Questions from Last Time?

- 2D texture maps
- Image maps
- Two-step mapping
- Texture anti-aliasing
- Environment/reflection maps
- Bump maps
- Displacement maps

More 2D Texture Mapping

Minimization of Distortion
(Malliott, Yahia, Verroust 93)

- Given a parametric mapping \( \phi: U \rightarrow E^3 \),
- define the elastic deformation energy by the Green-Lagrange deformation tensor:
  \[ E(\phi) = \int_U \left( \frac{\partial \phi}{\partial u} \cdot \frac{\partial \phi}{\partial u} \right) \cdot \left( \frac{\partial \phi}{\partial v} \cdot \frac{\partial \phi}{\partial v} \right) \, du \, dv \]

- We would like to find a mapping that minimizes the energy \( E \).

Triangulated Surfaces

- The mapping from/to texture space is affine over each triangle: \( M^T M_z \rightarrow m_1 m_2 m_3 \).
- Substituting into the Green-Lagrange formula gives a high-degree rational polynomial: solvable, but complicated.
Distance based energy

- A simpler discrete form of energy:
  \[ E_i = \sum_{(l_j,l_k) \text{ in edge}} \left( \frac{(m_{l_j} - m_{l_k})^2 + (M_{l_j} - M_{l_k})^2}{|M_{l_j} - M_{l_k}|} \right) \]
- This is similar to the energy of a spring net.
- A normalizing term is used to accommodate triangles of different sizes.

Surface based energy

- An energy term defined by measuring the difference in signed areas of the triangles:
  \[ E_i = \frac{1}{2} \sum_{(l_j,l_k) \text{ in edge}} \left( |A_{l_j,l_k} - A_{M_{l_j},M_{l_k}}| - |A_{M_{l_j},M_{l_k}} - A_{M_{l_j},M_{l_k}}| \right) \]
- The total distortion energy is a linear combination:
  \[ E = \alpha E_i + (1 - \alpha) E_j \]

Texture Synthesis

- 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
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

Disadvantages of Procedural Textures
- Difficult to build and debug
- Surprising results
- Slow evaluation

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

Simple Building Blocks
- Mix (lerp)
- Step, smoothstep, pulse
- Min, max, clamp, abs
- Sin, cos
- Mod, floor, ceil
Mix

\[ \text{mix}(a, b, x) \]

Step

\[ \text{step}(a, x) \]

Smoothstep

\[ \text{smoothstep}(a, b, x) \]

Sin() or piece-wise quadratics work well

Pulse

\[ \text{pulse}(a, b, x) = \text{step}(a, x) - \text{step}(b, x) \]

Clamp

\[ \text{clamp}(x, a, b) = \min(\max(x, a), b) \]

Mod

\[ \text{mod}(x, a) / a \]
Brick texture code

```cpp
uniform vec3 BrickColor, MortarColor;
uniform vec2 BrickSize, BrickPct;
varying vec2 MPosition;
void main(void) {
  vec3 color;
  vec2 position, useBrick;
  position = MPosition / BrickSize;
  if (fract(position.y * 0.5) > 0.5)
    position.x += 0.5;
  position = fract(position);
  useBrick = step(position, BrickPct);
  color = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
}
```

Example 1 - brick

Brick is primarily a 2D pulse
Input parameters may include:
- color of brick and mortar
- size of brick
- thickness of mortar
- mortar bump size
- frequency of brick color variation
- etc.

Example 2 - star

Exploit symmetry of star geometry
Input parameters may include:
- Inner and outer star radii
- Number of points
- Star and background colors
- Star bump parameters
- Parameters for star distribution
Star texture code

```cpp
vec3 p0, p1, d0, d1;
starangle = 2*Pi/npoints;
p0 = (max, 0, 0);
p1 = min( vec3(cos(starangle/2), sin(starangle/2), 0) );
d0 = p1 - p0;
ss = s - sctr; tt = t - tctr;
angle = atan2(ss, tt) + Pi;
r = sqrt(ss*ss + tt*tt);
a = modf(angle, starangle)/starangle;
if (a >= 0.5) a = 1 - a;
d1 = r*(cos(a), sin(a), 0) - p0;
in_out = step(0, zcomp(d0 X d1));
color = mix(BackColor, StarColor, in_out);
```

---

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

---

Bump Mapping - Computing N'

F(u,v) = bump height function
P(u,v) = surface position
U = ∂P/∂u (N X ∂P/∂u)
V = ∂P/∂v (N X ∂P/∂v)
D = U + V
N' = (N + D) / |N + D|

---

Bump-Mapped Brick

---

Bevelling Effects

Nice ridges along edges of geometric figures
Parameters:
- Total ridge and plateau widths
- Slope at top and bottom of ridge
Use perpendicular direction to closest edge as D (to add to normal), and scale according to ridge function
**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.

\[ U + 2V \rightarrow 3V \]

\[ V \rightarrow P \]

**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:
  \[ \frac{\partial C}{\partial t} = a^2 \nabla^2 C - bC + R \]
- Where \( a \) is the diffusion coefficient, \( b \) is the dissipation coefficient, \( R \) is the rate of change due to reaction, and \( \nabla^2 \) is the Laplacian of \( C \).

