

CS 432/637 Interactive Computer Graphics

Prof. David Breen
Computer Science Department
Drexel University

[More Texture Mapping](#)

Slide Credits

- Jonathan Cohen - Johns Hopkins University
- John Hart - University of Illinois
- Leonard McMillan - UNC, Chapel Hill
- Dani Lischinski - Hebrew University
- Craig Schroeder - Drexel University

Procedural Textures and Shading

Procedural Texture/Shading

- Define texture color/value with a procedure
- Allows for a wide variety of (programmable) surface materials and embellishments
- Many independent variables
 - Position, normal, curvature, geodesic distance, time



Potential Advantages of Procedural Textures

- Compact representation
- No fixed resolution
- No fixed area
- Parameterized - generates class of related textures

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



Disadvantages of Procedural Textures

- Difficult to build and debug
- Surprising results
- Slow evaluation

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



Procedural Texture Conventions

Avoid conditionals

- Convert to mathematical functions when possible
- Makes anti-aliasing easier

Parameterize rather than building in constants

- Assign reasonable defaults which may be overridden

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



Simple Building Blocks

Mix (lerp)

Step, smoothstep, pulse

Min, max, clamp, abs

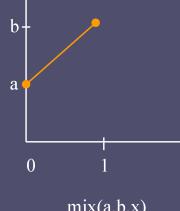
Sin, cos

Mod, floor, ceil

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



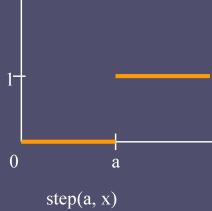
Mix



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



Step

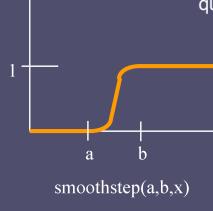


Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



Smoothstep

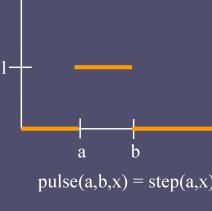
$\text{Sin}()$ or piece-wise quadratics work well



