Height Maps, Terrain, Noise, Erosion and Vegetation

Overview

- Height maps
- Terrain
- Noise algorithms
- Erosion Algorithm
- Vegetation Modelling

Height Map

- Heightmap is a grey scale image.
- When we read this grey-scale image in, each pixel represents a vertex.
- We can hence use an image to store the heights (y values) for our terrain mesh.
  - For example:

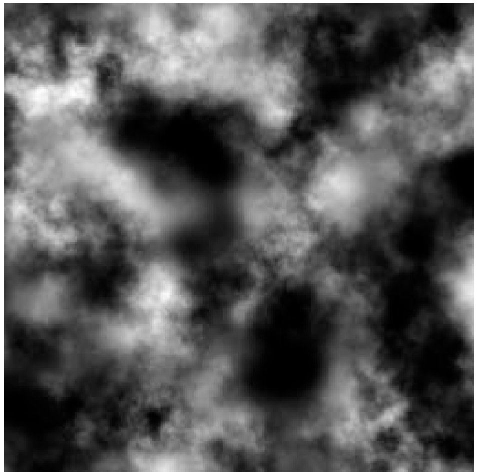


Fig. 1: Example height map with height displayed as brightness.

Terrain

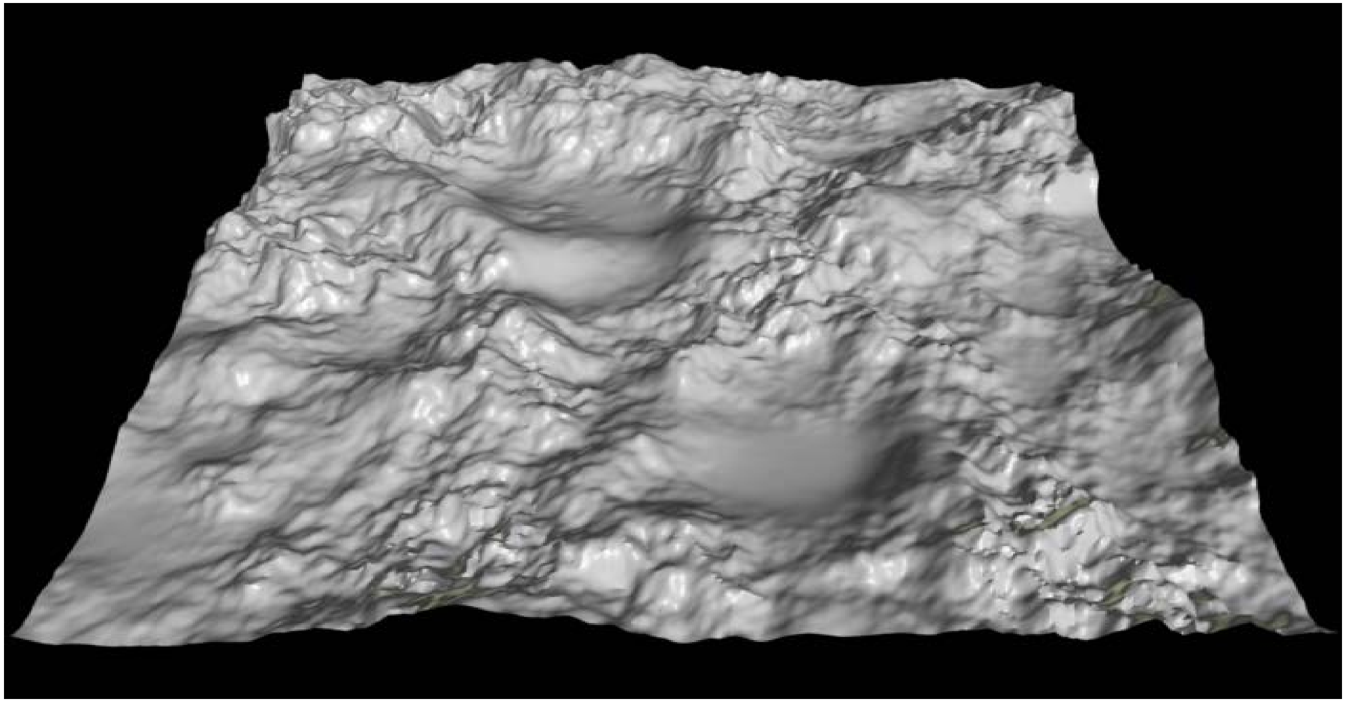


Fig. 2: Height map from Fig.1 converted into a 3D mesh.

Tools

- Height Map continued…
  - 256 x 256 pixels
  - Grey scale:
    - Every pixel has a value from 0 to 255.
    - R,G,B components are all equal.
- Tools
  - Terragen
    - Freeware
    - Can be used to generate height maps.
    - http://www.planetside.co.uk/
  - Photoshop
    - We can easily create our own…

Tools continued

- Bryce http://www.thebest3d.com/bryce/index.html
- Dark Tree http://www.darksim.com/

These tools have many procedural algorithms for generating height-maps and also have built-in height-map editors

Heightmap Generation

- After finishing drawing the heightmap,it needs to be saved in an 8-bit RAW file.
- RAW files simply contain the bytes of the image one after another
- This makes it easy to read the image into the program
- If any software asks to save the RAW file with a header ,specify no header

Heightmap Smoothing

- One of the problems of using an 8-bit heightmap is that it means we can only represent 256 discrete height steps .
- The truncation creates a “rougher” terrain than what may have been intended
- Once we truncate , we cannot recover the original height values but we can smooth the values

Heightmap Smoothing

- So we load the height-map into memory by reading raw bytes
- We copy the byte array into float array so that we have a floating-point precision
- Then we apply a filter to the floating-point heightmap
- This helps to smooth the heightmap out

i-1,j-1	**i-1,j**	**i-1,j+1**
i,j-1	i,j	i,j+1
i+1,j-1	i+1,j	i+1,j+1

Noise

Noise

- In the everyday world, noise is a naturally occurring nuisance that is generally covered up as much as possible
- However, in the field of Computer Science, especially 3D modelling, noise has become increasingly useful
- Irregular bumps and nicks make 3D models look much more realistic, but are difficult and time consuming to make by hand
- Noise algorithms create pseudo-random textures quickly with little or no interaction required from the user.

