Texture Mapping: Solid Texturing

CSE 681
Texture Mapping

Visual complexity on demand

Vary display properties over object

Visible **pixel** maps to **location** on object

**Location** on object
used to **lookup** display **attributes**

Or
as **function parameters** to generate **attributes**
Solid Texture Mapping

Object is ‘carved’ out of textured volume

Use x,y,z location of pixel

Use location in simple procedure to generate, e.g.
- Material color to be used in shading calculation
- Ambient, diffuse, or specular reflection coefficient
- Opacity
- Final color

World space coordinates v. object space coordinates?
Solid Texture Map Coordinates

If world space
   Ok in static scenes
   Object moves through texture if object animated

If object space
   Texture is ‘fixed’ to object
   need to inverse transform intersection
   or need to trace inverse ray in object space
Solid Texture Map Coordinates

Object Space

World Space

M

M^{-1}
texture

CSE 681
Space Filling Stripes

Uses: modulo divisor %

\[ \text{jump}(x,y,z) = \left(\left\lfloor \frac{x}{s.x} \right\rfloor + \frac{A}{s.x}\right) \mod 2 \]

if (jump == 0) color = yellow
else if (jump == 1) color = red
Space Filling 2D Checkerboard

$$\text{jump}(x,y,z) = ((\text{int})(A+x/s.x)+(\text{int})(A+y/s.y)) \% 2$$

if (jump == 0)
  color = yellow
Else if (jump == 1)
  color = red
Space Filling 3D Checkerboard

\[
\text{jump}(x, y, z) = ((\text{int})(A+x/s.x) + (\text{int})(A+y/s.y) + (\text{int})(A+z/s.z)) \mod 2
\]

if (jump == 0)
  color = yellow
Else if (jump == 1)
  color = red
Cube of Smoothly Varying Colors

Uses \( \text{fract}(x) = x - (\text{floor})(x) \)

Texture\((x,y,z) = (1 - |2*\text{fract}(x) - 1|, 1-|2*\text{fract}(y) - 1|, 1-|2*\text{fract}(z) - 1|) \)
Rings

\[
\text{rings}(r) = \left(\text{int}(r)\right) \mod 2
\]

\[
r = \sqrt{x^2 + y^2};
\]

\[
\text{rings}(x,y,z) = D + A \times \text{rings}(r/M)
\]

M - thickness

D & A

scale and translate into arbitrary values

Or, as before, map 0 & 1 into yellow and red

CSE 681
Wood Grain

Add twist to rings:

Rotate texture around y-axis by $\theta$

Implement by rotating point by $-\theta$ around y-axis

Similarly, rotate (x,y,z) point around z-axis

Use some randomness to break up regularity
Wood Grain

Make one color much thinner

Make jitter pseudo-random
Noise, Turbulence, Marble

- Define function of random values which is
  - A function of 3D point
  - continuous
  - repeatable

- Use 3D point to retrieve random value

- 3D volume has frequency determined by spacing of random values

- Scale point first to change frequency of noise function
Deposit random values at integer locations

Interpolate through values to get continuous function

Sample function at intersection points of object with ray
1D Noise Example

Sample too frequently - no randomness
1D Noise Example

Sample too sparsely - no continuity

(Nyquist limit)
Turbulence

Add multiple frequencies together

As frequency goes up, amplitude goes down

Each component similar under scale
Fractal
e.g. coastline
1D Turbulence Example
1D Turbulence Example
1D Turbulence Example

CSE 681
1D Turbulence Example
1D Turbulence Example

\[ \sin(x) + 0.5 \sin(2x) + 0.25 \sin(4x) \]
1D Turbulence Example

\[ \sin(x) + 0.5\sin(2x) + 0.25\sin(4x) + 0.125\sin(8x) \]
1D Turbulence Example
1D Turbulence Example
3D Noise

Visible point from surface of object

(x, y, z)

Texture value from 3D table or procedure

Need controlled randomness => varying but continuous function

CSE 681
Integer Lattice

Use 256x256x256 volume

Deposit random values at integer grid points
Interpolate values within cube

\[(x, y, z)\]

\[fx = \text{FRACT}(x)\]
\[fy = \text{FRACT}(y)\]
\[fz = \text{FRACT}(z)\]

Use tri-linear interpolation

\[d00 = d000 + fx(d100 - d000)\]
\[d10 = d010 + fx(d110 - d010)\]
\[d01 = d001 + fx(d101 - d001)\]
\[d11 = d011 + fx(d111 - d011)\]

\[d0 = d00 + fy(d10 - d00)\]
\[d1 = d01 + fy(d11 - d01)\]
\[d = d0 + fz(d1 - d0)\]
Implementation notes

NoiseTable[256]: random values [0, 1]

Index[256]: random permutation of values 0:255

```c
#define PERM(x) index[x & 255]
#define INDEX(ix,iy,iz) PERM(ix + PERM(iy + PERM(iz)))
```

Float latticeNoise(i,j,k)
Return NoiseTable[INDEX(i,j,k)]
Turbulence implementation

Noise(s,x,y,z)
Scale point by s, add 1000 to each coordinate
Get integer (ix,iy,iz) and fractional parts (fx, fy, fz)
Get cell lattice noise values
d000, d001, d010, d011, d100, d101, d110, d111
Do the trilinear interpolation by fx, fy, fz

\[
Turb(s, x, y, z, n1, n2) = \sum_{k=n1}^{n2} \left( \frac{1}{2} \right)^k noise(2^k s, x, y, z)
\]

Where \(n1, n2\) control how many, and which, frequencies
NoiseTexture

See examples
www.cse.ohio-state.edu/~parent/classes/681/Noise/noise.html
Marble Texture

Undulate(x) - basic ripple in x

\[ \text{Marble}(x,y,z) = \text{undulate}(\sin(2\pi xyz + A*\text{turb}(s,x,y,z,k))) \]

Parameters: amplitude, scale, number of frequencies
Marble Texture

See examples
www.cse.ohio-state.edu/~parent/classes/681/SolidTexture/solidTexture.html