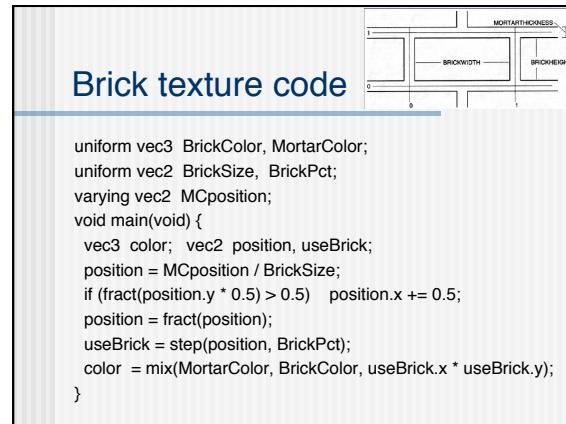
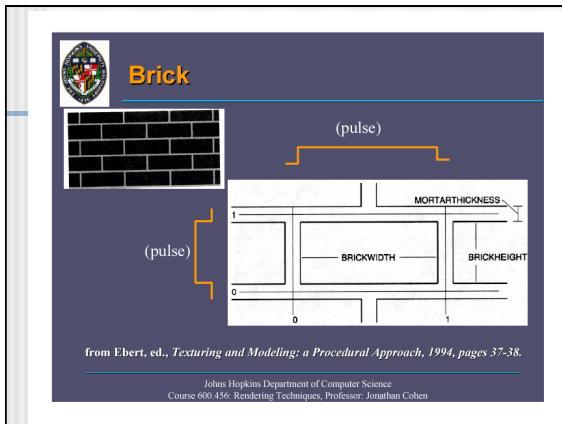
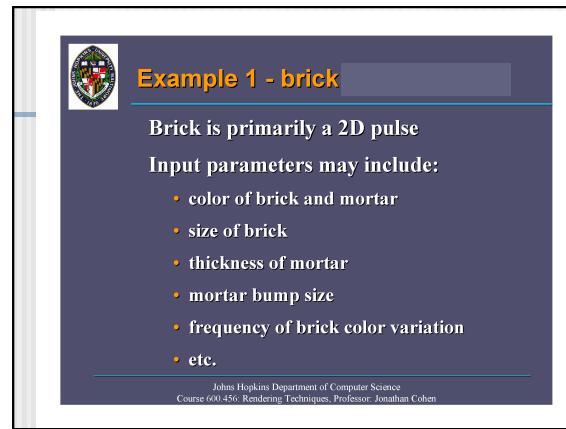
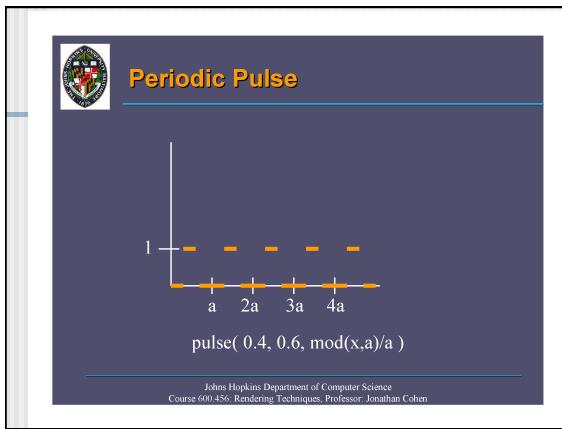
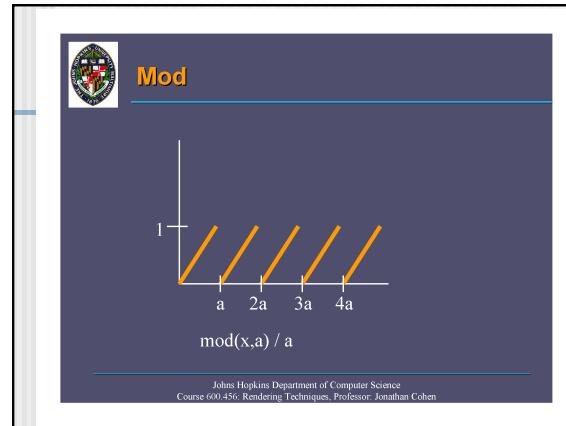
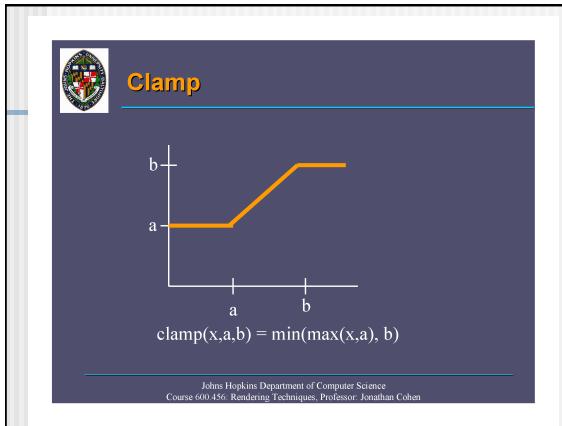
Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



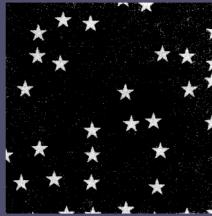
Pulse



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



 **Example - Star Wallpaper**



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

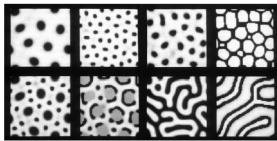
 **Example - Star Wallpaper**

- Divide 2D texture space into uniform grid
- Decide whether or not to place a star in each cell
- Perturb position of star within each cell
- To render a point on surface, check nearby cells for stars which may cover point

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

Reaction-Diffusion Textures
Turk 91; Witkin & Kass 91

- ◆ Reaction-diffusion is a mathematical model for generation of natural patterns, arising due to local non-linear interactions of excitation and inhibition.
- ◆ First proposed by Alan Turing in 1952



