Noise | Course Overview

Jim Whitehead
Computer generation

- Assume that there exists:
  - One or more instances of some “thing” (level, music, robot, map, NPC, texture, etc.) …
  - That meets a desirability guideline
    - Might be expressible as a fitness function
    - Might be human-guided, I’ll know it when I see it

- The “things” are comprised of sub-parts, and each sub-part typically has one or more ways it can vary
  - Sometimes discrete choices
  - Other times, a continuous range of possibilities
  - Example: small shed: has four walls, can vary width & height, location of door, type of roof (flat, peaked, slanted, etc.)
Variation space

- Each type of variation can be viewed as an axis in an abstract N-dimensional variation (or search) space
  - Shed example: shed length, shed width, shed height, shed door location, shed rooftype
  - Creates a 5-dimensional space

- Algorithmic creation of “things” can be viewed as a process of finding desirable instances within the variation space
Generative approaches

- There are different ways of finding good instances in the variation space:

- Random (noise)
  - Pick instance randomly

- Combine shapes and continuous functions (parametric design)

- Path-specified (grammars)
  - Explore only along certain paths in variation space

- Hill climbing with semi-randomness (evolutionary techniques)
  - Create population of instances, then select best ones, generate semi-random new ones from existing instances

- Constraint based (answer set programming)
  - Explicitly state desired qualities of instances
A noise function is a kind of random number generator.

Random number generator (typical):

- \( i = \text{random}() \)
- No parameters, returns random number in a range

With noise, return random number for a point

- \( l = \text{noise}(x) \)
- \( l = \text{noise}(x,y) \)
- \( l = \text{noise}(x,y,z) \)

That is, a noise function maps from \( \mathbb{R}^n \) to \( \mathbb{R} \)

- You input an \( n \)-dimensional point with real coordinates, it returns a real value
Why do we care about noise?

- Noise has many applications
  - Texture generation for a wide range of materials
  - World map generation
  - Cloud generation
  - Randomness in game systems

- Perlin noise examples
  - Experiments with Perlin Noise (sphere, lights, explosion)
  - Perlin Noise map generation
  - Shader demos of various materials
    - [https://github.com/ashima/webgl-noise/wiki](https://github.com/ashima/webgl-noise/wiki)
Problem with simple random noise

- A simple approach for procedurally creating a texture is:
  - Map a random number to each (x,y) coordinate

\[
\text{noise}(x,y) = \text{random()}
\]

- Random number generators often have a limited range (e.g., [0, 1])
- Control maximum value by multiplying by an amplitude value

\[a \times \text{noise}(x,y)\]

- Problem
  - Results don’t have any coherence to them.
  - Just a jumble of values.
  - Not appealing. Doesn’t look like anything, really.
Desirable noise qualities

- Ideally, a noise function will (informally):
  - Give the appearance of randomness
  - Be controllable, hence capable of generating many materials
  - Have random variations of roughly the same size, roughly isotropic
  - Quick to execute

- More formally:
  - Be a repeatable pseudorandom function of its inputs
  - Have a known range, often -1 to 1
  - Not exhibit obvious periodicity or regular patterns.
    - If pattern does repeat, do it on long time scales
  - Be stationary – should be translationally invariant
  - Be isotropic – should be rotationally invariant
Different families of noise functions

- Lattice noise
  - Value noise

- Gradient noise
  - Perlin noise
  - Simplex noise

- Value-gradient noise

- Lattice convolution noise

- Cell noise (Worley noise)

- Sparse convolution / spot noise
Lattice noise

- Uniformly distributed random number at each point
  \[ \text{noise}(x,y,z) = \text{random()} \]
- Interpolate for values in-between lattice points

For value noise:
- Linear interpolation: looks “boxy”, has lattice cell artifacts
- Use cubic interpolation instead (2\text{nd} derivative is continuous), such as cubic Catmull-Rom spline
- Can also use quadratic and cubic B-splines
Perlin Gradient noise

- Start with a lattice of random values (each point is P)
- For each lattice point, also create a random gradient vector, G

- Compute the inner product $G \cdot (P-Q)$
  - This will give the value at P of the linear function with gradient G which is zero at grid point Q
  - Or, the projection of the gradient vector along the direction vector to P

- In n dimensions, compute this product for all $2^n$ neighbors

- Now, interpolate between them
Noise

- Straight noise
- Make objects look dirty or worn
- Using gradient of noise to vary surface normal, can create bumpy appearance
- For more advanced effects, need to combine noise at multiple frequencies

Sum $1/f(\text{noise})$

- $\text{noise}(p) + \frac{1}{2} \text{noise}(2p) + \frac{1}{4} \text{noise}(4p) ...$
- Creates nice cloudy atmosphere effect

Amplitude/frequency view

- Amplitude: difference between min and max values
- Wavelength is distance from one red dot to next
- Frequency = 1/wavelength

Source: Hugo Elias Noise page
Amplitude & frequency variation

- Amplitude is halved as frequency doubles
Sum together to create $1/f(\text{noise})$

Source: Hugo Elias noise page:
http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
Sum $1/f(|\text{noise}|)$

- Instead of a fractal sum of noise, use a fractal sum of the absolute value of noise:
  - $|\text{noise}(p)| + \frac{1}{2} |\text{noise}(2p)| + \frac{1}{4} |\text{noise}(4p)| \ldots$

Sin(x + sum 1/f(|noise|))

- Use turbulence texture from the previous slide
- Do a phase shift in a simple stripe pattern.
- Stripe pattern itself is created with a sine function of the x coordinate of surface location:
  \[ \sin(x + |\text{noise}(p)| + \frac{1}{2} |\text{noise}(2p)| + ...) \]

Terrain generation

- Can map noise values to height map values
- Overlay a texture
- See:
Simplex noise

- Instead of interpolating among hypergrid neighbors, simplex noise divides space into simplices (n-dimensional triangles)

Advantages:
- Simplex noise has a lower computational complexity and requires fewer multiplications.
- Simplex noise scales to higher dimensions (4D, 5D) with much less computational cost, the complexity is $O(n)$ for dimensions instead of $O(n^2)$ of classic noise.
- Simplex noise has no noticeable directional artifacts (is isotropic).
- Simplex noise has a well-defined and continuous gradient everywhere that can be computed quite cheaply.
- Simplex noise is easy to implement in hardware.

Source: http://en.wikipedia.org/wiki/Simplex_noise
Noise pros/cons

- Noise is useful for generating a wide range of textures
- Via interesting mappings, can be used as a smooth source of randomness in a wide range of situations
- Most useful in content generation situations where a wide range of outputs are acceptable
  - I.e., there are few constraints on the generated artifacts
  - Those constraints that exist are baked into the noise algorithm
  - Once there are many constraints, or many areas of unacceptability, or need wide variation, start needing more control