Use of Noise in Games

- In games, noise algorithms are used to generate landscapes
- Generation of these landscapes often doesn't stop with noise, often erosion, vegetation, and water models are applied to increase realism

Types of Noise Algorithms

- Mid-point displacement
- Diamond-square
- Value noise
- Perlin noise
- Simplex noise
- Cell/Whorley noise
- Voronoi noise
- Alligator noise
- Space –convolution noise

Basic noise algorithms

- Mid-point displacement
- Diamond Square noise
- Perlin noise

Mid-point displacement algorithm

- The Mid Point Displacement, aka the Plasma Algorithm, is a subdivision algorithm
- The terrain is built iteratively, in each iteration the level of detail increases
- This algorithm was conceived to generate square terrains with dimensions (2^n + 1) x (2^n+1) where *n* stands for the number of iterations

In 1 Dimension

```
Start with a single horizontal line segment.
  Repeat for a sufficiently large number of times
     {
        Repeat over each line segment in the scene
           {
              Find the midpoint of the line segment.
              Displace the midpoint in Y by a random amount.
              Reduce the range for random numbers.
           }
     }
```

In 1 Dimension

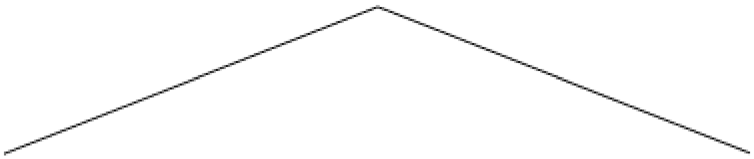


Fig 2 First Iteration

Fig 3 Second Iteration

Fig 4 Third Iteration

Fig 5 Final Iteration

Diamond –square algorithm

- This is midpoint-displacement method in 2-dimension
- Assign a height value to each corner of the rectangle
- Divide the rectangle into 4 sub rectangles, and let their height values be the mean values of the corners of the parent rectangle.
- When computing the middle height, one should add a small error that depends on the size of the rectangle (the standard is to let the error be proportional to the size of the rectangle and some constant.
- The constant controls the "roughness" of the fractal
- A bigger constant results in more valleys and mountains.
- Iterate and subdivide each rectangle into smaller ones.

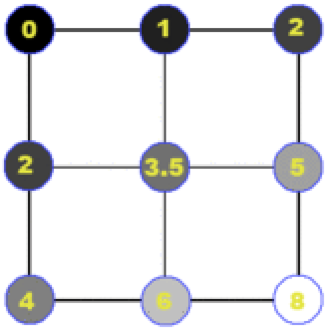
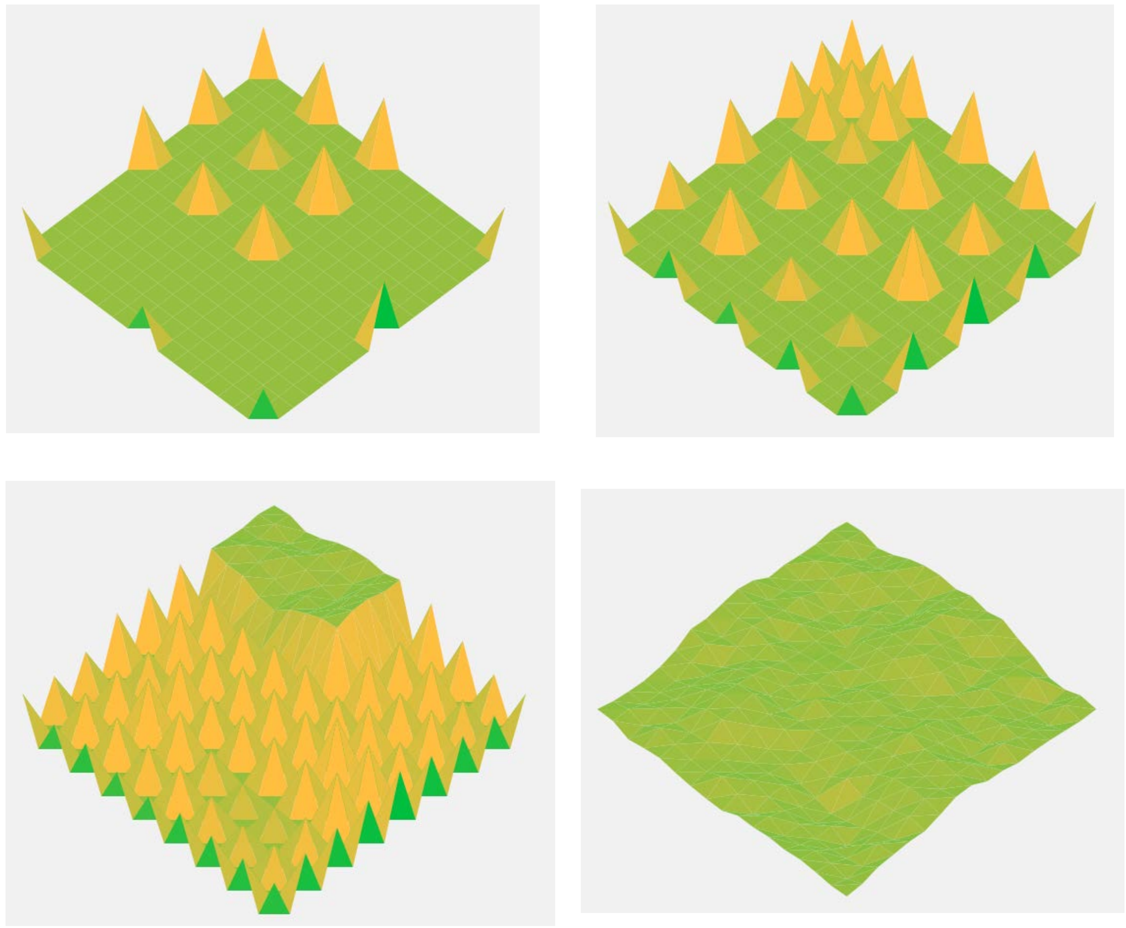
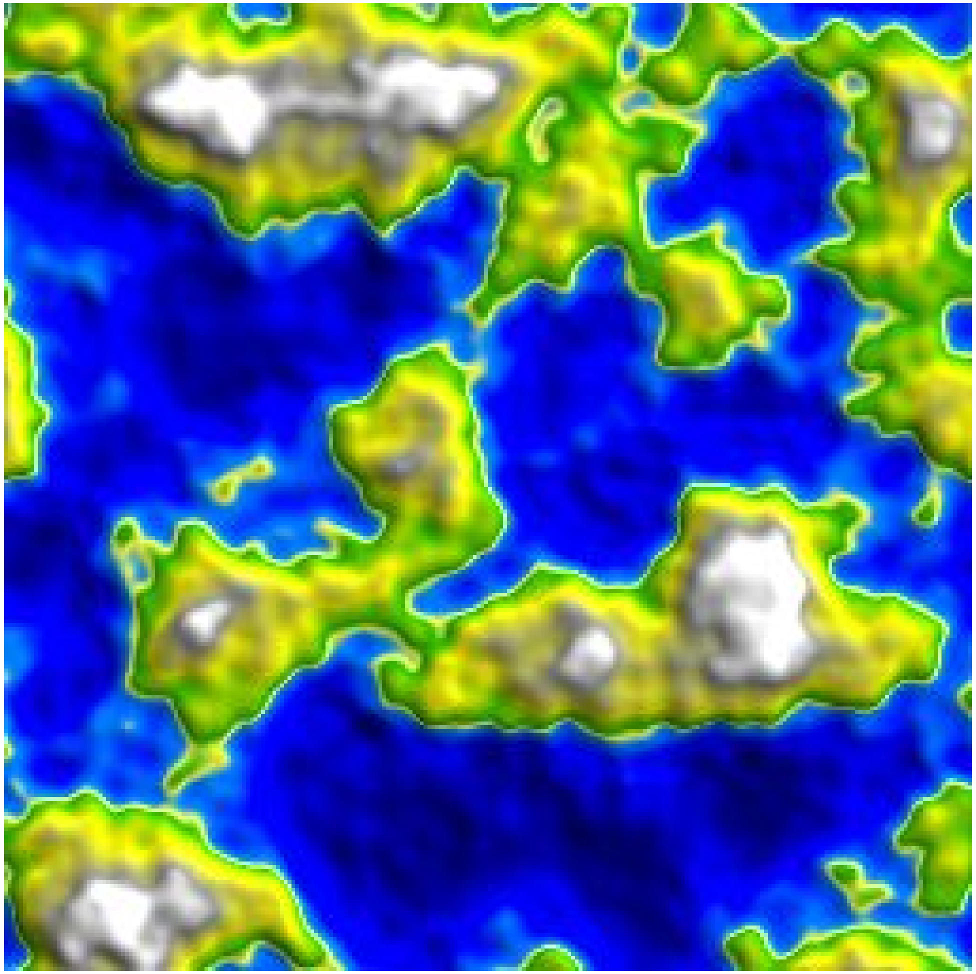