1

Main Idea

- ◆ Certain cell properties (such as generation of pigments) are determined in the embryo based on the local concentration of one or more chemicals, called morphogens.
- ◆ Morphogen concentrations are determined by two concurrent processes:
 - ◆ Diffusion - spreading of morphogens through the tissue
 - ◆ Reaction - chemical reactions that create or destroy morphogens, based on their concentration in each cell
- ◆ Natural patterns can be generated by simulating the R-D processes.

$$U + 2V \rightarrow 3V \quad 2$$

$$V \rightarrow P$$

2

Mathematical Model

- ◆ A system of non-linear partial differential equations.
- ◆ Let $C(x,y)$ denote the concentration of a morphogen in a (2D) system. The governing equation is:
$$\dot{C} = \frac{\partial C}{\partial t} = a^2 \nabla^2 C - bC + R$$
- ◆ where a is the diffusion coefficient, b is the dissipation coefficient, R is the rate of change due to reaction, and $\nabla^2 C = \frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}$ is the Laplacian of C .

3

Finite Differences Apprx

- ◆ On a 2D integer grid with spacing h , the Laplacian is approximated as:
$$\nabla^2 C \approx \frac{C_{i+1,j} + C_{i-1,j} + C_{i,j+1} + C_{i,j-1} - 4C_{i,j}}{h^2}$$
- ◆ In other words, the Laplacian operator is a convolution with:

$$L = \frac{1}{h^2} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

4

Finite Differences (cont'd)

The original equation $\dot{C} = \frac{\partial C}{\partial t} = a^2 \nabla^2 C - bC + R$

can be rewritten as $\dot{C} = M * C + R$

where $M = \frac{1}{h^2} \begin{bmatrix} 0 & a^2 & 0 \\ a^2 & -4a^2 - h^2 b & a^2 \\ 0 & a^2 & 0 \end{bmatrix}$

5

Euler's Method

- ◆ We are really after C , rather than its derivative. Therefore, we must integrate the equation:

$$C_t = C_0 + \int_0^t \dot{C} dt$$

- ◆ The integration is performed using small discrete time steps:

$$C_{t+\Delta t} = C_t + \Delta t(M * C_t + R_t)$$

6

Anisotropic Diffusion

- ◆ More interesting patterns can be created if the diffusion rates are different in different directions. Instead of modeling diffusion with the Laplacian operator

$$a^2 \nabla^2 C = a^2 \left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} \right)$$

- ◆ use different rates along X and Y directions:

$$a_1^2 \frac{\partial^2 C}{\partial x^2} + a_2^2 \frac{\partial^2 C}{\partial y^2}$$

7

Space-Varying Diffusion

- ◆ Control diffusion pattern spatially by specifying a "diffusion map" - a map that gives the diffusion coefficients at each position on the surface.

10



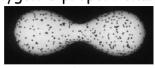
Reaction-Diffusion on Surfaces

- ◆ To avoid the difficulties associated with mapping 2D textures onto objects in 3D, we'd like to perform the RD simulation directly on the surface of an object.
 - ◆ Randomly distribute points over the object's surface
 - ◆ Create a mesh of even-sized cells
 - ◆ Simulate R-D on this mesh

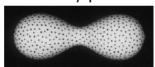
12

Relaxation of Random Points

- ◆ Given a polyhedral model, randomly place points (for uniform distribution, the probability of a point being placed in a polygon is proportional to its area).



- ◆ Relaxation: move each point according to repulsion forces applied to it by other nearby points (iterate several times).



13

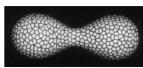
Relaxation

- ◆ Loop k times:
 - ◆ For each point P on surface
 - ◆ Determine nearby points to P
 - ◆ Map these points onto plane of P's polygon
 - ◆ Compute and store repulsive forces on P
 - ◆ For each point P on surface
 - ◆ Compute the new position of P based on repulsive forces

14

Mesh Construction

- ◆ Compute Voronoi regions, based on the points computed in the previous stage:



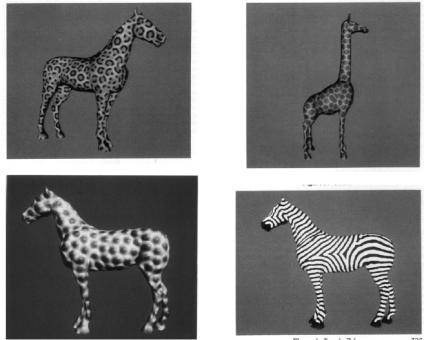
- ◆ The Voronoi region of a point P contains all points that are closest to P
- ◆ For each point we compute it's Voronoi neighbors and the lengths of the edges between adjacent Voronoi regions.

