Procedural Graphics

Rachel Hwang, CIS 277 Spring 2016
What is Proceduralism?

- Creation using algorithms rather manual work
- Finding a “recipe”
- Applications:
  - Music
  - Visual Art
  - Graphics
  - Sculpture
  - Food?
  - Etc.
The Power of Infinity!
Artificial Life

- How to generate a series of rules that mimic the chaos of organic life?
  - Underlying regularity is everywhere
  - Surprisingly easy to capture complex behavior with a relatively simple set of rules.
Example: Conway’s Game of Life
Example: Procedural Mountains

- Let’s say we want to draw a mountain range in 2D
- For every step along x, find a height value.
- Can we use a random number generator?
Example: Procedural Mountains

- Let’s say we want to draw a mountain range in 2D
- For every step along x, find a height value.
- Can we use a random number generator?

Eh.
Example: Procedural Mountains

- Let’s say we want to draw a mountain range in 2D
- For every step along x, find a height value.
- Let’s try a sine function + random noise.
Perlin Noise

- Gradient Noise technique invented by Ken Perlin, 1983
  - Academy Award for Technical Achievement
- Idea: sample a noise function at multiple “resolutions” then add them together to get organic-looking results.

![Graphs showing amplitude and frequency changes](image-url)
Perlin Noise

- Uses a pseudo-random noise function — input always yields same output.
- Example:

```c
float Noise(int x)
{
    x = (x<<13) ^ x;
    return ( 1.0 - ( x * ( x * x * 15731 + 789221) + 1376312589 ) & 0x7fffffff ) / 1073741824.0;
}
```

- Each successive noise function you sum is called an octave.
- Perlin Noise varies the output of the noise using two parameters:
  - Frequency: sample rate
  - Persistence: decay rate of the functions amplitude.
  - Usually both these parameters are powers of 2 (hence octaves)
Perlin Noise

<table>
<thead>
<tr>
<th>Frequency</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistence = 1/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/4</td>
<td>1/16</td>
<td>1/64</td>
<td>1/256</td>
<td>1/1024</td>
</tr>
<tr>
<td>Result:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence = 1/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/8</td>
<td>1/16</td>
<td>1/32</td>
</tr>
<tr>
<td>Result:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence = 1 / \sqrt{2}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1/1.414</td>
<td>1/2</td>
<td>1/2.828</td>
<td>1/4</td>
<td>1/5.656</td>
</tr>
<tr>
<td>Result:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence = 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplitude:</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Result:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Perlin Noise

- Any number of dimensions
- PerlinNoise_1d(float x)
  - total = 0
  - persistence = 1/2
  - loop i in range(0, numOctaves)
    - frequency = 2^i
    - amplitude = persistence^i
    - total += sampleNoise(x, frequency) * amplitude
  - return total
Perlin Noise Applications

- Infinite
L-Systems

- An L-system or Lindenmayer system is a formal grammar.
- A powerful tool for encoding recursive structure.
- Consists of:
  - Alphabet of symbols
  - Set of production rules for replacing one symbol with a larger string
  - Initial “axiom” string
- Originally used to describe plant shape and growth
We can treat each symbol in our grammar as instructions to a simple turtle program.

Example:
- Axiom: FX, Rule: X -> S[-FX]+FX
- F: draw a branch
- -/+: turn (-/+45 degrees
- S: Return here after completing []
L-System Applications
Procedural Buildings

- Urban environments are common in games and animation.
- Layers of detail
  - Basic geometry
  - Decorative features
  - Texture
  - Layout
- Lots of repetition!
Models with Shape Grammar

- Formal language of strings and production rules
- Encodes spatial distribution of features
- Concise rule notation
  - \texttt{pred : cond \rightarrow successor : probability}
  - \texttt{pred} = symbol of the shape to replace
  - \texttt{cond} = condition on shape parameters
  - \texttt{successor} = shape(s) to replace predecessor
Production Process

Shape = \{\text{symbol, geometry, numeric attributes}\} Symbol = \{\text{terminal \mid non-terminal}\}

Begin with configuration of shapes

1. Select an active shape $S$ from set.
2. Choose a production rule with $S$ as pred, compute a successor $S_{\text{NEW}}$, add to set.
3. Mark $S$ inactive.
4. Repeat until all active are terminal.
Example Rule
1. temple -> Subdiv(“Y”, ..., ...) { podium | columns | roof } □
2. column -> Subdiv(“Y”, ...) { base | shaft | capital }
3. columns -> Repeat(“X”, ...) { column } □
4. □base -> (corinthian_base) □
5. shaft -> (corinthian_shaft) □
6. capital -> (corinthian_capital)
7. podium -> (podium)
8. roof -> (roof)
Encoding Design

- Problems: numerous rules lead to chaos
- Need stylistic consistency to look intentional
- Solution: Control grammar to “clean up” after the first production process.

1: floors $\sim$ Repeat("Y",floor_height){ floor Snap("XZ") }
2: entrance $\sim$ Snap("Y","entrancesnap") door
3: floor $\sim$ Repeat("XS",tile_width){ tile }

<table>
<thead>
<tr>
<th>Repeat</th>
<th>Subdiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>default:</td>
<td></td>
</tr>
<tr>
<td>snapped:</td>
<td></td>
</tr>
</tbody>
</table>
In Action
Technical Requirements

- Application-specific customization
- Games are real-time!
- Solution: LODs
  - Textures
  - Polycount
  - Interiors
A Procedural City in Action
Houdini: A Procedural Animation Tool

- “Go Procedural”
- An extremely powerful effects and animation tool.
- Allows technical artists to “program” with a node-based system.
- Support for modeling geometry, fire, fluid, shading, etc.
- Houdini and technical artists in high demand!
Houdini: A Procedural Animation Tool

- “Go Procedural”
- An extremely powerful effects and animation tool.
- Allows technical artists to “program” with a node-based system.
- Support for modeling geometry, fire, fluid, shading, etc.
- Houdini and technical artists in high demand!
- Very recently getting into interactive graphics
  - Introduced Unity plugins for generating procedural content.
Houdini Demo
Creative Coding

- Large online community dedicated to procedural graphics (and music).
- Shadertoy, created by Inigo Quilez (CG extraordinaire), lets you program in GLSL to create images
  - IQ’s procedural snail
- Nop Jiarathanakul, a DMD and CGGT grad, programmed a particle simulation that can form interesting shapes
  - Nop’s particle simulation