Procedural Shading

The 2nd half

- Cook's Shade Trees
- Ken Perlin: PSE and Noise
- Shader Languages

Shading

- So far we have considered:
  - BRDFs
  - Shading & Illumination Models
  - Texture Maps
- Today we start to look at shaders that handle shading (and texturing) procedurally
  - Surface characteristics are defined by a function
    - Shading model – simulates behavior of surface material w.r.t. diffuse and specular reflection
    - Pattern generation - texture pattern and sets surface property values

Procedural Shading

Shading (and/or texture) determined by a function

Advantages

- Compact
- Resolution Independent
- Unlimited Extent
- Parameterizable -> class of textures

Disadvantages

- Programming=> debugging
- Unpredictable results
- Time vs. space tradeoff (can take a long time)

Shade Trees [Cook84]

- First procedural shading system
- Allowed use of different shading model for each surface as well as light sources and atmospheric considerations, i.e., light and atmosphere trees
- Traditional shading techniques could be combined
- Handled complexity and simplicity in same image
  - Color and transparency
  - Textures
  - Reflection mapping
  - Displacement mapping
  - Solid texturing

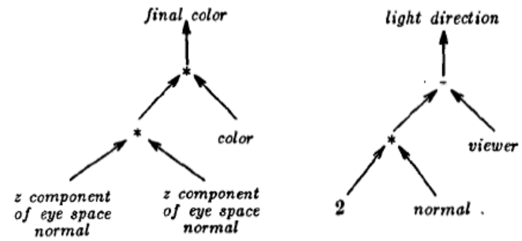
Shade Trees [Cook84]

- Shading calculated by combining basic functional operations using appearance parameters
- Operations are organized in a tree (directed acyclic graph).
  - Nodes – Operations
    - Uses zero or more appearance parameters as input
    - Produces one or more appearance parameters as output
  - Children – operands – basic geometric info: normals, location, etc.
- Result of shade tree evaluation is a color
- Evaluating equivalent to parsing tree (post order - compiler design)

Procedural Shading – Shade Trees

- Basic operations include
  - Vector operations (normalize, dot product / cross product)
  - Arithmetic operations
  - Interpolation / “mix”
  - stochastic functions
  - Variables (points in eye or world)
  - Expandable dynamically
- Basis for Renderman Shading Language

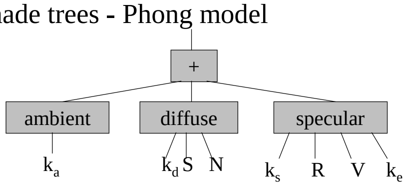
Procedural Shading – Shade Trees



[Cook84]

Procedural Shading – Shade Trees

- Shade trees - Phong model

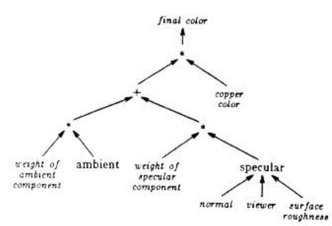


$$L(V) = k_a L_a + k_d \sum_i L_i (\mathrm{S}_i \bullet \mathrm{N}) + k_s \sum_i L_i (\mathrm{R}_i \bullet \mathrm{V})^{k_e}$$

ambient diffuse specular

Procedural Shading – Shade Trees

- Shade Trees - example…copper



[Cook84]

Procedural Shading – Shade Trees

Shade trees - example code: for “metal” shade tree

```
float a=.5, s=.5 ;
float roughness=.1 ;
float intensity ;
color metal_color=(1,1,1) ;
intensity = a*ambient() +
          s*specular(normal,viewer,roughness);
final_color = intensity * metal_color ;
```

Built into language

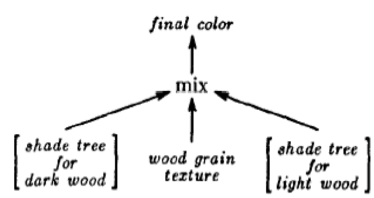
Surface Command – designates shade tree for object (overrides default values)

```
surface "metal",
        "metal_color", material bronze,
        "roughness", .15
```

[Cook84]

Procedural Shading – Shade Trees

- Shade trees – “mix” uses one of inputs to interpolate between the other two



[Cook84]

Procedural Shading – Shade Trees

- Are parameterizable
- Have access to “important” attributes of the point in question
  - Normals, viewer vector, light vectors
- Can be functionally combined
  - Output of one shade tree can be input to another by attaching as a branch
  - Nothing more that a parse tree for a function
  - Functional Programming (LISP)

Procedural Shading – Shade Trees

- Effectively using shade trees is more of an art than a science.

