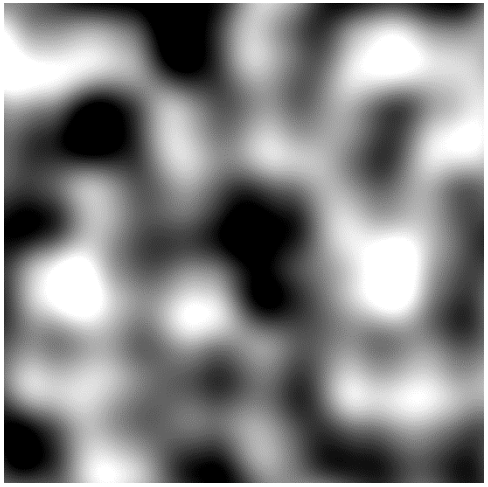
Procedural Noise



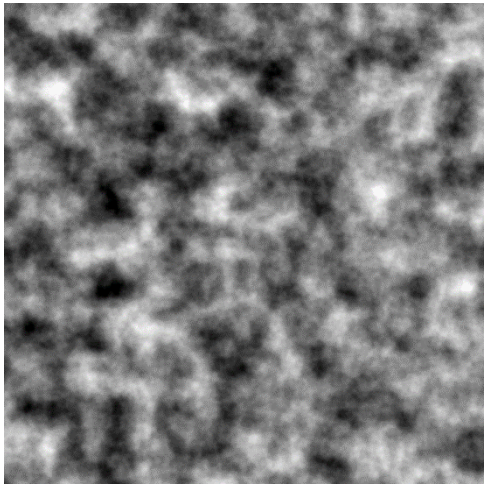
Value Noise



Sum of 8 octaves of Value Noise



Gradient Noise

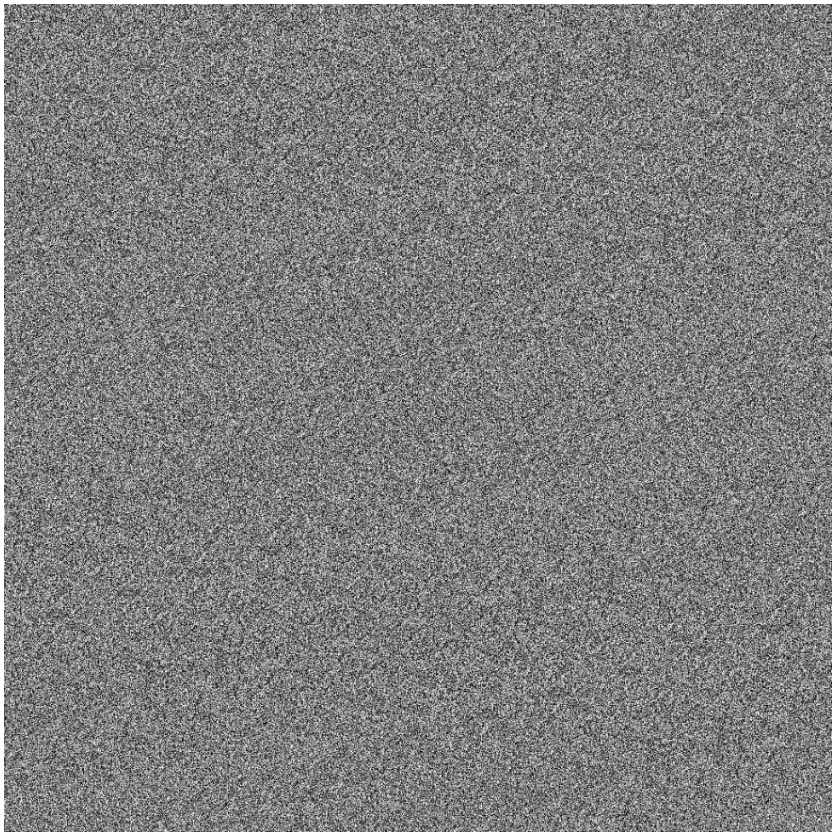


Sum of 8 octaves of Gradient Noise

Procedural Noise

Generating random numbers in GLSL:

```
float random(vec2 v, float salt){
    // returns a value in [0,1)
    float a = 5678.1234;
    vec2 b = vec2(12.3456, 78.910);
    float c = 1.1235+salt;
    return fract(a*sin(dot(b,v)+c));
}
```



This is actually a hash function.