Procedural Generation

Using code to generate virtual worlds

Programming – Computer Graphics

Contents

- Random
  - Pseudo-Random Numbers
- Perlin Noise
- Fractals
- Fault Generation
- Procedural Content
  - Pros and Cons

Random

- Random in Computer Science is very rarely “truly” random

- Instead randomness is formed by creating patterns of apparent randomness, based off an initial starting value
  - A seed value

Pseudo-Random Numbers

- These patterns are called Pseudo-Random as they are not truly random
  - If we used the same seed value then the results would be the same as before
  - If we know the seed ahead of time, and the way in which the random pattern is generated from the seed, then we could predict all of the results, and thus they aren't "random"
- Different seed values can give wildly differing results, or just slightly different results

Pseudo-Random Numbers Generators

- Pseudo-Random Number Generators (PRNGs) are methods of creating random values based off of a seed value

- There are many different methods
  - Some generate a long list of results from a single seed
  - Some need an initial seed to generate a value, and the next value is based off the previous result, and so on
  - Some need a seed value each time they are called and return a single result

```
// example PRNG
unsigned int getRandom( unsigned int seed0,
                        unsigned int seed1 )
{
    seed1 = 36969 * (seed1 & 65535) + (seed1 >> 16);
    seed0 = 18000 * (seed0 & 65535) + (seed0 >> 16);
    return (seed1 << 16) + seed0;
}
```

Perlin Noise

- Perlin Noise is a well-known PRNG that creates smooth “noise”
- Random 1-D, 2-D and 3-D patterns are formed where the seed values come from the location of the current sample
  - Values are smoothly interpolated with neighbouring samples creating a gradient

2-Dimensional Perlin Noise

Perlin Noise

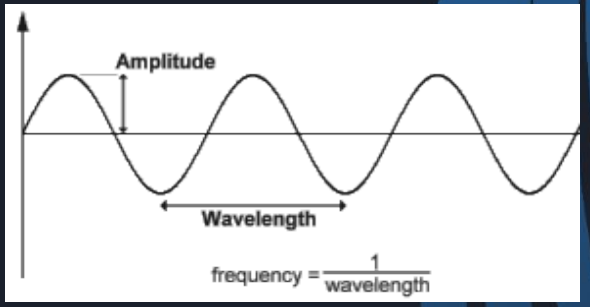
- Perlin Noise works by sampling “noise” and interpolating the results together
  - In 1 dimension, sampling for X we would interpolate between X-1 and X+1, using X as a scale between X-1 and X+1 for interpolation
    - As an example, if X was a value **3.7**, we could sample noise at **3.0** and **4.0**, then use **0.7** as the interpolation scale between the noise of **3.0** and **4.0**

- Noise functions are a form of PRNG that returns a single result from a seed
  - The following is an example 1-D noise function that returns values in the range [-1.0,1.0]:

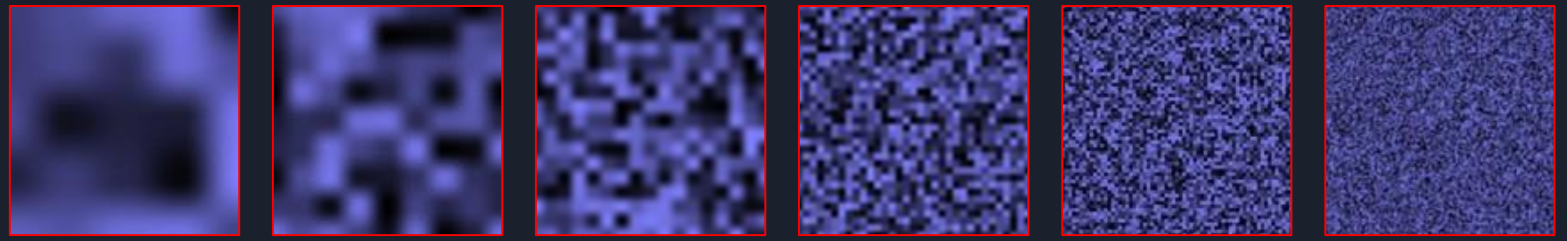
```
double noise( int x )
{
    x = pow(x << 13, x);
    return ( 1.0 - ((x * (x * x * 15731 + 789221) + 1376312589) & 0x7fffffff)/ 1073741824.0);
}
```

