VGP352 – Week 8

- Agenda:
  - Noise
  - Noise based procedural textures
  - Wang tiles

Brief history of noise

- Developed by Ken Perlin in the early 80s
  - Ken worked on the revolutionary graphics for the movie *Tron*
  - Frustrated that everything in *Tron* looked so “machine-like,” he wanted to get out of the "machine-look ghetto."

Brief history of noise

- Developed by Ken Perlin in the early 80s
  - Ken worked on the revolutionary graphics for the movie *Tron*
  - Frustrated that everything in *Tron* looked so “machine-like,” he wanted to get out of the "machine-look ghetto."
- *Tron* was *not nominated* for the Academy Award for Special Effects because it “cheated” by using computers
  - What movie won?

Brief history of noise

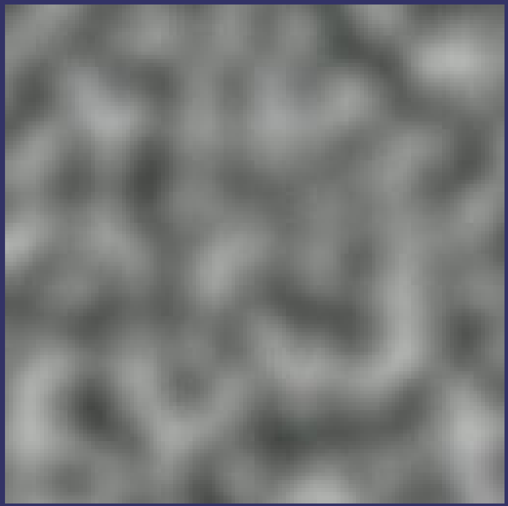
- Developed by Ken Perlin in the early 80s
  - Ken worked on the revolutionary graphics for the movie *Tron*
  - Frustrated that everything in *Tron* looked so “machine-like,” he wanted to get out of the "machine-look ghetto."
- *Tron* was *not nominated* for the Academy Award for Special Effects because it “cheated” by using computers
  - What movie won?
  - *E.T. the Extra Terrestrial*
    - Defeating *Blade Runner* and *Poltergeist*

Brief history of noise

- In 1983 Perlin worked on creating a space filling, apparently random signal function
  - Needed to appear random
  - Needed to be controllable
  - Needed all the features to be approximately the same size
  - Needed all the features to be roughly isotropic
  - Needed to have a range [-1, 1]
- First presented as a course at SIGGRAPH '84
  - The paper followed at SIGGRAPH '85
  - The Academy Award for Technical Achievement followed in 1997

Using Noise

- In Perlin's words, “noise is salt for graphics.”
  - Salt by itself is boring
  - Without salt, food is boring too



Original image from http://en.wikipedia.org/wiki/Perlin_noise

Using Noise

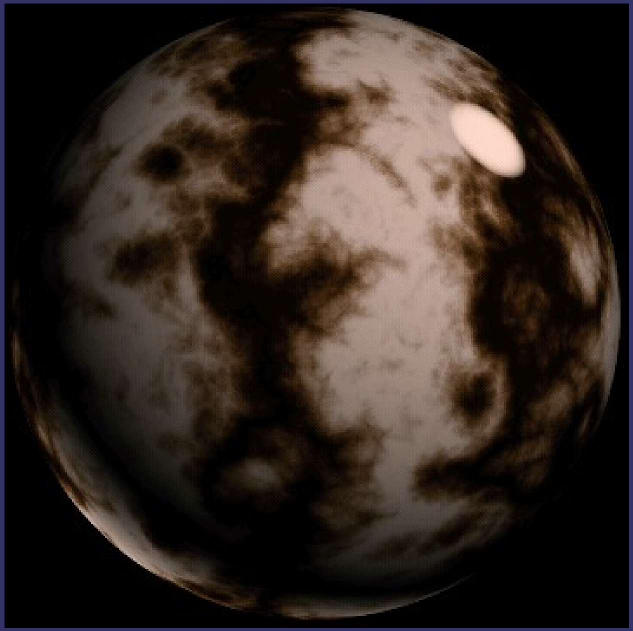
- Noise is typically used in multiple frequencies
  - Each frequency band is called an *octave*
  - As octave frequency increases, the amplitude decreases

$$NOISE(p) = \sum_{i=0}^{N-1} \frac{noise(f_i p)}{a_i}$$

Using Noise

- Add noise to boring functions or textures to make them interesting
  - Marble is the *classic* example

$$\sin\left(x + \left|NOISE\left(y\right)\right|\right)$$



Original image from http://www.noisemachine.com/talk1/23.html, copyright Ken Perlin

Implementing Noise

- Use GLSL noise function
  - Most (all?) implementations are *really* bad
  - Some go as far as to return a constant value for all inputs
- Implement noise in C, generate large noise texture
  - Has tiling artifacts
  - Can consume a lot of texture memory
- Implement noise in GLSL code
  - Several implementations exist
    - See *GPUGems 2*
  - Most use several textures for tables
  - Use 60 – 80 GPU instructions

References

Perlin, K. 1999. Making Noise. Presented at GDCHardCore. http://www.noisemachine.com/talk1/

Perlin, K. 2002. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and interactive Techniques* (San Antonio, Texas, July 23 - 26, 2002). SIGGRAPH '02. ACM, New York, NY, 681-682. http://mrl.nyu.edu/~perlin/noise/

Zucker, Matt. 2001. The Perlin noise math FAQ. http://www.cs.cmu.edu/~mzucker/code/perlin-noise-math-faq.html

http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Break

Wang Tiles

- Goal: we want to create an infinite, non-repeating texture for things like grass, sand, etc.