15

Reaction-Diffusion on a Mesh

- ◆ The edge lengths become the diffusion coefficients (normalized to sum to 1)
 - ◆ Results in isotropic diffusion.
 - ◆ How can anisotropic diffusion be accommodated?
- ◆ The Laplacian of a cell is computed taking into account all of the cells neighbors. Each neighbor's contribution is weighted by the diffusion coefficient corresponding to the edge between the two cells.

16

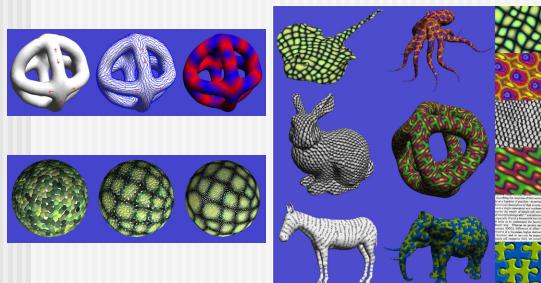


Texture Synthesis

"Texture Synthesis on Surfaces", Greg Turk,
Proc. SIGGRAPH 2001, pp. 347-354

- Spread small texture patch over an arbitrary multiresolution mesh
- Color vertices from coarse to fine, while following a user-defined vector field
- Determine color by examining the color of neighboring points and finding best match to a similar pixel neighborhood in texture sample

Texture Synthesis



Procedural Noise

Noise Functions

- Break up regularity
- Add realism
- Enable modeling of irregular phenomena and structures

White Noise

Sequence of random numbers

Uniformly distributed

Totally uncorrelated

- no correlation between successive values

Not desirable for texture generation

- Too sensitive to sampling problems
- Arbitrarily high frequency content

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor Jonathan Cohen



Ideal Noise for Texture Generation

Repeatable pseudorandom function of inputs

Known range [-1, 1]

Band-limited (maximum freq. about 1)

No obvious periodicities

Stationary and isotropic

- statistical properties invariant under translation and rotation

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor Jonathan Cohen



Lattice Noise

Low pass filtered version of white noise

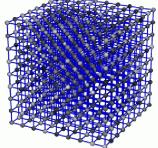
- Random values associated with integer positions in noise space
- Intermediate values generated by some form of interpolation
- Frequency content limited by spacing of lattice

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor Jonathan Cohen

Noise and Turbulence

When we say we want to create an "interesting" texture, we usually don't care exactly what it looks like -- we're only concerned with the overall appearance. We want to add random variations to our texture, but in a controlled way. Noise and turbulence are very useful tools for doing just that.

A *noise function* is a continuous function that varies throughout space at a uniform frequency. To create a simple noise function, consider a 3D lattice, with a random value assigned to each triple of integer coordinates:



4/3/2003

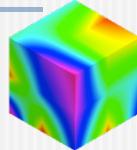
Lecture 19

11

Making Noise

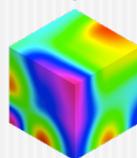
Good:

- Create 3-D array of random values
- Trilinearly interpolate



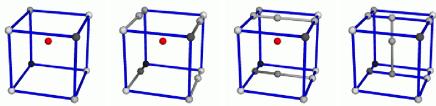
Better

- Create 3-D array of random 3-vectors
- Hermite interpolate



Interpolating Noise

To calculate the noise value of any point in space, we first determine which cube of the lattice the point is in. Next, we interpolate the desired value using the 8 corners of the cube:



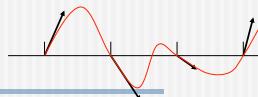
Trilinear interpolation is illustrated above. Higher-order interpolation can also be used.