Perlin Noise

- Typically Perlin Noise is generated multiple times, with the sample location X scaled by a Frequency, and the returned result scaled by an Amplitude
  - Frequency controls the gradient changes
  - Amplitude controls the strength of the gradient



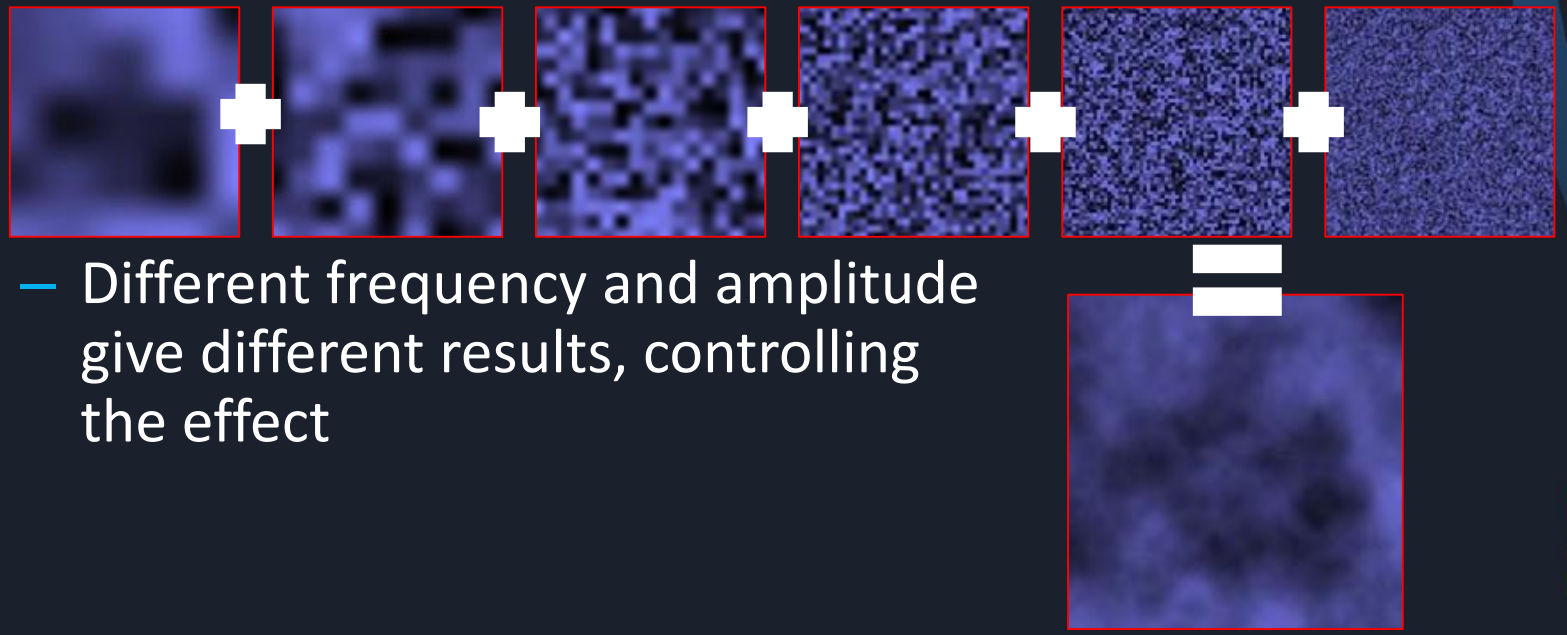
- The following are different 2-D Perlin Noise results with various Frequency and Amplitude:



