Texturing
CS 6965 Fall 2011
Program 2
Texture mapping

- Most real objects do not have uniform color
- Texture: color and other material properties as a function of space
Texture mapping topics

- Image textures
- Texture coordinates
  - Linear, cylindrical, spherical mappings
  - Barycentric coordinates for triangles
  - Software architecture
- Procedural textures
  - Simple: checkerboards, tiles, etc.
  - Fractal noise based: marble, granite, wood, etc.
- Bump mapping

Ken Perlin, NYU
Lambertian Shading

Compute hit position ($\vec{P} = \vec{O} + t\vec{V}$)

Call primitive to get normal ($\vec{N}$) (normalized)

$$costheta = -\vec{N} \cdot \vec{V}$$

if (costheta < 0)

normal = -normal

Color light $=$ scene.ambient*Ka

foreach light source

get $C_L$ and $\vec{L}$

$$dist = ||\vec{L}||, \vec{L}_n = \frac{\vec{L}}{||\vec{L}||}$$

$$cosphi = \vec{N} \cdot \vec{L}_n$$

if (cosphi > 0)

if (!intersect with $0 < t < dist$)

light $+= C_L * (Kd * cosphi)$

result $= light * surface$ color
Simple image texture

Use hit position to determine image location

\( \overline{P} \) : hit position

\( \overline{C} \) : lower left corner of image

\( \overline{U} \) : image X axis in world space

\( \overline{V} \) : image Y axis in world space

x, y: normalized image coordinates (0-1)

\( \overline{P} = \overline{C} + x\overline{U} + y\overline{V} \)

if \( \overline{U} \cdot \overline{V} = 0 \):

\[
x = \frac{(\overline{P} - \overline{C}) \cdot \overline{U}}{\|\overline{U}\|}
\]

\[
y = \frac{(\overline{P} - \overline{C}) \cdot \overline{V}}{\|\overline{V}\|}
\]
Simple image texture

\[
x = \frac{(\bar{P} - \bar{C}) \cdot \bar{U}}{\| \bar{U} \|} \\
y = \frac{(\bar{P} - \bar{C}) \cdot \bar{V}}{\| \bar{V} \|}
\]

\[
i_x = (\text{int})\left(x \left(xres - 1\right)\right), \quad f_x = i_x - x \left(xres - 1\right) \\
i_y = (\text{int})\left(y \left(yres - 1\right)\right), \quad f_y = i_y - y \left(yres - 1\right)
\]

Bilinear interpolation of image\((i_x, i_y)\) - image\((i_x+1, i_y+1)\)
using \(f_x, f_y\) as interpolation factors

Use interpolated color in lambertian shading
Miscellaneous details

• What happens for points outside of the texture?
• Tile texture (mod function)
• Clamp (stretch out last value)
• Use an “outside” color
• What if U/V are not orthogonal?
  • Solve a 2x2 matrix to get x/y
• Other interpolations:
  • Nearest neighbor
  • Higher order interpolation
  • Filter reconstruction (what filter width?)
Texturing process

- This example illustrated two orthogonal issues of texturing for ray tracing:
  - Mapping hit position to image space (coordinate transformation), example: linear
  - Mapping texture coordinate to color or other attributes, example: linear interpolation of an image
Texture coordinates

- Textures are not always flat
- Textures live in a space called “uvw”
- Texture coordinate mapping: xyz to uvw
- Texture space may not be linear

NASA Blue marble project
Cylinder projection

\[ x = \cos(\theta) \]
\[ y = \sin(\theta) \]

inverse mapping:

\[ R = \sqrt{x^2 + y^2} \]
\[ \theta = \text{atan} \ 2(y, x) \]
\[ \text{atan} \ 2 : \text{robust atan}(y/x) \] for all quadrants
Cylinder projection

\[ x = \cos \theta \]
\[ y = \sin \theta \]

inverse mapping:

\[ R = \sqrt{x^2 + y^2} \]
\[ \theta = \text{atan} \ 2(y, x) \]
\[ \text{atan} \ 2 : \text{robust atan}(y/x) \text{ for all quadrants} \]
\[ u = \frac{\theta}{2\pi}; \text{if}(u < 1)u+ = 1, \ v = z, w = R \]
Sphere mapping