4/3/2003

Lecture 19

12

Hermite Interpolation



Some cubic $h(t) = at^3 + bt^2 + ct + d$ s.t.

- $h(0) = 0 (d = 0)$
- $h(1) = 0 (a + b + c = 0)$
- $h'(0) = r_0 \quad (c = r_0)$
- $h'(1) = r_1 \quad (3a + 2b + r_0 = r_1)$

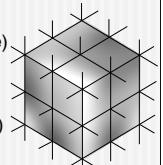
Answer:

$$h(t) = (r_0 + r_1)t^3 - (2r_0 + r_1)t^2 + r_0t$$

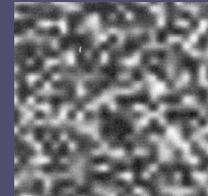
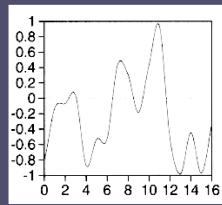
Noise Functions



- Add "noise" to make textures interesting
- Perlin noise function $N(x,y,z)$
 - Smooth
 - Correlated
 - Bandlimited
- $N(x,y,z)$ returns a single random number in $[-1, 1]$
- Gradient noise (like a random sine wave)
 - $N(x,y,z)=0$ for int x,y,z
 - Values not at lattice points are interpolated, using gradients as spline coefficients
- Value noise (another random sine wave)
 - $N(x,y,z)=\text{random}$ for int x,y,z

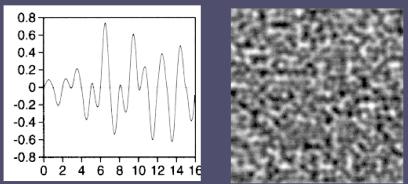


1D and 2D Value Noise



Johns Hopkins Department of Computer Science
Course 600.450: Rendering Techniques, Professor Jonathan Cohen

 **1D and 2D Gradient Noise**



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

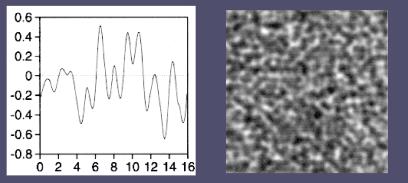
 **Value vs. Gradient Noise**

- Both noises have limited frequencies
- Value noise slightly simpler to compute
- Gradient noise has most of the energy in the higher frequencies
 - forced zero crossings
- Gradient noise has regularity because of zero crossings

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

 **Value Gradient Noise**

Weighted sum of value and gradient noises



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

 **Example - Perturbed Texture**

- Use noise function to apply perturbation to texture coordinates
- Look up image texture (or generate procedural texture) using modified coordinates

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

 **Example - Perturbed Texture**



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

Evaluating Noise

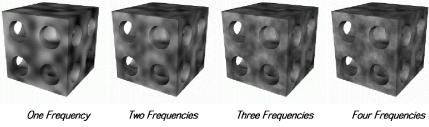
Since noise is a 3D function, we can evaluate it at any point we want. We don't have to worry about mapping the noise to the object, we just use (x, y, z) at each point as our 3D texture coordinates! It is as if we are carving our object out of a big block of noise.



4/3/2003 Lecture 19 13

Turbulence

Noise is a good start, but it looks pretty ugly all by itself. We can use noise to make a more interesting function called turbulence. A simple turbulence function can be computed by summing many different frequencies of noise functions:



One Frequency Two Frequencies Three Frequencies Four Frequencies

Now we're getting somewhere. But even turbulence is rarely used all by itself. We can use turbulence to build even more fancy 3D textures...