$$h = perlin(\ x\ \times f\ ) \times a$$

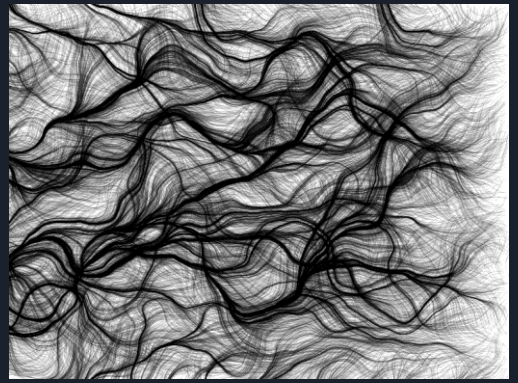
Perlin Noise

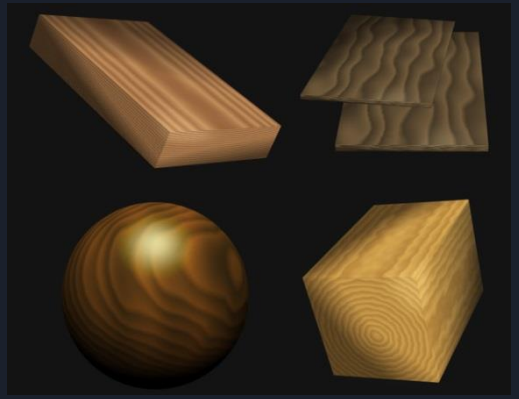
- The results can be summed together to create a smooth noise pattern that has a softer gradient:

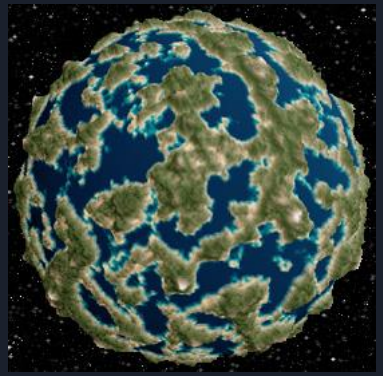


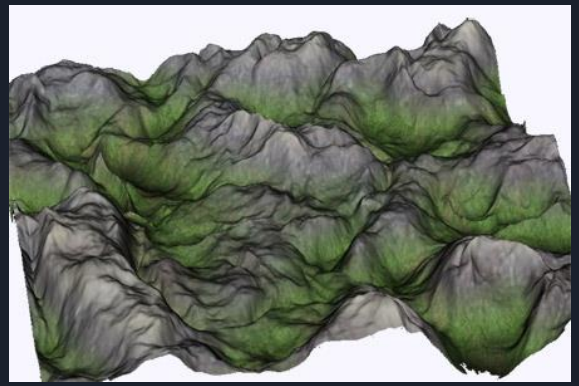
  - Different frequency and amplitude give different results, controlling the effect

Perlin Noise Uses









aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

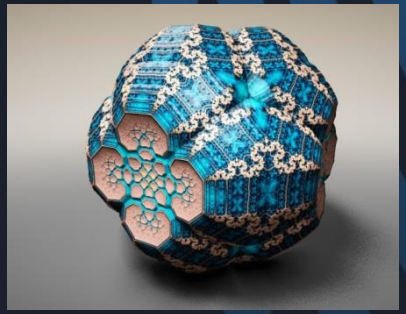
Terrain Generation

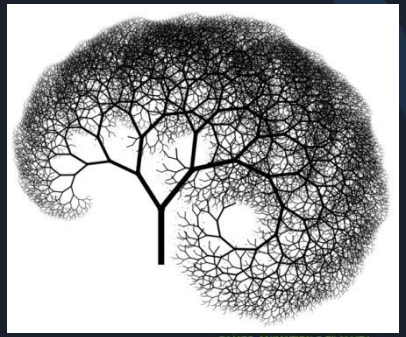
- Terrain is one of the most well-known uses of noise algorithms
  - The noise creates a height-field that represents the height of the terrain at each location, in the case of 1-D or 2-D height-fields
  - 3-D noise can create “Caves”

- But noise is not the only way to generate terrain

Fractals

- Fractals are a form of mathematical equation that can create repeating patterns





Fractals

- We can use Fractal equations to generate landscapes
  - Create a height-field with a pattern rather than noise
  - Can also be used to create plants and vegetation

- A common Fractal for creating height-fields is the Diamond-Square Algorithm, sometimes referred to as Midpoint-Displacement