Fig 6 Diamond square algorithm

Diamond –square



Perlin Noise



Perlin Noise

- Perlin Noise is an extremely powerful algorithm that is used often in procedural content generation
- The man who created it, Ken Perlin, won an academy award for the original implementation
- In game development, Perlin Noise can be used for any sort of wave-like, undulating material or texture.
- For example, it could be used for procedural terrain, fire effects, water, and clouds

Process

- Generate Noise
  - For each point in width and height of texture
  - Generate random number
  - Smoothen the value
  - Interpolate
- Apply perlin noise algorithm per point.
- Get a gray scale value per point similar to height map.

Generic noise function

- Noise Function
  - A noise function is essentially a seeded number generator.

```
float random(int x, int y) {

    int n = x + y * 57;
    n = (n << 13) ^ n;
    int t = (n * (n * n * 15731 + 789221) + 1376312589) & 0x7fffffff;
    return 1.0 - double(t) * 0.931322574615478515625e-9;
}
```

Smoothen the noise

- Smooth Noise
- Smoothens the value by averaging the corners, sides and center.

```
float smooth(int x, int y)

   float corners;
   float sides;
   float center;

   corners = ( random(x-1, y-1)+random (x+1, y-1)+random (x-1, y+1)+random (x+1, y+1) ) / 16
   sides   = (random(x-1, y)+random (x+1, y)+random (x, y-1)+random (x, y+1) ) /  8
   center  =  random (x, y) / 4
   return corners + sides + center

}
```

Interpolation of noise

- Interpolation – Can be linear, cubic, cosine

```
float interpolate(float a, float b, float x) {

        return a*(1 - x) + b*x;
}
```

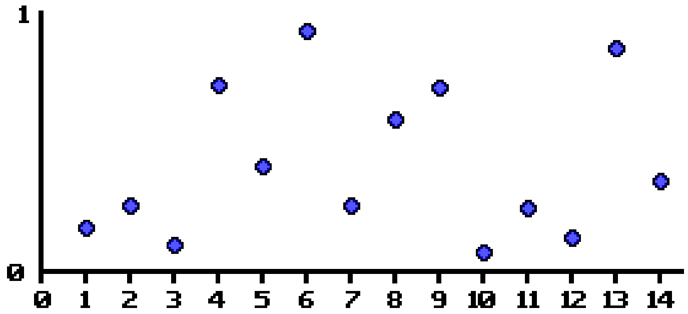


Fig 7 Random numbers plotted

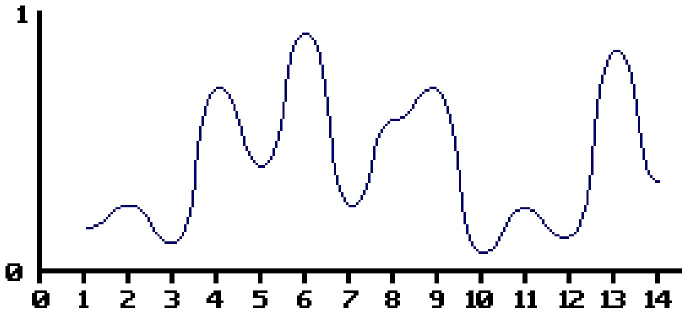


Fig 8 Interpolation

MEDIA DESIGN SCHOOL GAME DEV

Perlin Noise

- function **Linear_Interpolate**(**a**, **b**, **x**)

  return **a***(**1**-**x**) + **b***x**

  end of function
- function **Cosine_Interpolate**(**a**, **b**, **x**)

  ft = **x** * **3.1415927; f** = (**1** - cos(**ft**)) * **.5**

  return **a***(**1**-**f**) + **b***f

  end of function
- function **Cubic_Interpolate**(**v0**, **v1**, **v2**, **v3**,**x**)

  P = (**v3** - **v2**) - (**v0** - **v1**)

  Q = (**v0** - **v1**) - **P**

  R = **v2** - **v0**

  S = **v1**

  return $Px^3 + Qx^2 + Rx + S$

  end of function

Averaged and Interpolated Noise

```
float noise(float x, float y) {

float fractional_X = x - int(x);
float  fractional_Y = y - int(y);

//smooths
float v1 = smooth(int(x), int(y));
float v2 = smooth(int(x) + 1, int(y));
float v3 = smooth(int(x), int(y) + 1);
float v4 = smooth(int(x) + 1, int(y) + 1);

// interpolates
float i1 = interpolate(v1, v2, fractional_X);
float i2 = interpolate(v3, v4, fractional_X);

return final = interpolate(i1, i2, fractional_Y);

}
```

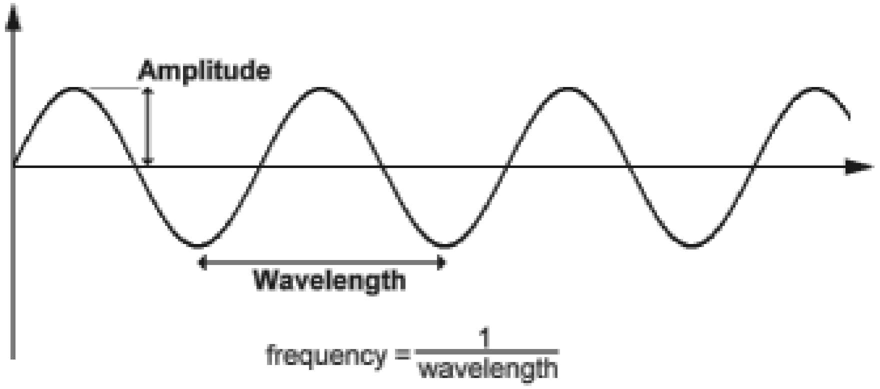
Terminologies

- Octaves
  - Each successive noise function you add is known as an octave
- Persistence
  - A multiplier that determines how quickly the amplitudes diminish for each successive octave in a **Perlin-noise** function.
- Amplitude
  - The maximum extent of a vibration or oscillation, measured from the position of equilibrium.
  - amplitude = $persistence^i$