**Finite Differences Approx**

- On a 2D integer grid with spacing \( h \), the Laplacian is approximated as:
  \[ \frac{C_{i,j}^{t+1} - C_{i,j}^t}{h^2} = \frac{C_{i+1,j}^t + C_{i-1,j}^t + C_{i,j+1}^t + C_{i,j-1}^t - 4C_{i,j}^t}{h^2} \]
- In other words, the Laplacian operator is a convolution with:
  \[ L = \frac{1}{4} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \]

**Finite Differences (cont’d)**

The original equation

\[ \frac{\partial C}{\partial t} = a^2 \nabla^2 C - bC + R \]

can be rewritten as

\[ \dot{C} = M \ast C + R \]

where

\[ M = \frac{1}{a^2} \begin{bmatrix} 0 & 0 & 0 \\ 0 & a^2 & 0 \\ 0 & 0 & a^2 \end{bmatrix} \]
**Euler's Method**
- We are really after \( C \), rather than its derivative. Therefore, we must integrate the equation:
  \[
  C_i = C_i + \int_0^t C \, dt
  \]
- The integration is performed using small discrete time steps:
  \[
  C_{i+1} = C_i + \Delta t (M * C_i + R_i)
  \]

**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
  \[
  \Delta^2 = \frac{\partial^2 C}{\partial x^2} \quad \frac{\partial^2 C}{\partial y^2}
  \]
- Use different rates along \( x \) and \( y \) directions:
  \[
  \frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}
  \]

**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.

**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 A-B on this mesh

**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).
**Procedural Noise**

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

---

**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 apply repulsive forces on \( P \)
  - For each point \( P \) on surface
    - Compute the new position of \( P \) based on repulsive forces

---

**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 its Voronoi neighbors and the lengths of the edges between adjacent Voronoi regions.

---

**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 cell’s neighbors. Each neighbor’s contribution is weighted by the diffusion coefficient corresponding to the edge between the two cells.

---

**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

---

**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

---

**Making Noise**

- **Good:**
  - Create 3-D array of random values
  - Trilinearly interpolate
- **Better:**
  - Create 3-D array of random 3-vectors
  - Hermite interpolate

---

**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

---

**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 radial 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:

---

**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.
Some cubic $h(t) = at^3 + bt^2 + ct + d$ s.t.
- $h(0) = 0$ ($d = 0$)
- $h(1) = 0$ ($a + b + c = 0$)
- $h(0) = r_0$ ($c = r_0$)
- $h(1) = r_1$ ($3a + 2b + r_0 = r_1$)

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

Hermite Interpolation

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)
  - $N(x,y,z)$=random for int $x,y,z$

1D and 2D Value Noise

1D and 2D Gradient Noise

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

Value Gradient Noise

Weighted sum of value and gradient noises
Example - Perturbed Texture

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

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.

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)
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

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

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”
  - Wood\((s, t, r)\) = Woodmap\((f(s, t, r))\) mod 1

Using Noise
- Add noise to cylinders to warp wood
- Wood\((s^2 + t^2 + M(s, t, r))\)
- Controls
  - Amplitude: power of noise effect
    \(a M(s, t, r)\)
  - Frequency: coarse vs. fine detail
    \(M(f(s, t, r))\)
  - Phase: location of noise peaks
    \(M(s + \phi_s, t + \phi_t, r + \phi_r)\)
An Image Synthesizer

- 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
  - Gray(N(40*x,40*y,40*z))
  - Gray() - ramp colormap
  - Single 0Hz frequency
- Bozo donut
  - Bozo(N(4*x,4*y,4*z))
  - Bozo() - banded colormap
  - Cubic interpolation means contours are smooth

Bump Mapped Donuts

- Bumpmapped donut
  - Same procedural texture as spotted donut
  - Noise replaced with Dnoise and bump mapped

Composite Donuts

- Stucco donut
  - Noise(x,y,z)*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

$$f(x) = \sum_{n} 2^{-\beta n(2^n)}$$

- Add several octaves of noise function
- Scale amplitude appropriately

Fractal Bump-Mapped Donut

```c
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

\[ f(s) = \sum_{n} 2^n n(2^n s) \]

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 these planes. By varying the frequency of the sine function, you get a few thin veins, or many thin veins. Varying the amplitude of the turbulence function controls how distorted the veins will be.

\[ f(s,t,r) = r + \sum_{i=1}^{14}(2^{i}s,2^{i}t,2^{i}r) \]

Marble

\[ f(s,t,r) = r + \sum_{i=1}^{14} n(2^{i}s,2^{i}t,2^{i}r) \]

Marble

4/23/2003  Lecture 10  15
Fire


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)

Planets

Wrap Up
- Discuss next programming assignment
  - 2D image texture mapping to spheres
  - Add 3D procedural texture mapping to triangle meshes
  - Or
  - Refraction

Refraction Issues
- Flip normal for interior ray offsets and reflections/refractions
- Differing offsets
- Coping with total internal reflections
- Transparent objects and shadows
  - Objects don’t totally block lights
  - How to attenuate and color light through object?
- Distances for attenuations
- Unintersected rays return background color
- Only shoot reflected/refracted rays when necessary