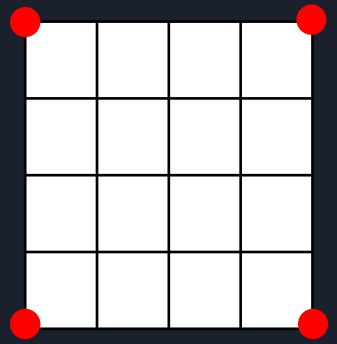
Diamond-Square Algorithm

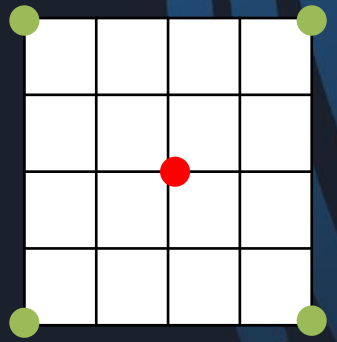
- Diamond-Square works on a 2-D grid of vertices the size of a power-of-2 + 1
  - 33x33, 513x513 etc

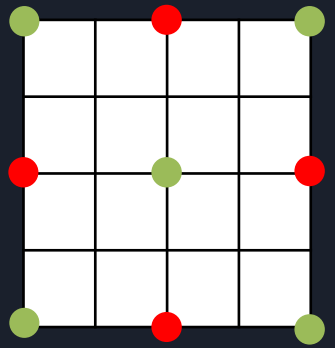
- The algorithm recursively sub-divides the grid, assigning height values to certain points
  - It has two steps to it…

Diamond-Square Algorithm

- First we seed the grid corners
  - Can all be the same, or seeded to control the initial shape of the terrain
- We then generate the point between the 4 corners by averaging the corners then adding a random offset
- Then we generate the point along each edge made by the 4 corners by averaging the 2 corners along that edge, and the center point, and add a random offset