Terminologies

- Frequency
  - The distance between successive crests of a wave
  - Especially points in a sound wave or electromagnetic wave
  - frequency = 2^i

- i is the i^{th} noise function being added for each octave
- i might range from 0 to 8

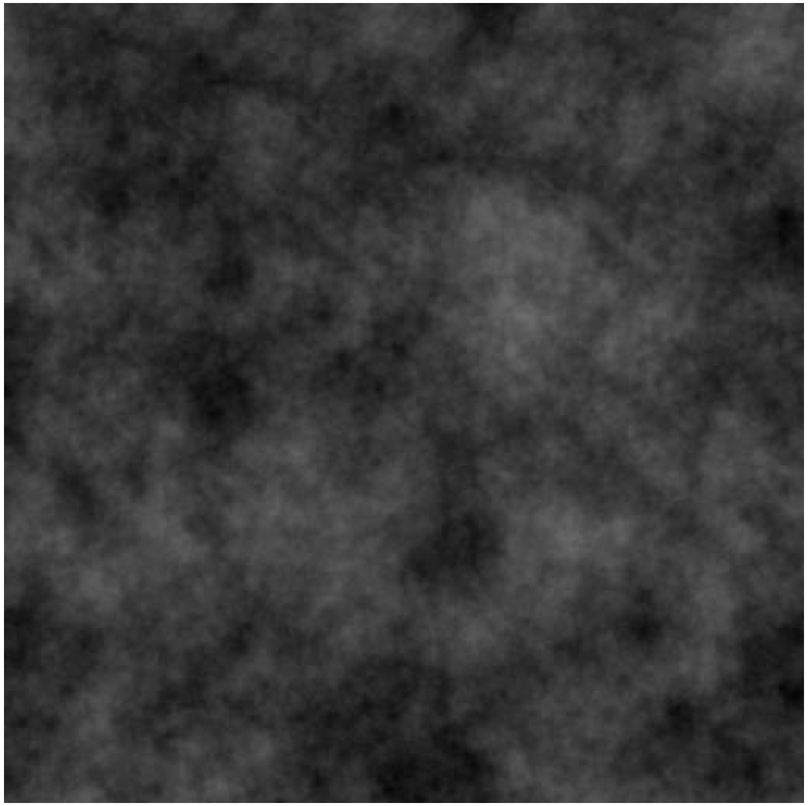


Total Noise Per Point (x, y)

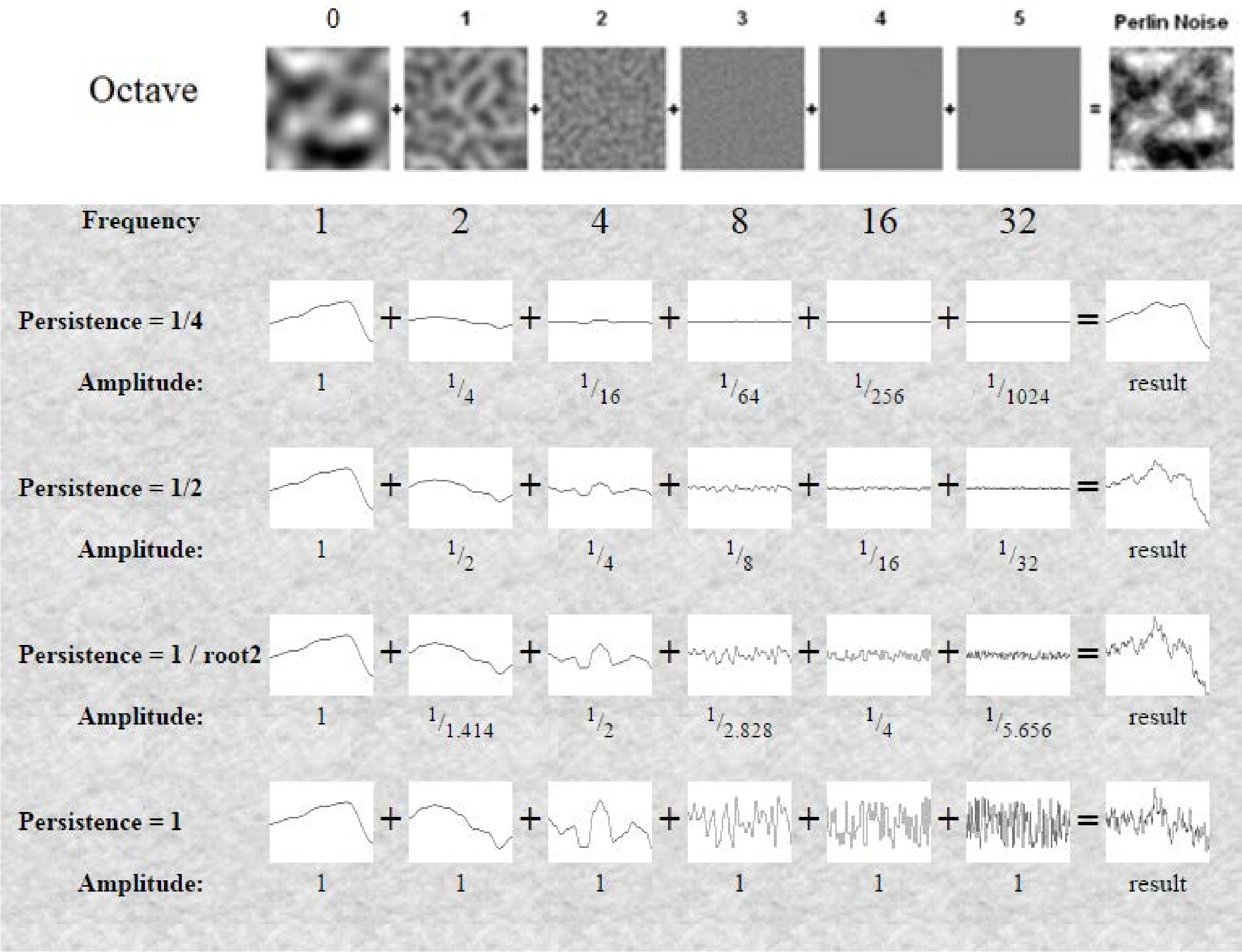
```
float totalNoisePerPoint(int x, int y){

int octaves = 8;
float zoom = 20.0f;
float persistence = 0.5f;
float total = 0.0f;

        for (int i = 0; i < octaves - 1; i++) {

        float frequency = pow(2, i)/zoom;
        float amplitude = pow(persistance, i);

        total +=  noise(x * frequency, y * frequency) *  amplitude;
        }
return total;
}
```

Grayscale image

- For each point x, y generate a greyscale value.