- Lots of ways to map a plane (image) to a sphere
- Remember grade school geography

http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj_f.html
Polar projection

Use longitude/latitude angles:

\[ x = \cos \theta \sin \phi \]
\[ y = \sin \theta \sin \phi \]
\[ z = \cos \phi \]

inverse mapping:

\[ R = \sqrt{x^2 + y^2 + z^2} \]
\[ \phi = \cos^{-1} \frac{z}{R} \]
\[ \theta = \text{atan2}(y, x) \]

\text{atan2} : robust \text{atan}(y/x) for all quadrants

\[ u = \frac{\theta}{2\pi}; \text{if}(u<0)u++1, \quad v = 1 - \frac{\theta}{\pi}, \quad w = R \]
Arbitrary position/orientation

\( \vec{Z} \): vector to north pole
\( \vec{X} \): vector to seam
\( \vec{Y} = \vec{Z} \times \vec{X} \)
\( x' = (\vec{P} - \vec{C}) \cdot \vec{X} \)
\( y' = (\vec{P} - \vec{C}) \cdot \vec{Y} \)
\( z' = (\vec{P} - \vec{C}) \cdot \vec{Z} \)

\( x' = \cos \theta \sin \phi \)
\( y' = \sin \theta \sin \phi \)
\( z' = \cos \phi \)

inverse mapping similar
Linear projection

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} =
\begin{bmatrix}
U_x & V_x & W_x \\
U_y & V_y & W_y \\
U_z & V_z & W_z
\end{bmatrix}
\begin{bmatrix}
u \\
v \\
w
\end{bmatrix} +
\begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]

inverse mapping:

\[
\begin{bmatrix}
u \\
v \\
w
\end{bmatrix} =
\begin{bmatrix}
U_x & V_x & W_x \\
U_y & V_y & W_y \\
U_z & V_z & W_z
\end{bmatrix}^{-1}
\begin{bmatrix}
x - T_x \\
y - T_y \\
z - T_z
\end{bmatrix}
\]
Linear special case

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = \begin{bmatrix}
u \\
v \\
w
\end{bmatrix} + \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]

inverse mapping:

\[
\begin{bmatrix}
u \\
v \\
w
\end{bmatrix} = \begin{bmatrix}
x - T_x \\
y - T_y \\
z - T_z
\end{bmatrix}
\]
Barycentric coordinates

\(0 \leq b_1, b_2, b_3 \leq 1\)

\(b_1 + b_2 + b_3 = 1\)

\[
\overrightarrow{P} = b_1 \overrightarrow{P_1} + b_2 \overrightarrow{P_2} + b_3 \overrightarrow{P_3}
\]

\[
= b_1 \overrightarrow{P_1} + b_2 \overrightarrow{P_2} + \left(1 - b_1 - b_2\right) \overrightarrow{P_3}
\]

\(u = b_1\)

\(v = b_2\)

\(w = 0\)
Specified texture coordinates

\[0 \leq b_1, b_2, b_3 \leq 1\]

\[b_1 + b_2 + b_3 = 1\]

\[\overrightarrow{P} = b_1 \overrightarrow{P_1} + b_2 \overrightarrow{P_2} + b_3 \overrightarrow{P_3}\]

\[= b_1 \overrightarrow{P_1} + b_2 \overrightarrow{P_2} + (1 - b_1 - b_2) \overrightarrow{P_3}\]

\[u = b_1 u_1 + b_2 u_2 + (1 - b_1 - b_2) u_3\]

\[v = b_1 v_1 + b_2 v_2 + (1 - b_1 - b_2) v_3\]

\[w = b_1 w_1 + b_2 w_2 + (1 - b_1 - b_2) w_3\]
Other coordinate transforms

- Any of those spherical projections
- Could transform u/v on the disc or ring (similar to cylindrical)
- Conical projections
- Many more
Implementation choices

- Where do these coordinate transforms go?
  - Object? Might want linear or cylindrical mapping on a sphere
  - Material? Material doesn’t know about coordinates
  - Somewhere else?
Implementation tradeoffs