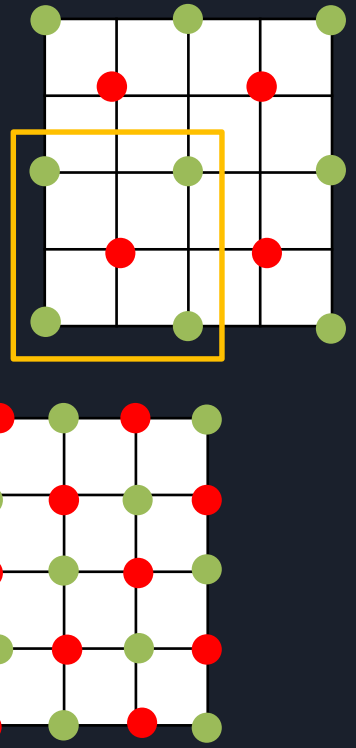




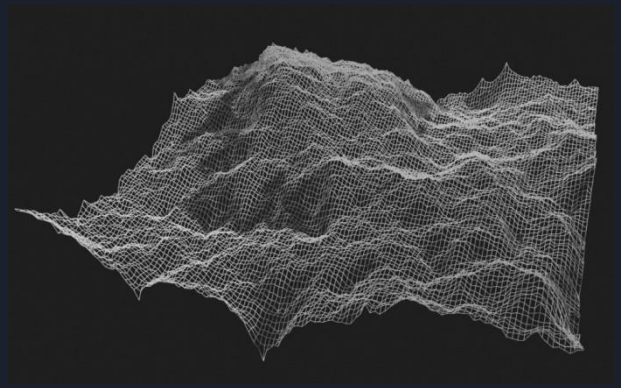


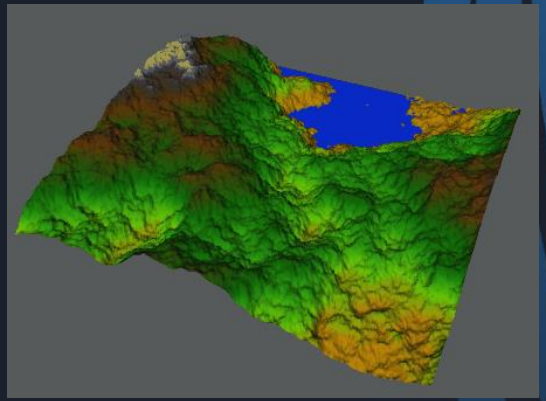
Diamond-Square Algorithm

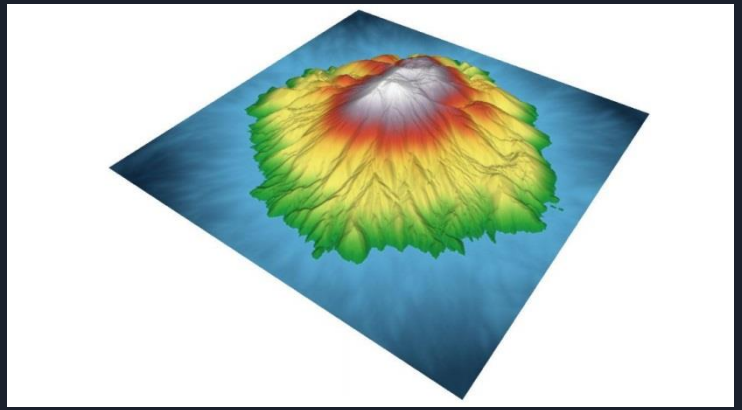
- We then repeat the first steps, this time on the 4 new squares that have been formed by the previous iteration
  - Each iteration we also reduce the amount of random offset that is added after averaging the points
- This pattern repeats until all points have been set
- Can be adapted to 1-D and 3-D



Diamond-Square Algorithm



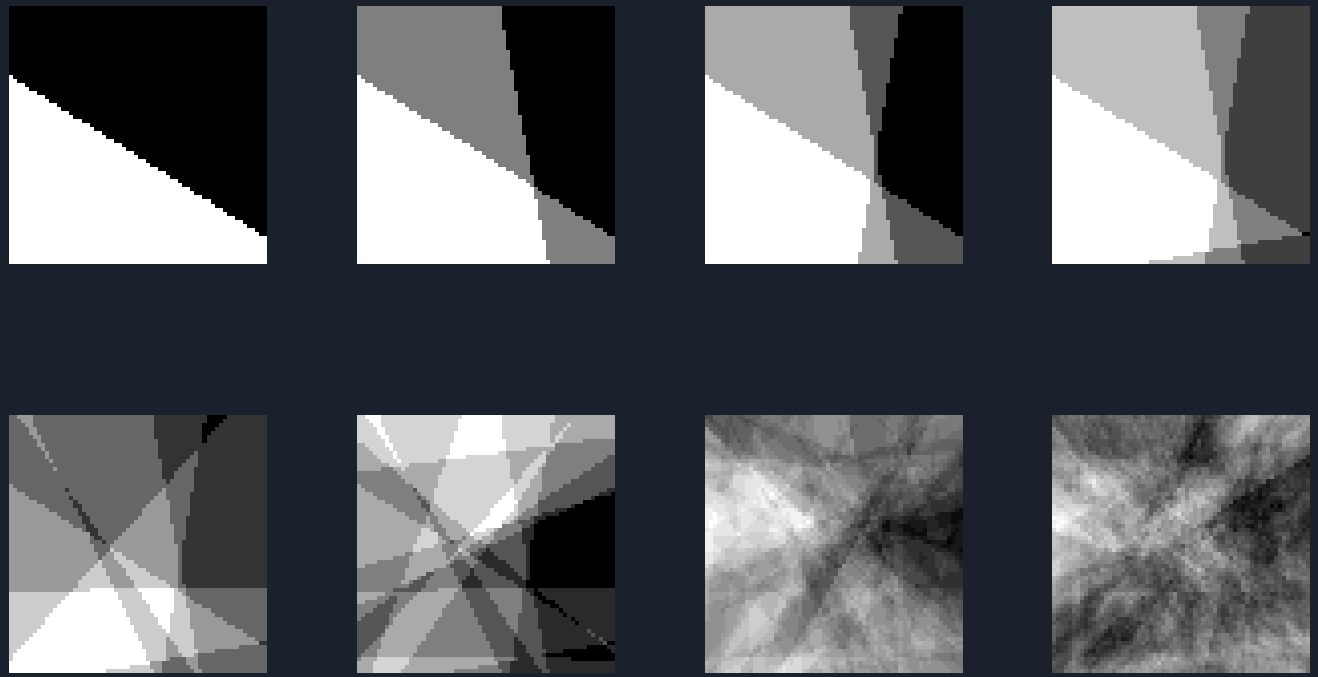




Fault Generation

- Another technique for generating a height-field is Fault Generation
- In Fault Generation we simulate the effect of fault lines and tectonic plates
  - Sections of land push against each other along a fault line, with one side pushing up while the other pushes down
- In Fault Generation we iterate many times, splitting the terrain in two sides
  - On one side we raise all the heights slightly
  - On the other side we can leave them or lower them