[Cook84]

Perlin’s Pixel Stream Editor (PSE)

- Attempt to create a language around functional shade generation
  - C like language
  - Included control structures
- Originally designed to work on pixels of an image as a postprocessor
  - Input image -> PSE (filter) -> output image
  - Input image has variable list: surface identifiers, point – location, normal, etc.

Procedural Shading – Perlin’s PSE

- Example

Variable related to input image; others point, normal

```
if surface == 1
        color = [1 0 0] * max(0.1, dot(normal,[1 0 0])
else
        color = [0 0 0.1]
```

color

normal

Produces diffusely shaded red object lit from positive x direction on a dark blue background.

Procedural Shading – Perlin’s PSE

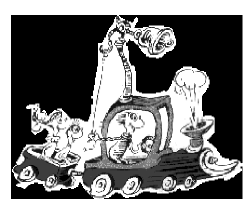
- Any space function can be thought of as representing a solid material
- If evaluated at visible surface points, get sculpture!
  - Shape and texture independent
  - Small code!
- PSE programs are evaluated in 3D space to produce such solid textures
  - Knowledge of x,y,z coordinates
  - Knowledge of important “vectors” at surface

Procedural Shading – Perlin’s PSE

- But the biggest contribution from the PSE was
  - THE NOISE

For, tomorrow, he knew, all the Who girls and boys
Would wake bright and early. They'd rush for their toys!
And then! Oh, the noise! Oh, the noise!
Oh, the Noise! Noise! Noise! Noise!
That's one thing he hated! The NOISE!
NOISE! NOISE! NOISE!

How the Grinch Stole Christmas



Procedural Shading – Perlin Noise

- Observation:
  - Most things in the world have some sort of random or stochastic component to them
  - A procedural shading system requires the use of randomness (“noise”) for realism.
  - Need more than simple random number generator.

Procedural Shading – Perlin Noise

- What is noise
  - Random signal with rich frequency distribution
  - Applet

    http://graphics.lcs.mit.edu/~legakis/MarbleApplet/marbleapplet.html
  - Types of noise:
    - White – uniform frequency
    - Pink – filtered
    - Gaussian – based on Gaussian distribution
  - None appropriate for shader use

Procedural Shading – Perlin Noise

- Perlin on noise:
  - “Noise appears random but it is not. If it were really random, then you’d get a different result each time you call it. Instead it is “pseudo-random” – it gives the *appearance* of randomness”
  - “Noise is a mapping from $R^n \rightarrow R$ – you input an n-dimensional point with real coordinates and it gives you a real value. Currently, the most common uses is for n=1, 2, and 3. The first is used for animation, the second for cheap texture hacks, and the third for less-cheap texture hacks.”

Procedural Shading-Noise Properties

- Repeatable
- Known range [-1, 1]
- Band limited / scalable
- Doesn’t exhibit obvious periodicities
- Statistically invariant under translation
- Statistically invariant under rotation

Procedural Shading-Perlin Noise

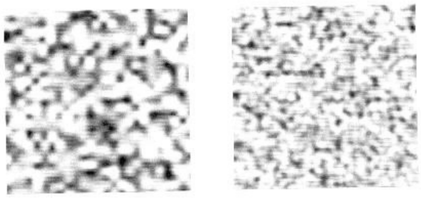
- Controllable random number generator
- Emphasized importance of stochastic functions in texture design
- Very efficient in time and space
- Implemented as a basic operation in the MMX chipset and other graphics hardware
- Won Ken an Academy Award

Procedural Shading-Perlin Noise

- “Controlled” Noise function
  - White noise = noise at all frequencies
  - Control the frequency of the noise used
  - e.g. `noise (2x)` will contain twice as much frequency (detail) `as noise(x)`

Procedural Shading-Perlin Noise

- Noise frequency and detail



Procedural Shading-Perlin Noise

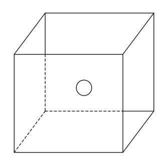
- Perlin Noise
  - Returns a scalar value between -1 and 1
  - takes a 3d vector as an argument
  - `float noise3 (float [3] vec)`