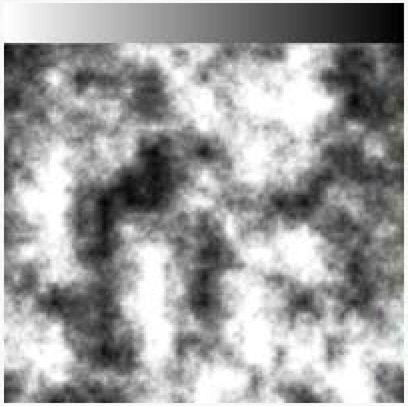
Perlin Noise



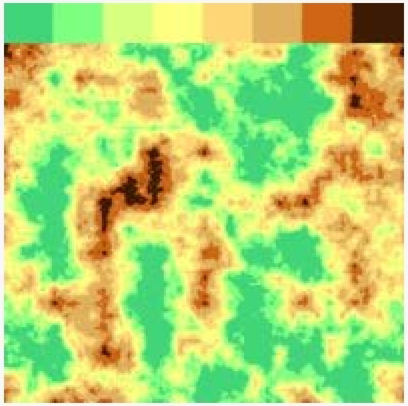
Perlin Noise Application

Applications

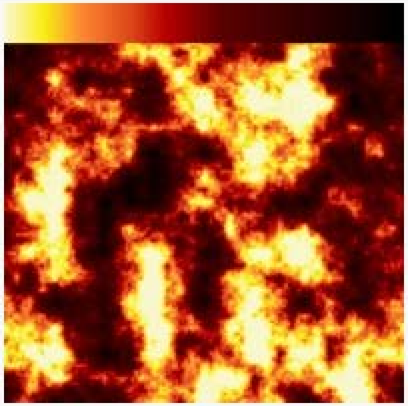
- Color Gradiant



Grayscale gradient



Gradient with discrete colours



Fire gradient

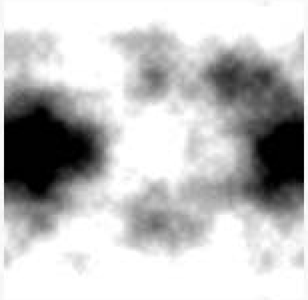
MEDIA DESIGN SCHOOL GAME DEV

Applications

- Texture Blending


Image 1

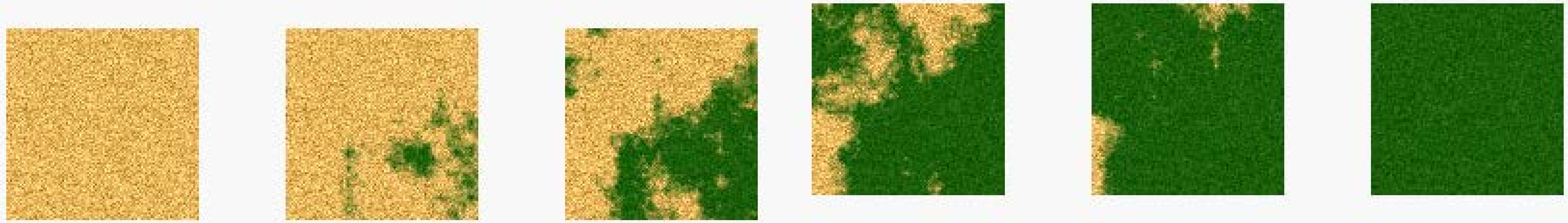

Image 2


Perlin noise

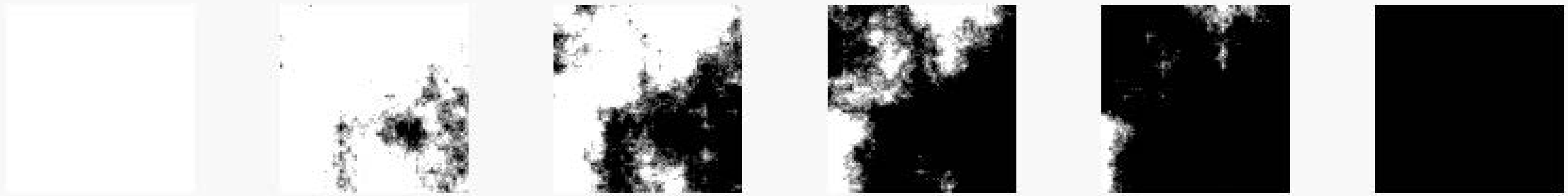

Blend using noise

Applications

- Realtime Transition

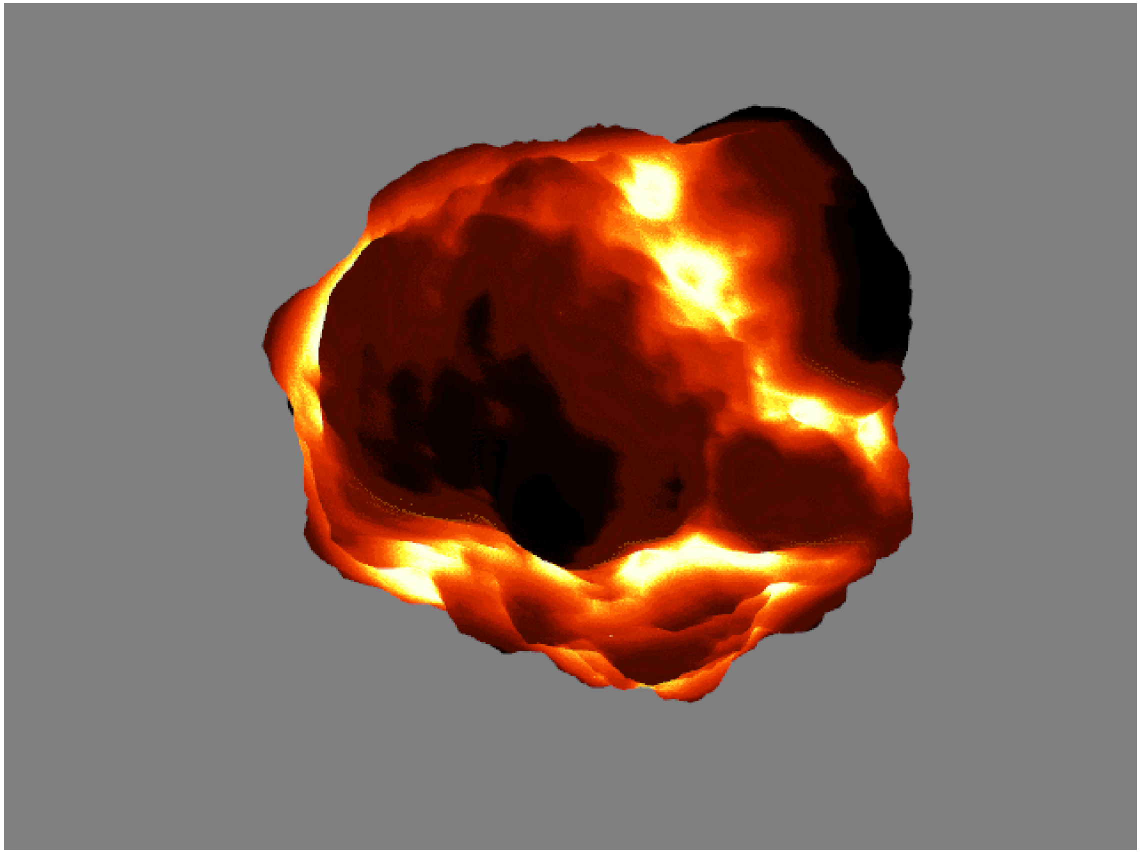


Real time transitions using Perlin noise



Blend textures

3D Perlin Noise



Erosion Algorithms

- Noise alone can create interesting landscapes, but erosion can help add an extra layer of realism.
  - Thermal Erosion
  - Hydraulic Erosion
  - Inverse Thermal Erosion

Thermal Erosion

- Thermal erosion models gravity eroding cliffs that are too steep
- If the angle is too sharp, soil will fall to a lower area.
- Thermal erosion is fairly simple to model