Fault Generation

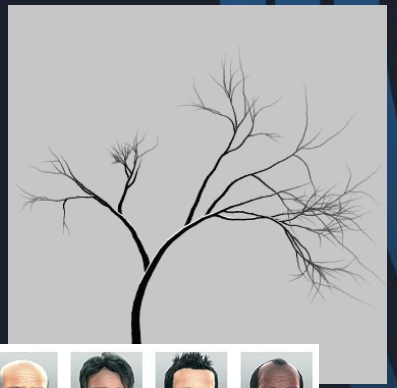


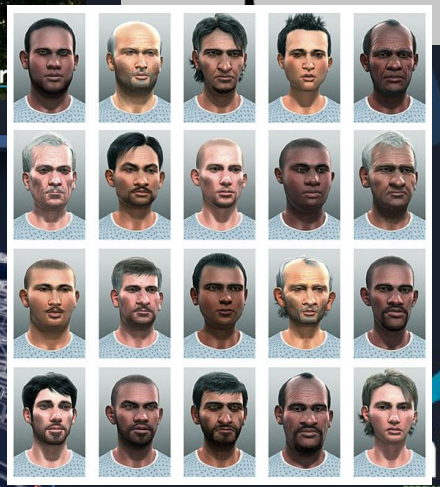
Procedural Content

- All of the previous methods are forms of procedural content
  - Content that isn't first created by an artist, but is instead procedurally generated via a combination of algorithms
- Procedural Content isn't just limited to Terrain Generation

Uses for Procedural Content:

- Terrain / Environment
- Characters
- Objects in the world
  - Trees
  - Buildings / Cities
- Textures and Animations
- Gameplay





Randomised Tiles

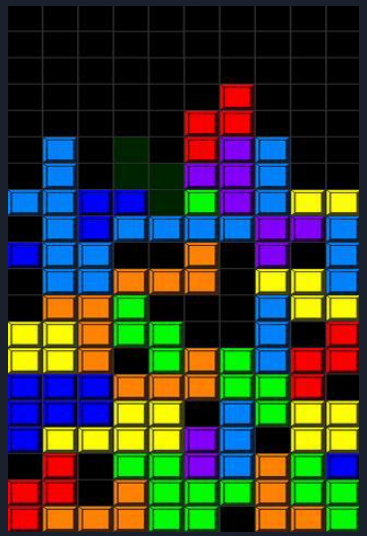
- We could randomised our levels by making them out of interlocking tiles
  - Can work in both 2-D and 3-D games
- Each tile has a certain type of link that must match up to a tile with a similar link
  - A door must link to a tile with another door, for example

Real-time Level Creation

- We could generate tiles as we come to them rather than generate the whole level on load
  - Based on difficulty, time played, time since that tile appeared, etc

Other Examples

- And depending on the game they don’t actually have to be levels that are randomly strung together:



Pros and Cons

Advantages

- More content!
- Can increase the replay value of games
- Less time spent on content development
- Unique looking game objects
- Reduces size of game content: Great for downloading and streaming online games

Disadvantages

- Game design and game-play issues:
  - Random is unpredictable and not always interesting
- Unrealistic looking environments
- Repetitive game objects:
  - Bad random can be bland and repetitive
- Generation time
- Tech can be very complicated and require significant up-front investment

Summary

- Programmers can be designers too!
- Algorithms can be used to generate all kinds of content
- Many successful and addictive games are built off pseudo-random systems

Further Reading

- Random.org, Introduction to Randomness and Random Numbers, https://www.random.org/randomness/
- Wikipedia, Perlin Noise, https://en.wikipedia.org/wiki/Perlin_noise
- Nelson, M, Shaker, N & Togelius, J, Procedural Content Generation in Games, http://pcgbook.com/
- Biagioli, A, Understanding Perlin Noise, http://flafla2.github.io/2014/08/09/perlinnoise.html