Procedural Shading-Perlin Noise

- 3D lattice (3D array) with 4 pseudorandom real numbers per point in the array
- for each point (x_0,y_0,z_0) we assign a set of 4 pseudorandom numbers (a, b, c, d).
- Compute $d' = d - (ax_0+by_0+cz_0)$
- noise (x,y,z)
  - if (x, y, z) is on the lattice, noise (x,y,z) = d
  - if (x,y,z) is NOT on the lattice, the values of (a,b,c,d) are interpolated from the (a,b,c,d) values of neighboring lattice points. Then noise(x,y,z) = ax + bx +cz +d' using the interpolated (a,b,c,d)

Procedural Shading-Perlin Noise

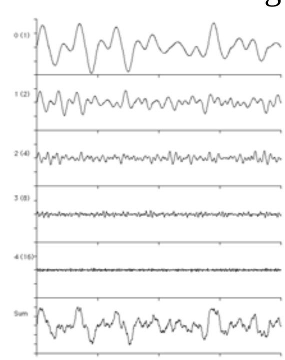
- Perlin noise - Lattice



Procedural Shading-Perlin Noise

- Perlin has further optimized using look up tables
- Complete "C" code (approx 150 lines) on Web at:
  - http://mrl.nyu.edu/~perlin/doc/oscar.html#noise
- Perlin has since revised the basic noise algorithm in order for efficiency, functionality, and ease of hardware implementation.
- Perlin has since applied same paradigm to:
  - Solid Modeling
  - Animation / Gesturing

Procedural Shading – Perlin Noise

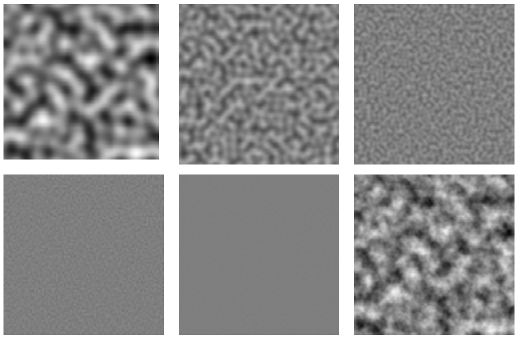


Increasing harmonics of 1-D Perlin noise

Sum of 1st 8 harmonics

Paul Burke, 2000

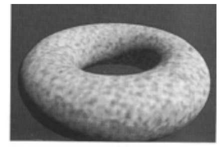
Procedural Shading – Perlin Noise



Paul Burke, 2000 Sum

Procedural Shading-Perlin Noise

- Example 1 – *Spotted Donut* -No detail outside certain size range



[Perlin85]

```
Color = white * noise (point)
```

Vector

Procedural Shading-Perlin Noise

- Example 2 – *Bozo's Donut*

[Perlin85]

```
Color = Colorful(noise (k*point))
```

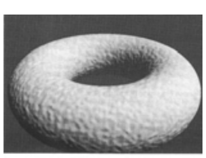
Constant multiplier

Procedural Shading-Perlin Noise

- Dnoise – Vector valued differential of noise signal, i.e., gradiant/derivative of noise function
- Dnoise (x,y,z) = (dNoise/dx, dNoise/dy, dNoise/dz)
- Good for modifying normal vector (bump mapping)

Procedural Shading-Perlin Noise

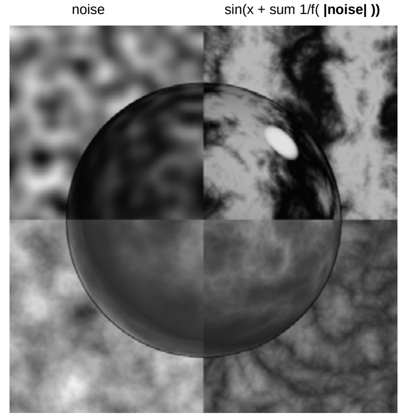
- Dnoise example – *Bumpy Donut*



[Perlin85]

```
Normal += Dnoise (point)
```

noise sin(x + sum 1/f(**|noise|**))



sum 1/f(noise) sum 1/f(**|noise|**)