- First, define the difference T, which is the maximum difference allowed before gravity takes over

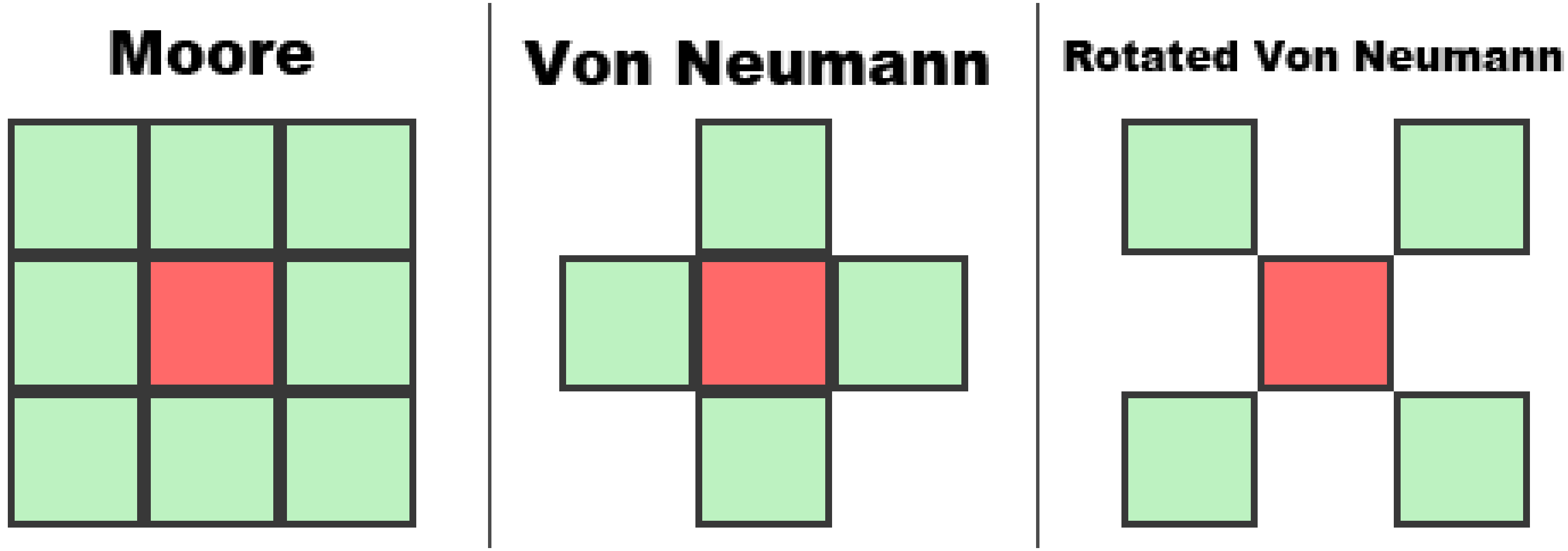
Thermal Erosion Algorithm

- For all pixel
- Get the difference in height between this pixel and the neighbouring pixel
- If the difference is greater than T, remove some amount of soil from the taller pixel and deposit it into the lower pixel.
- The speed of this algorithm can be improved further by changing the neighbourhood type.
- The three standard type are the
  - Moore neighbourhood,
  - the Von Neumann neighbourhood,
  - and the rotated Von Neumann neighbourhood

Neighbourhood Types

- In cellular automata, the Moore neighbourhood comprises the eight cells surrounding a central cell on a two-dimensional square lattice
- In cellular automata, the von Neumann neighbourhood comprises the four cells orthogonally surrounding a central cell on a two-dimensional square lattice.
- While the Moore neighborhood provides the best results, it is also the slowest. The rotated Von Neumann neighborhood gives good results while increasing speed.