Wang Tiles

- Goal: we want to create an infinite, non-repeating texture for things like grass, sand, etc.
  - Even a 2048x2048 texture will show tiling artifacts
  - *And* it will use 16MB of texture memory...yuck!

Wang Tiles

- Goal: we want to create an infinite, non-repeating texture for things like grass, sand, etc.
  - Even a 2048x2048 texture will show tiling artifacts
  - *And* it will use 16MB of texture memory...yuck!
- Create a “mosaic” from small a few small “tiles”

Wang Tiles

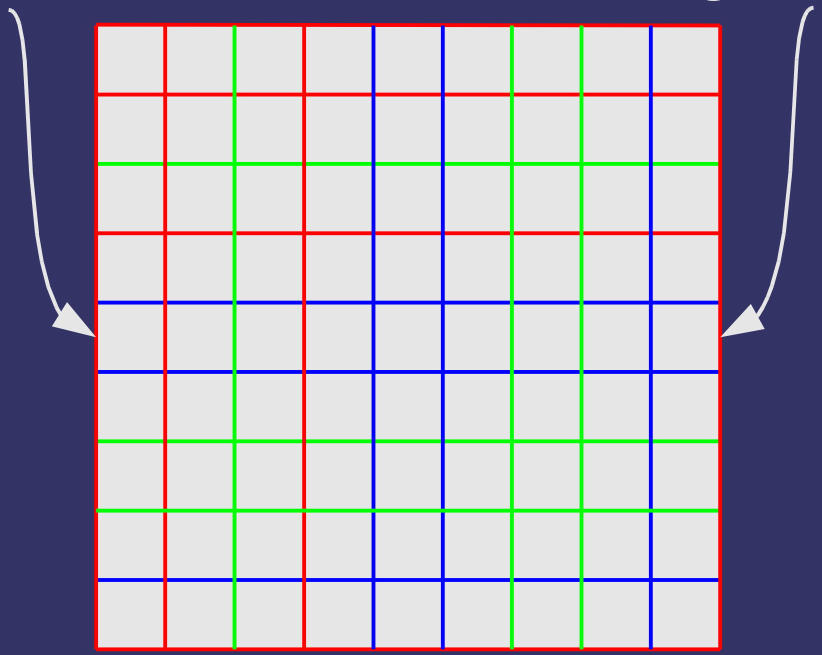
- Goal: we want to create an infinite, non-repeating texture for things like grass, sand, etc.
  - Even a 2048x2048 texture will show tiling artifacts
  - *And* it will use 16MB of texture memory...yuck!
- Create a “mosaic” from small a few small “tiles”
  - If the tile selection is pseudo-random, as few as 32 tiles can have a *very* large repeat period
  - Unlike mosaic tiles, texture tiles have to match at the edges
    - Either all tiles edges have to match or the selection algorithm has to pick a tile that will match edges with its neighbors

Wang Tiles – Edge Coloring

- Name the four tile edges N, E, S, W
  - The N/S edges can have one of K_v edge "colors"
  - The E/W edges can have one of K_h edge "colors"
    - A tile with an N edge of color X must be south of a tile with an S edge of color X
  - A tile with each possible combination of edge colors must exist
    - There must be at least $K_v^2 \times K_h^2$ tiles

Wang Tiles – Tile Arrangement

- Assuming we have a set of tiles...
  - Generating tiles from a sample source image is a larger topic than we have time for
- Arrange tiles in a $K_v \times K_h$ pattern in a 2D texture
  - Neighboring tiles must obey edge coloring rules
  - Even neighbors across border edges!



Wang Tiles – Tile Arrangement

- Given a pair of edge colors, the following place-ment algorithm is use:

$$Index(e_1, e_2) = \begin{cases} 0 & if\ e_1 = e_2 = 0 \\ e_1^2 + 2e_2 - 1 & if\ e_1 > e_2 > 0 \\ e_2^2 + 2e_1 & if\ e_2 > e_1 \geq 0 \\ (e_2 + 1)^2 - 2 & if\ e_1 = e_2 > 0 \\ (e_1 + 1)^2 - 1 & if\ e_1 > e_2 = 0 \end{cases}$$

Wang Tiles – Tile Selection

- Given texture coordinate (s, t):
  - Calculate tile index
    - $O_h = t / T_h$
    - $O_v = s / T_v$
  - Hash tile index to calculate edge colors
    - $C_s = H(H(O_h) + O_v) \% K_v$
    - $C_n = H(H(O_h) + O_v + 1) \% K_v$
    - $C_w = H(O_h + H(O_v * 2)) \% K_h$
    - $C_e = H(O_h + 1 + H(O_v * 2)) \% K_h$
    - Notice that $C_e(x, y) = C_w(x + 1, y)$
  - Convert edge colors to row / column indexes

Wang Tiles – Tile Selection

- Given texture coordinate (s, t):
  - Calculate row / column position in texture
    - $t_{base} = I_h * T_h$
    - $s_{base} = I_v * T_v$
  - Calculate texel offset within tile
    - $t_{offset} = t \% T_h$
    - $s_{offset} = s \% T_v$
  - Sample the texture!
    - Final coordinate is ($s_{base} + s_{offset}$, $t_{base} + t_{offset}$)

Wang Tiles – Hash Function

- Implement as a permutation table
  - Use a texture rectangle that is 1 texel tall
    - Use roughly 4x entries in table as possible edge colors
  - More recent hardware can use uniform arrays
    - Geforce 6 or Radeon X800

Wang Tiles – Filtering Gotchas

- Mipmap filtering can be a problem...
  - The 1x1 level blends all of the tiles together...bad!!!
  - Need to clamp the minimum LOD to the level lowest level that doesn't blur across tile boundaries
  - This is *much* easier with texture arrays

Next week...

- More procedural textures

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.