- Object: different sphere subclasses for different UV mappings
- Object: multiple inheritance
- Material: similar tradeoffs
- Somewhere else: redundant data?
- Somewhere else: multiple inheritance
What Steve does

• Add to primitive base class:
  TexCoordMapper* tcmapper;
  void setTexCoordMapper(TexCoordMapper*);
  TexCoordMapper* getTexCoordMapper();

• Sphere is multiply inherited and default tcmapper is “this” pointer

• Objects with no inherent uv space default to linear tcmapper (xyz == uvw)

• Default space can be overridden in scene
Compute hit position ($\bar{P} = \bar{O} + t\bar{V}$)
Call primitive to get tcmapper
Call tcmapper to get UVW coordinates ($\bar{T}$)
Interpolate surface color from image using $\bar{T}$
scaled to image resolution
Call primitive to get normal ($\bar{N}$) (normalized)

costheta = $-\bar{N} \cdot \bar{V}$

if(costheta < 0)
    normal = -normal

Color light = scene.ambient*Ka
for each light source
    get $C_L$ and $\bar{L}$
    
    dist =$\mid\mid\bar{L}\mid\mid$, $\bar{L}_n = \frac{\bar{L}}{\mid\mid\bar{L}\mid\mid}$
    
    cosphi = $\bar{N} \cdot \bar{L}_n$
    
    if(cosphi > 0)
        if(!intersect with 0 < $t$ < dist)
            light += $C_L \times (Kd \times cosphi)$
    
result = light*surface color
Checkerboard texture

\[ i_1 = \text{int}(u \times \text{scale}) \]
\[ i_2 = \text{int}(v \times \text{scale}) \]

\[ \text{cell} = (i_1 + i_2) \% 2 \]

if (\text{cell} = 0) use \text{color1} 
else use \text{color2}
3D Checkerboard texture

\[ i_1 = (\text{int})(u \times \text{scale}) \]
\[ i_2 = (\text{int})(v \times \text{scale}) \]
\[ i_3 = (\text{int})(w \times \text{scale}) \]

\[ \text{cell} = (i_1 + i_2 + i_3) \mod 2 \]

if (cell = 0) use color1
else use color2

http://www.chez.com/jrlivenais/vdesprit/tut_index/check_sphere/check_sphere_eng.htm
Checkboard comparisons

3D checker planar mapping

http://courses.dce.harvard.edu/~cscie234/projects/Slocum/checker-spheres-4x4-jitt-1.gif

2D (or 3D) checker spherical mapping

http://cgg.ms.mff.cuni.cz/~pepca/lectures/textures/sample/checker.jpg
Floor tile texture

\[ s = u \times \text{scale} - (\text{int})(u \times \text{scale}) \]

\[ t = v \times \text{scale} - (\text{int})(v \times \text{scale}) \]

if \((s < \text{grout\_width} \ || \ t < \text{grout\_width})\)

\[ \text{color} = \text{grout\ color} \]

else

\[ \text{color} = \text{tile\ color} \]
Brick texture

\[ ti = (\text{int})(v * vscale) \quad // \text{get brick row} \]
\[ u' = u * uscale - (ti \% 2) * 0.5 \quad // \text{shift column for odd rows} \]
\[ s = u' - (\text{int})u' \quad // \text{brick column fraction} \]
\[ t = v * vscale - ti \quad // \text{brick row fraction} \]

\[ \text{if} (s < \text{mortar}_-\text{width} \lor t < \text{mortar}_-\text{width}) \]
\[ \text{color} = \text{mortar}_-\text{color} \]
\[ \text{else} \]
\[ \text{color} = \text{brick}_-\text{color} \]
More flexible implementation

• Instead of choosing between two (or more) colors, choose between two materials

```c
if(cell == 0)
    matl1->shade(...);
else
    matl2->shade(...);
```
Other simple procedural textures

- Hexagonal mapping
- Cutout shapes
- Use your imagination!
Solid textures

- Many objects are created out of solid materials
- Texture is three-dimensional
- Surface texture is cutaway of the 3D texture space
- Examples:
  - Wood
  - Marble
  - Granite
Perlin noise