Von-Neumann Neighbourhood Type



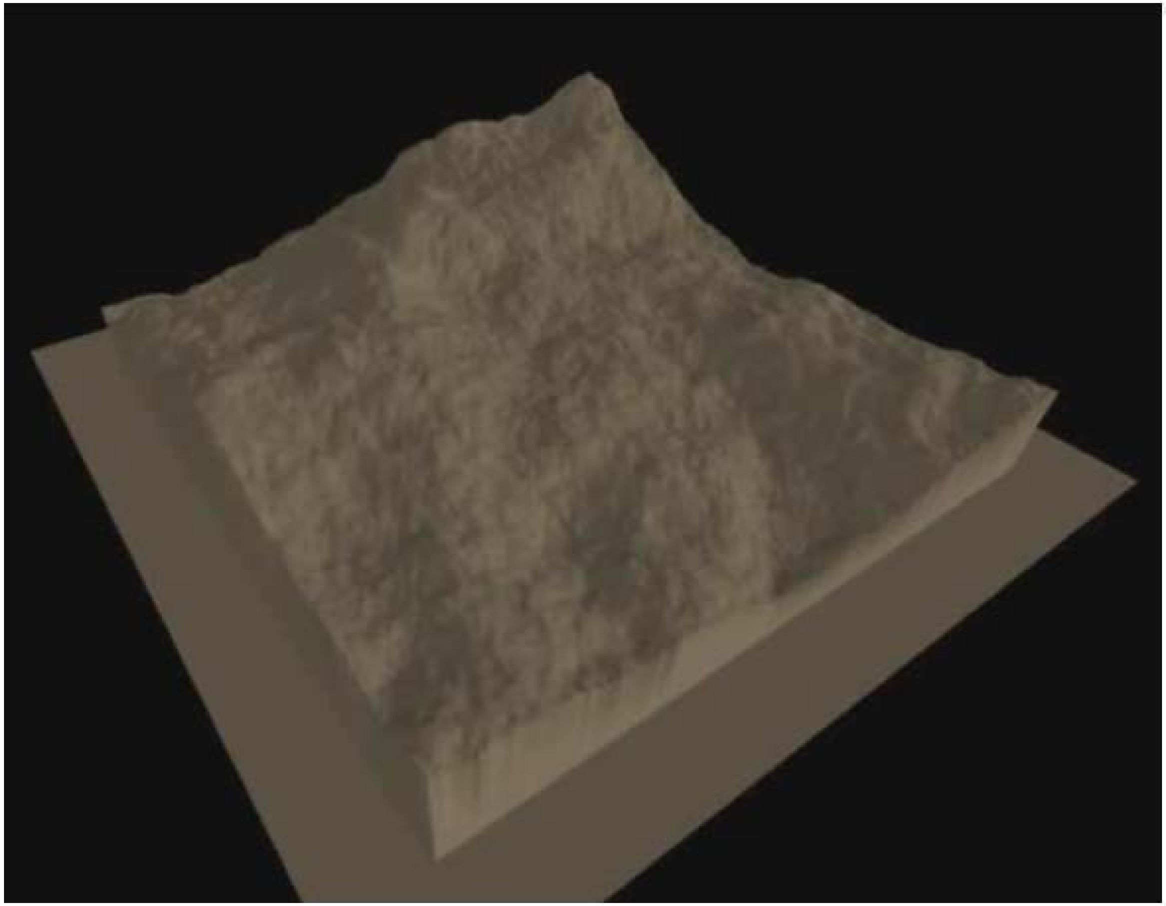
Hydraulic Erosion

- Hydraulic Erosion models rainwater picking up soil, washing down into basins, then evaporating and depositing soil.
- Hydraulic erosion provides good quality results, but is very slow.
- It may use up to 3 times the memory of Thermal Erosion.

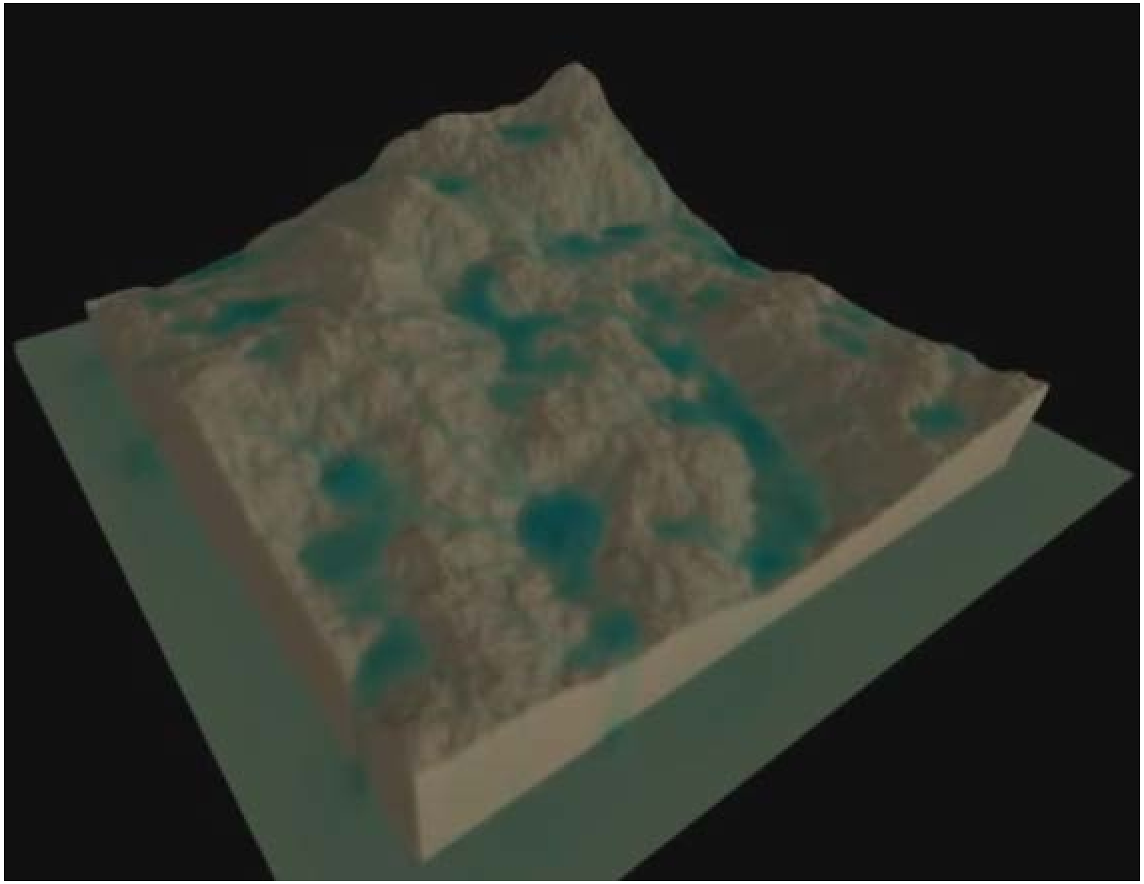
Inverse Thermal Erosion

- For gaming, plateaus and cliffs are much more desirable than rolling hills.
- Thermal Erosion destroys cliffs and rarely creates plateaus.
- By flipping the erosion condition the opposite effect can be created.