4/3/2003 Lecture 19 14



Turbulence

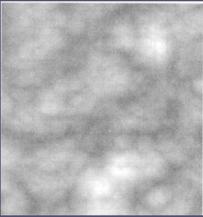
```
float turbulence(point Q)
{
    float value = 0;
    for (f= MINFREQ; f < MAXFREQ; f *= 2)
        value += abs(noise(Q*f))/f;
    return value;
}
```

(in practice, don't use a round number like 2)

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor Jonathan Cohen



Turbulence



Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor Jonathan Cohen

3D Procedural Texture Maps

Texture Mapping

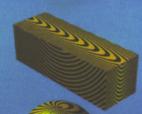
- Maps image onto surface
- Depends on a surface parameterization (s, t)
 - Difficult for surfaces with many features
 - May include distortion
 - Not necessarily 1:1



Kettle, by Mike Miller

Solid Texturing

- Uses 3-D texture coordinates (s, t, r)
- Can let $s = x$, $t = y$ and $r = z$
- No need to parameterize surface
- No worries about distortion
- Objects appear sculpted out of solid substance



Surface Texture
features
don't
line up



Solid Texture
features
do
line up

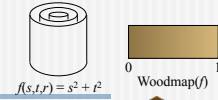
Darwyn Peachey, 1985

Solid Texture Problems

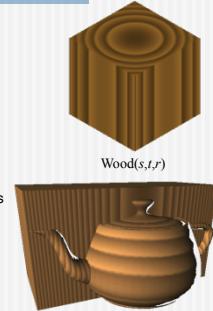
- How can we deform an object without making it swim through texture?
- How can we efficiently store a procedural texture?



Procedural Texturing

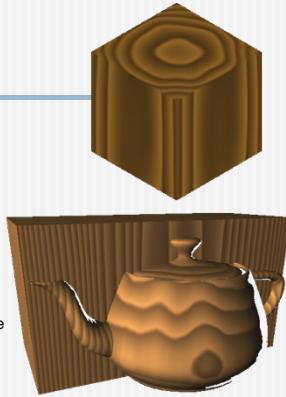


- Texture map is a function
- Write a procedure to perform the function
 - input: texture coordinates - s, t, r
 - output: color, opacity, shading
- Example: Wood
 - Classification of texture space into cylindrical shells
 $f(s,t,r) = s^2 + t^2$
 - Outer rings closer together, which simulates the growth rate of real trees
 - Wood colored color table
 - Woodmap(0) = brown "earlywood"
 - Woodmap(1) = tan "latewood"
- $\text{Wood}(s,t,r) = \text{Woodmap}(f(s,t,r) \bmod 1)$



Using Noise

- Add noise to cylinders to warp wood
 - $\text{Wood}(s^2 + t^2 + N(s,t,r))$
- Controls
 - Amplitude: power of noise effect
 $a N(s, t, r)$
 - Frequency: coarse vs. fine detail
 $N(f_s s, f_t t, f_r r)$
 - Phase: location of noise peaks
 $N(s + \phi_s, t + \phi_t, r + \phi_r)$

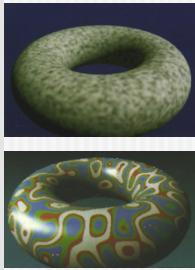


An Image Synthesizer

K. Perlin, Proc. SIGGRAPH '85, July 1985, p. 287

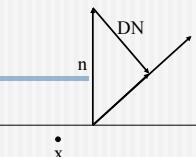
- Created an interpreted, high level language for describing 2D/3D textures
 - Easy to program
 - No compilation necessary
 - Define a set of intrinsic procedural primitives
 - More efficient, reusable, flexible, rich set of texturing tools
 - High level operations (arithmetic, comparisons)
 - Include looping and branching
 - High level, intrinsic types like vectors
- New primitives
 - Noise, Dnoise, Turbulence, Composition

Colormap Donuts



- Spotted donut
 - $\text{Gray}(N(40*x, 40*y, 40*z))$
 - $\text{Gray}()$ - ramp colormap
 - Single 40Hz frequency
- Bozo donut
 - $\text{Bozo}(N(4*x, 4*y, 4*z))$
 - $\text{Bozo}()$ - banded colormap
 - Cubic interpolation means contours are smooth

Bump Mapped Donuts



- ```
n += DNoise(x,y,z); normalize(n);
```
- $\text{DNoise}(s,t,r) = \nabla \text{Noise}(s,t,r)$
  - Bumpy donut
    - Same procedural texture as spotted donut
    - Noise replaced with Dnoise and bump mapped