- A pseudorandom method for making natural textures
- Appeared in Siggraph 1985 (preview in 1984)
- Ken Perlin won an academy award (technical achievement award) in 1997
- More info: http://noisemachine.com
- Improved in 2002:
  - http://mrl.nyu.edu/~perlin/noise
Perlin noise

- Need a function that:
  - Looks random
  - Has fixed frequency content
  - Is coherent (value changes smoothly from one point to another)
- Solution: smooth random values on a regular grid

http://www.robo-murito.net/code/perlin-noise-math-faq.html
Value noise function

- Idea: place random values on a grid lattice
- Interpolate between them (spline)
- Spline is expensive in 3D
Perlin noise (gradient noise)

- Idea: place random gradients on a grid lattice
- Weighted interpolation of gradient values

\[
0 \leq x \leq 1
\]

\[
g_0 = G_0 x, g_1 = G_1 (1 - x)
\]

\[
noise(x) = lerp(g_0, g_1, ease(x))
\]

Note: \(noise(0) = 0, noise(1) = 0\)
Noise

• Properties of good noise functions
  • Repeatable
  • Bandlimited
  • Stays within known limits
  • Isotropic
Value Noise

Take a continuous random process.
(white noise)
Value Noise

Sample it.
Convolve the samples with a reconstruction filter.
Value Noise

This produces basic value noise.
Value Noise

This produces basic value noise.
Perlin’s Insight

Value noise just multiplies the kernel with a simple constant.
Suppose instead that we multiplied the kernel with a gradient?
Perlin Noise

Instead of random intensities at each point, use random unit vectors.
The result is Perlin noise.
The result is Perlin noise.
Classic Implementation

// P[n] = permutation table: numbers 0 through n-1, shuffled
// G[n] = gradients: random unit vectors
float noise(point S)
{
    I = floor(S)
    F = S - I
    ease = 3F^2 - 2F^3
    for each corner, C, of the containing cube:
        hash = P[C_x + P[C_y + P[C_z mod n] mod n] mod n] mod n]
        dot = G[hash] . (S-C)
    return trilinear(dot, ease)
}
“Improved” Perlin

- Higher order ease function: $6F^5 - 15F^4 + 10F^3$
- Evenly distribute gradients: use just 12 vectors to center of cube edges
2D perlin noise

- Sum over each corner: weighted gradient dot vector to point

http://www.robo-murito.net/code/perlin-noise-math-faq.html
Ease curve

- Get rid of discontinuities at grid boundaries
- Interpolate with hermite polynomial

First derivative == 0 at t=0 and t=1

- Linear
- $2t^3 - 3t^2$
Faster way

- Randomly choose a gradient vector based on a hash function:
  - Precompute table of permutations $P[n]$
  - Precompute table of gradients $G[n]$
  - $G = G[ (i + P[ (j + P[k]) \mod n ]) \mod n]$
2002 implementation

- Randomness comes from permutation and random gradients
- Random gradients are not necessary
- Use gradients to edges of a cube:
  - \([0 \ 1 \ 1] \ [0 \ 1 \ -1] \ [0 \ -1 \ 1] \ [0 \ -1 \ -1]\)
  - \([1 \ 0 \ 1] \ [1 \ 0 \ -1] \ [-1 \ 0 \ 1] \ [-1 \ 0 \ -1]\)
  - \([1 \ 1 \ 0] \ [1 \ -1 \ 0] \ [-1 \ 1 \ 0] \ [-1 \ -1 \ 0]\)
- Use better ease function:

\[6t^5 - 15t^4 + 10t^3\]
Comparison

- 2002 implementation is a little faster and looks better
Easy way

- Perlin’s 2002 reference implementation (Java)
- http://mrl.nyu.edu/~perlin/noise/
Improved Noise

- Better gradient noise
- Software implementation
- Hardware-accelerated gradient noise for graphics
  - VLSI implementation
  - 4-stage pipeline at 1 GHz
  - www.cs.utah.edu/~aek/research/hwnoise.pdf
Fractal noise

