Making Noise

CSE 681
Han-Wei Shen
Procedural Textures

- Generate textures as you are shade intersection points
- Texture map is a function of texture coordinates
  - Input: s,t,r
  - Output: color, opacity, or even shading
Procedural Texture Example

- Fire
Procedural Texture Example

- Planet
Procedural Texture Example

- Cloud and Water
Solid Texturing

- Procedural textures are often used as “solid textures”
- Let \( s = x, t = y, r = z \)
- No need to parameterize surfaces
- Objects appear sculpted out of solid substances
Noise Function

- A key ingredient to introduce “controlled” randomness
- Famous noise function – Perlin Noise
  - $N(x,y,z)$ returns a random number in $[-1,1]$, where $(x,y,z)$ can be an arbitrary 3D point
- Properties (unlike white noise):
  - Smooth
  - Correlated
  - Band limited
Basic idea of Perlin Noise

- Divide the space to a 3D regular lattice
- A pseudo random gradient vector is assigned to each lattice point
  - this gradient is used to compute a dot product
- The noise value for any arbitrary 3D point is a weighted sum of the linear functions at its eight nearest lattice point
The resolution of the lattice does not need to be high as we can repeat it without obvious pattern (no global pattern)

The lattice structure makes it easy to find which cell a given \((x,y,z)\) is in

\[[i,j,k] = [\text{fl}(x), \text{fl}(u), \text{fl}(z)]\] fl: floor function

The cell is then have eight corners: \([i,j,k], [i+1, j, k], [i,j+1, k] \ldots, [i+1,j+1,k+1]\)
Pseudo random gradients

- Create a table of 255 random gradients uniformly distributed over a unit sphere
  - Choose points uniformly within the cube \([-1, 1]^3\)
  - Throw out any points outside the sphere
  - Project surviving points onto the unit sphere (normalize the point, that is)
- Then map any \([i,j,k]\) into this table to assign a gradient to each lattice point
  - create a permutation table \(P\)
  - Map \([I,j,k]\) to a gradient \(G[n]\) using \(P\)
    - \(n = P[i\%256] \rightarrow n = P[(n+j) \%256] \rightarrow n = P[(n+k)\%256]\)
Pseudo random gradients

1. Constructing the Gradient table

for i in [0...255]
repeat
  x = random(-1. ... +1.)
  y = random(-1. ... +1.)
  z = random(-1. ... +1.)
until x*x + y*y + z*z < 1.0
G[i] = normalize[x, y, z]

2. Get a permutation index for [i,j,k]

fold(i, j, k)
{
  n = P[i mod 256]
  n = P[(n+j) mod 256]
  n = P[(n+k) mod 256]
}
or
P[P[P[i]+j]+k]

Lattice [I,j,k] then has the gradient G[fold(i,j,k)]
Evaluate the wavelets

- For any arbitrary point \([x,y,z]\), the wavelet from lattice \([i,j,k]\) is evaluated as:
  - \([u,v,w] = [x-i,y-j,z-k]\)
  - Weight = \(\text{drop}(u) \times \text{drop}(v) \times \text{drop}(w)\)
    where \(\text{drop}(t) = 1-(3|t|^2+2|t|^3)\)
  - \(F = G[i,j,k] \cdot [u,v,w]\)
  - Wavelet = Weight \times F

- The noise value for \((x,y,z)\) is then the sum of eight Wavelets from the point’s cell corners
Improved Perlin Noise

- Published in 2002 after almost 20 years
- The drop off (or interpolating) function is changed from $3|t|^2-2|t|^3$ to $6|t|^5-15|t|^4+10|t|^3$

old

new
Application of noise

- Random surface texture
  - Color = white * noise(point)
Application of Noise

- Colored noise:
  - Color = Colorful(Noise(k*point)); k is to control the feature size. The larger k is, the smaller the feature
Application of Noise

- Simulate 1/f noise

1/f noise, is a signal or process with a frequency spectrum such that the power spectral density is proportional to the reciprocal of the frequency. Sometimes pronounced as one over f noise, it is also called flicker noise. In other words, it is a sound that falls off steadily into the higher frequencies, instead of producing all frequencies equally.

\[ \sum_i \frac{\text{Noise}(\text{point} * 2^i)}{2^i} \]
Application of Noise

\[ \text{noise} \]

\[ \sin(x + \sum \frac{1}{f(|\text{noise}|)}) \]

\[ \sum \frac{1}{f(\text{noise})} \]

\[ \sum \frac{1}{f(|\text{noise}|)} \]
Application of Noise

- The differential of the noise function $D_{\text{noise}}()$
  - $\text{Normal} += D_{\text{noise}}(\text{point})$
Application of Noise

- 1/f bump texture

```plaintext
f = 1
while f < pixel_freq
    normal += Dnoise(f * point)
    f *= 2
```
Application of Noise

- Turbulence (very useful!!)

```plaintext
function turbulence(p)
    t = 0
    scale = 1
    while (scale > pixelsize)
        t += abs(Noise(p / scale) * scale)
        scale /= 2
    return t
```
Application of Noise

- Simulating marble effects

```python
function boring_marble(point)
    x = point[1]
    return marble_color(sin(x))

function marble(point)
    x = point[1] + turbulence(point)
    return marble_color(sin(x))
```