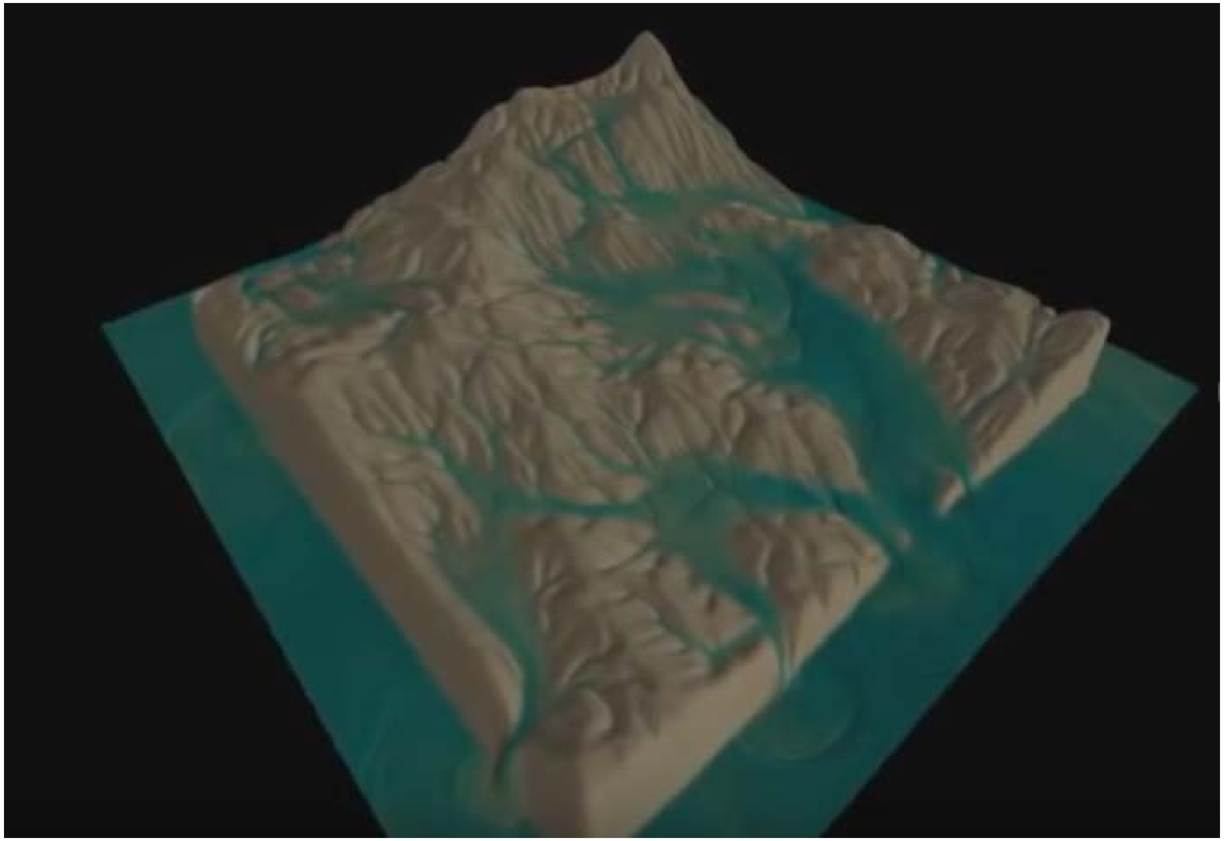
Hydraulic erosion



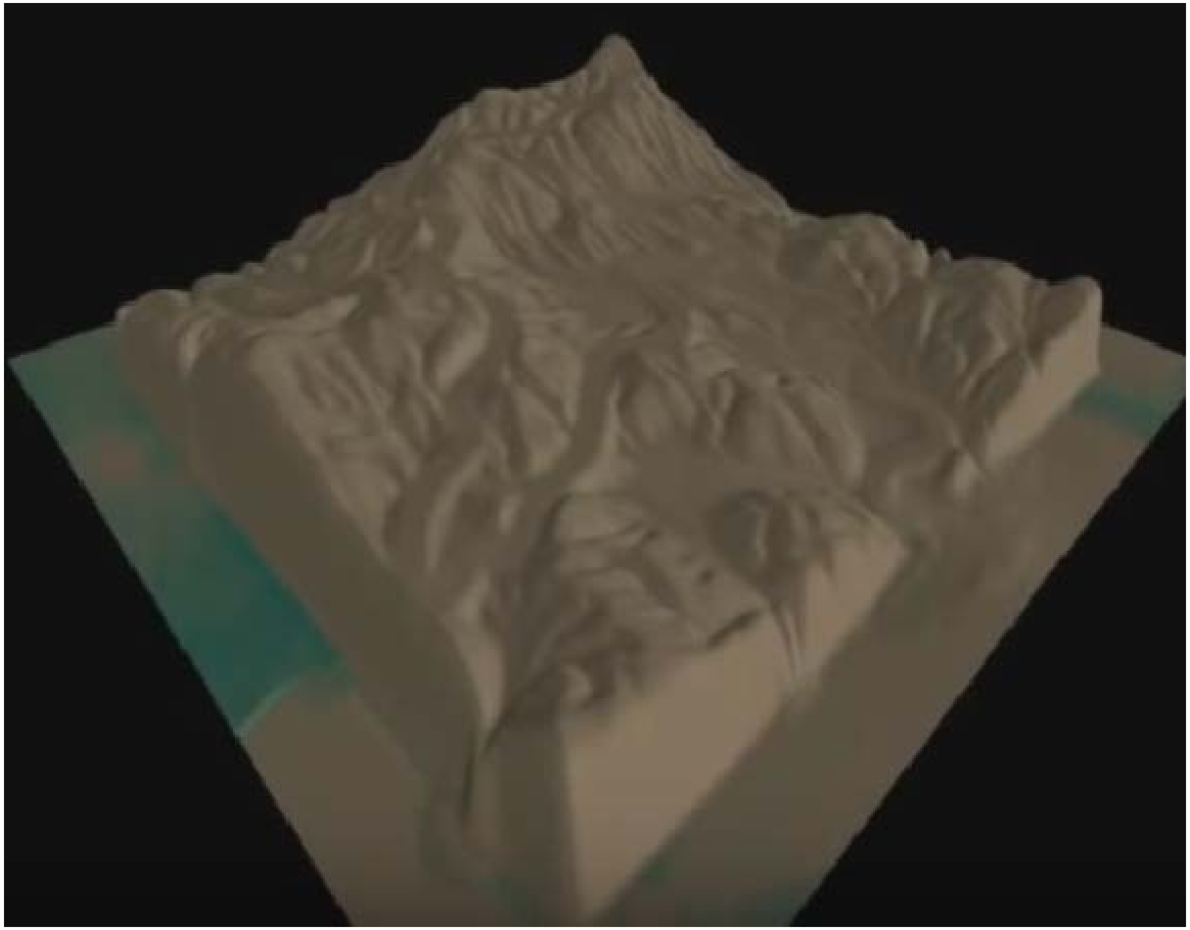
Hydraulic erosion



Hydraulic erosion



Hydraulic erosion



Vegetation Modelling

- While noise and erosion may create realistic features, without color the terrain is completely alien to the eye.
  - Simple Colour Mapping
    - Portions of height to colors or gradients of colors
  - Terrain Mandated
    - Waterflow is defined by terrain height.
    - Vegetation takes slope and height into account
  - Dynamic Modelling
    - Dynamic systems such as rainfall and water-flow could control where vegetation grows

References

- https://www.clicktorelease.com/blog/vertex-displacement-noise-3d-webgl-glsl-three-js/
- http://libnoise.sourceforge.net/tutorials/tutorial4.html
- http://flafla2.github.io/2014/08/09/perlinnoise.html
- http://devmag.org.za/2009/04/25/perlin-noise/
- http://paulboxley.com/blog/2011/03/terrain-generation-mark-one
- http://web.mit.edu/cesium/Public/terrain.pdf
- http://old.cescg.org/CESCG-2011/papers/TUBudapest-Jako-Balazs.pdf
- http://micsymposium.org/mics_2011_proceedings/mics2011_submission_30.pdf
- https://www.reddit.com/r/gamedev/comments/1rl0vs/realtime_hydraulic_erosion_simulation_on_gpu/?st=j6bestag&sh=d83b301c