## Composite Donuts



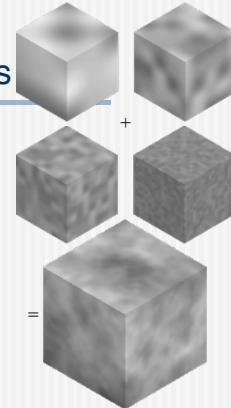
- Stucco donut
  - $\text{Noise}(x,y,z) * \text{DNoise}(x,y,z)$
  - Noisy direction
  - noisy amplitude



- Fleshy donut
  - Same texture
  - Different colormap

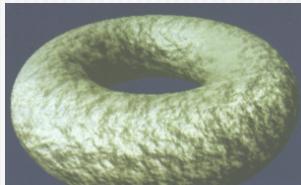
## Fractional Brownian Textures

- $1/f^\beta$  distribution
- Roughness parameter  $\beta$ 
  - Ranges from 1 to 3
  - $\beta = 3$  - smooth, not flat, still random
  - $\beta = 1$  - rough, not space filling, but thick
- Construct using spectral synthesis
 
$$f(\mathbf{s}) = \sum_{i=1}^4 2^{-i\beta} n(2^i \mathbf{s})$$
  - Add several octaves of noise function
  - Scale amplitude appropriately



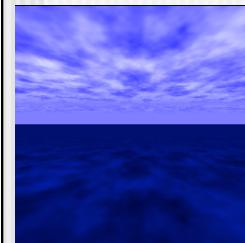
## Fractal Bump-Mapped Donut

```
fbm(beta) {
 val = 0; vec = (0,0,0);
 for (i = 0; i < octaves; i++) {
 val += Noise(2^i*x, 2^i*y, 2^i*z)/pow(2,i*beta);
 vec += DNoise(2^i*x, 2^i*y, 2^i*z)/pow(2,i*beta);
 }
 return vec or val;
}
```



## Clouds Water

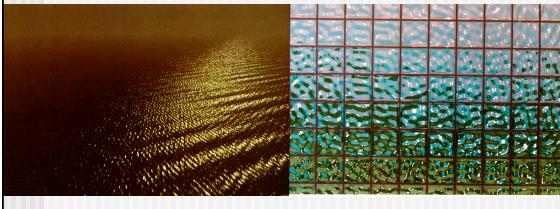
$$f(\mathbf{s}) = \sum_{i=1}^4 2^{-i\beta} n(2^i \mathbf{s})$$



Gunther Berkus via Mojoworld

## Water

- Created by making a wave with Dnoise
- Create multiple waves to simulate water



## Clouds & Bubbles

Created using turbulence, reflection, and refraction

## Marble Example

We can use turbulence to generate beautiful 3D marble textures, such as the marble vase created by Ken Perlin. The idea is simple. We fill space with black and white stripes, using a sine wave function. Then we use turbulence at each point to distort those planes. By varying the frequency of the sin function, you get a few thick veins, or many thin veins. Varying the amplitude of the turbulence function controls how distorted the veins will be.

$$\text{Marble} = \sin(f * (x + A * \text{Turb}(x,y,z)))$$

4/3/2003      Lecture 19      15

## Marble

$$f(s,t,r) = r + \sum_{i=1}^4 2^{-i} n(2^i s, 2^i t, 2^i r)$$

Ken Perlin, 1985

## Example - Blue Marble

Marble vase (right) from Foley, van Dam, Feiner, and Hughes. *Computer Graphics: Principles and Practice*.

Johns Hopkins Department of Computer Science  
Course 600.456, Rendering Techniques, Professor Jonathan Cohen

## Fire

See "Texture and Modeling: A Procedural Approach," Ebert et al., Morgan Kaufmann , 2003

Ken Musgrave

## Fire (Corona)

- Apply turbulence when flow exists
- Fire in a solar corona is flowing away from the center

```
Function corona(v)
 radius = norm(v)
 dr = turbulence(v)
 return color_of_corona(radius + dr)
```