- Perlin noise is (nearly) fixed frequency
- Defined by the grid
- Fractal noise:
  \[ \text{turbulence}(x) = \text{noise}(x) + \frac{1}{2} \text{noise}(2x) + \frac{1}{4} \text{noise}(4x) + \ldots \]

More generally:

\[ \text{turbulence}(x) = \sum_{i=0}^{\text{octaves}} \frac{a^i \text{noise}(2^i)}{b^i} \]

- \(a\): lacunarity
- \(b\): gain or persistence
Fractal noise

Using noise

- Creativity is in using functions of noise or turbulence

\[ \sin(x + \text{turb}(x)) \]
Marble

- Swirled veins of molten rock

\[ \sin(40x) \times 0.5 + 0.5 \]
Marble

- Swirled veins of molten rock

\[ \sin(40x + \text{noise}(4x)) \times 0.5 + 0.5 \]
Marble

- Swirled veins of molten rock

$\sin(40x + 5\text{noise}(4x)) \times 0.5 + 0.5$
Marble

- Swirled veins of molten rock

\[ \sin(40x + 10\text{noise}(4x)) \times 0.5 + 0.5 \]
Marble

- Swirled veins of molten rock

\[
\sin(40x + 16\text{noise}(4P) + 8\text{noise}(8P)) \times 0.5 + 0.5 = \\
\sin(40x + 16\text{turbulence}(2, 4P, 2, 0.5)) \times 0.5 + 0.5
\]
Marble

= \sin(40x + 16 \text{turbulence}(3, 4, 2, 0.5)) \times 0.5 + 0.5

- Swirled veins of molten rock
Marble \[=\sin(40x+16\text{turbulence}(4, 4P, 2, 0.5)) \times 0.5 + 0.5\]

- Swirled veins of molten rock
Marble

\[ = \sin(40x + 16\text{turbulence}(6, 4P, 2, 0.5)) \times 0.5 + 0.5 \]

- Swirled veins of molten rock
Marble

\[ = 1 - (\sin(40x + 16\text{turbulence}(6, 4, 2, 0.5))\times 0.5 + 0.5)^6 \]

- Swirled veins of molten rock
Advanced marble

- Use \(|\text{noise}|\) instead of noise
- Makes sharper features
- Use different colors
  - Interpolate between two colors
  - General spline function
Wood texture

- Trees are concentric cylinders of growth
Wood cuts:
Quarter-sawn
Flat-sawn
Rift-sawn
Rotary sliced
Wood texture

- Dark rings are a little further apart

\[ \cos(r) \]
Wood texture

- Rings aren’t perfect

\[ \cos(R + \text{noise}(P))^8 \]
Wood texture

- Large imperfections make the wood “curly”
Other improvements

- Modulate center based on noise
- Model finer grain structure of wood grain
- Model rays
Advanced wood

- Render grain
- Better transitions
- Shader from Advanced Renderman has 11 parameters
Wooden bunny

- Marble base and simple wood texture on bunny
Other noise tricks

• Embed noise in higher dimension and animate (2D ->3D, 3D->4D)
• Fire burning
• Clouds evolving
Other procedural textures

- Granite
- Stucco
- Dented/Eroded
- Flames
- Clouds
- Many more - use your imagination!

noisemachine.com
Combinations of shaders

- Combine shaders to get even better results
Recipe for success

- Add method to Primitive based class to compute texture coordinates:
  
  ```cpp
  void computeUVW(Vector& uvw, const RenderContext& context, const Ray& ray, const HitRecord& hit) const;
  ```

- Default implementation sets `uvw` as hit position:
  