Perlin web

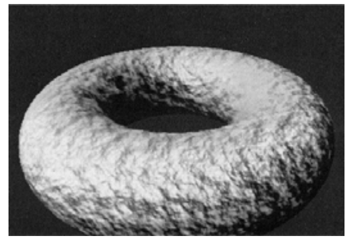
Creating Wrinkles

- Adding successive noise at different but regular frequencies
- 1/f, self-similar quality (Fractal-like…more on fractals later)

$$\mathrm{NOISE}(x) = \sum_{i=0}^{i=N-1} \frac{\mathrm{Noise}(\mathrm{b}^i\, x)}{\mathrm{a}^i}$$

Creating Wrinkles

- Perlin example: *Wrinkled Donut*



[Perlin85]

Procedural Shading - Perlin

- Turbulence used to model – stocastic components
  - Water
  - Clouds
  - Bubbles
  - Falling leaves
  - Swaying trees
  - Flocks of birds
  - Rippling muscles

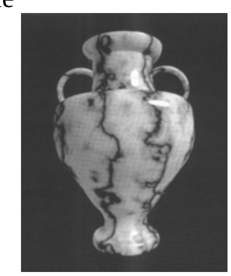
[Perlin85]

Procedural Shading - Perlin

- Perlin - turbulence example

```
Function marble(point)
     x  = point[1] +
          turbulence (point)
     return marble_color(sin x)
```

Perturbs the layer



[Perlin85]

Procedural Shading-Perlin Noise

- Perlin Noise Demo Applet http://mrl.nyu.edu/~perlin/noise/
- Perlin Noise Applied to Animation http://mrl.nyu.edu/~perlin/facedemo/

Procedural Shading - Perlin

- Summary
  - Compact, functional shading specifications
  - Efficient "controllable" noise function
  - Noise adds to complexity and realism
  - Building good procedural textures is more of an art than a science.

Procedural Shading - Noise

- For a discussion on other noise functions see:
  - Ebert, et al, *Texture and Modeling: A Procedural Approach*, Chapter 2
- A nice discussion on Perlin Turbulence:
  - http://astronomy.swin.edu.au/~pbourke/texture/perlin/

Shading Languages

- Renderman Shading Language
  - Grew out of shade trees
  - Goals
    - Abstract shading language based on ray optics. Independent of any specific algorithm or implementation
    - Interface between rendering program and shading model
    - High level language that is easy to use.

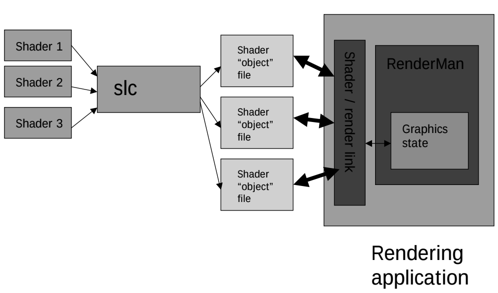
Shading Language

- RenderMan shaders
  - Renderman provides a complete programmable model of light transport.
    - More next time
  - Surface reflectance shaders
    - Compute the light reflected from a surface in a given direction, i.e., programmable BRDFs .

Runtime architecture

- Renderman consists of three parts:
  - Functional scene description mechanism (API for C)
  - ***RenderMan is an Interface Specification!***
    - State Model Description – Maintains a current graphics state that can be placed onto a stack.
    - Geometry is drawn by utilizing the current graphics state.
  - File format - RenderMan Interface Bytestream (RIB)
  - Shading Language and Compiler.

Runtime architecture



Renderman Shading Language

- Creating effective shaders with the Renderman Shading Language is more of an art than a science.

Shading Languages

- Many commercial renderers (e.g. Ray Dream 3D / Lightwave) now come with a shading / plugin API.
- Allows shaders to be written using a native programming language (like C or C++).
- Using these APIs effectively is more of an art than a science.

Real Time Shaders

- Programmable shading capability is now built in to current graphics hardware (GPU)
  - Same flavor as Renderman shaders.
    - However, model is far less comprehensive.
  - Examples
    - Cg – nVidia
    - RenderMonkey – ATI
    - OpenGL Shader language – hardware independent
  - Using real time shaders effectively is more of an art than a science.

Next time

- In depth look at the RenderMan shader language.