  ```cpp
  uvw = (ray.origin + ray.direction*t)
  ```
Polar Coordinates

- Used for left two spheres (with maps)
- Create subclass of Sphere (SpherePolar)
- Override computeUVW method to compute polar coordinates
- In constructor:

\[
\vec{Z} = \text{poleAxis} \quad (4\text{th argument})
\]
\[
\vec{X} = \text{primeMeridianAxis} \quad (5\text{th argument})
\]
\[
\vec{Y} = \vec{Z} \times \vec{X}
\]

normalize \( \vec{X}, \vec{Y}, \) and \( \vec{Z} \)
Polar coordinates

\[ \vec{P} = \vec{O} + t\vec{V} \]
\[ \vec{P}' = \vec{P} - \vec{C} \]
\[ x = \vec{P}' \cdot \vec{X}, y = \vec{P}' \cdot \vec{Y}, z = \vec{P}' \cdot \vec{Z} \]
\[ R = \sqrt{x^2 + y^2 + z^2} \]
\[ \theta = \text{atan} 2(y, x) \]
\[ u = \frac{\theta}{2\pi} \]
\[ \text{if } (u < 0) u^+ = 1 \]
\[ \phi = \text{arccos} \left( \frac{z}{R} \right) \]
\[ v = 1 - \frac{\phi}{\pi} \]
\[ uvw = \text{Vector}(u, v, R) \]
Checkerboard texture

Checkerboard a subclass of Material

In constructor:

Given: $\vec{C}$ (center of checkerboard), $\vec{V}_1, \vec{V}_2$ (checker axis directions)

$\vec{V}_1$ should be vector to first checker vertex

We want to switch checker vertices at integers 1, 2, ...

$\therefore \vec{V}_1 \cdot \vec{V}_1' = 1$

$\vec{V}_1' = \frac{\vec{V}_1}{\|\vec{V}_1\|^2}, \vec{V}_2' = \frac{\vec{V}_2}{\|\vec{V}_2\|^2}$
Checkerboard texture

Call computeUVW on hit primitive: $uvw$

\[ i_1 = \text{floor}(uvw \cdot V_1') \]

\[ i_2 = \text{floor}(uvw \cdot V_2') \]

\[ \text{which_matl} = (i_1 + i_2) \% 2 \]

\[ \text{if} (\text{which_matl} == 0) \]
\[ \text{matl0->shade(....)} \]
\[ \text{else} \]
\[ \text{matl1->shade(....)} \]
Checkerboard optimization

- Put materials in a two-element array, indexed by which_matl
- Beware (-1)%2 = -1
- One solution: make a three-element array, indexed by which_matl+1
- matls[which_matl+1]->shade(…)

Phong Marble Material

Compute $uvw$ using hit primitive

$$\bar{T} =uvw \ast scale \ast fscale$$

$$value = \cos(u \ast scale + tscale \ast turbulenceAbs(octaves, \bar{T}, lacunarity, gain))$$

$$value = value \ast 0.5 + 0.5$$

$$color = c_1(1 - value) + c_2 value$$

Use color in phong lighting calculations

Parameters:

scale = spatial magnification

fscale = frequency magnification of turbulence (smaller ~ coarser structure)

tscale = distortion due to marble (0: pure sinusoid, large: more veins)

octaves = number of frequencies in fractal turbulence

lacunarity = frequency multiplication

gain = decay of higher frequencies in fractal turbulence
Phong Image Material

Compute $uvw$ using hit primitive

\[ x = u \times xres \]
\[ y = v \times yres \]
\[ ix = \text{Floor}(x), iy = \text{Floor}(y) \]
\[ fx = x - ix, fy = y - iy \]

Interpolate color from image(ix,iy) using fx,fy

Use color in phong shading calculations
Image wrapping

Need to interpolate at prime meridian:

Interpolation:

\[\text{image}(x,y)(1-fx)(1-fy)\]
\[+\text{image}((x+1)\%xres, y)(fx)(1-fy)\]
\[+\text{image}(x,(y+1)\%yres)(1-fx)(fy)\]
\[+\text{image}((x+1)\%xres,(y+1)\%yres)(fx)(fy)\]

This still isn't quite right, but it works fine for this assignment.

You probably won't notice the difference unless you use a smaller image.
Floor

- Note that Floor(x) != (int)x
- Wrong for x<0
- Integer cast actually undefined for x<0
- Most machines round toward 0 (not what we want)

Correct Floor implementation:

```cpp
inline int Floor(float d) {
    if(d<0){
        int i = -static_cast<int>(-d);
        if(i == d)
            return i;
        else
            return i-1;
    } else {
        return static_cast<int>(d);
    }
}
```