cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

texture mapping

"glue" image to surface

---

![Texture Mapping Image](image1)

![Texture Mapping Triangle](image2)
What is \( f(p) \)?

A point \( q \) on the triangle \( T \) can be uniquely represented as \( q = t_0 + \beta u + \gamma w \) where \( \beta \geq 0 \), \( \gamma \geq 0 \), \( \beta + \gamma \leq 1 \).

Computing \( f(p) \):

- Compute \( f(u) = f(t_1) - f(t_0) \) and \( f(w) = f(t_2) - f(t_0) \).
- Compute \( f(p) = f(t_0) + \beta f(u) + \gamma f(w) \).

What is image color at \( f(p) \)?

Need to resample! For your ray tracer use bilinear interpolation.

Using the color:

1. Use as pixel color.
2. Use as diffuse and specular coefficient of surface at \( p \).
3. Use as diffuse coefficient of surface at \( p \).
a point p on the surface can be represented relative to c and r, t, u.

\[ p - c = \alpha r + \beta t + \gamma u \]

where
\[ \alpha = r \cos(\theta_p) \cos(\phi_p) \]
\[ \beta = r \cos(\theta_p) \sin(\phi_p) \]
\[ \gamma = r \sin(\theta_p) \]

\( \theta_p \) and \( \phi_p \) are the angles in degrees and \( w, h \) are the image width and height in pixels.

\[ f(p) = (w_{\theta_p}/360, h_{\phi_p}/360) \]

as before

- resample image
- use color as...
problems

• given \( p \) compute \( \phi, \theta \)
  
you can do that!

• poles
  
test for pole and use default texture coordinate

• seams
  
  - use good textures
  
  - overlap & blend or mix
  
  - don’t look there
  
  - 3d textures

3d textures

• use stack of images

how do we generate these images?

cheap tricks

• texture mapping

• procedural texture mapping

• bump mapping

• transparency mapping

• depth of field

• lens effects

• jittering

• soft shadows

procedural texture mapping

• procedure returns a texture color for any point in 3d space (note this is not an image stack)

• sample to find texture for surface

procedural textures

• advantages
  
  - don’t need to find a mapping from a (complex) 3d surface to a 2d texture image

  - concise representation of texture

• disadvantages
  
  - ad hoc techniques cannot duplicate photographs
### Perlin Noise - 1D Example

**Step 1:** Generate discrete noise function with specified length, amplitude, sampling frequency.

Example: length=8, amplitude = 3, sampling frequency is 7 Hz.

- Random number generator
  - \( r_0, r_1, \ldots, r_7 \)
  - \( r_i \in [0,1] \)

\[ 3 \]

\[ t \]

**Step 2:** Interpolate with smoothing.

**Step 3:** Repeat with various amplitudes/frequencies.

**Step 4:** Add together.

---

### Creating Bumpy Surfaces

- Texture mapping
- Procedural texture mapping
- Bump mapping
- Transparency mapping
- Depth of field
- Lens effects
- Jittering
- Soft shadows

---

For more info see Perlin Noise link on proj2 web site.

---

Adrian Mettler, Spring 2003
bump mapping vs texture mapping

- bump mapping effects change with lighting changes
- texture mapping is computationally easier

displacement mapping

displacement map=height field

surface

bumpy
surface

q = p + f(p)n

bump mapping intuition

a surface appears to have a bump surface
- jagged silhouette
- surface normals fluctuate across surface

simulate this by perturbing normals in the lighting calculations

bump mapping

use regular surface but normal of bumpy surface

bumpy surface

q = p + f(p)n

we'll assume our bumpy surface is locally smooth so normals are well-defined

bump mapping

computing \( \mathbf{n}_q \):
1. find vectors \( \mathbf{v}_0 \) and \( \mathbf{v}_1 \) in plane tangent to bumpy surface at point \( q \)
2. \( \mathbf{n}_q = (\mathbf{v}_0 \times \mathbf{v}_1)/||\mathbf{v}_0 \times \mathbf{v}_1|| \)
find vectors in tangent plane

take partial derivatives of $Q$ in two directions

what directions?

$Q + \Delta$

(0,0,0)

2d parameterization of original surface

$p(\phi, \theta)$

$p(\beta, \gamma)$

parameterization

$2d$ parameterization of surface

$u =$ direction of constant $\phi$

$w =$ direction of constant $\gamma$

note: by direction we mean a unit vector
parameterization

\[ u = (t_1-t_0)/(t_2-t_0) \] -- this is a little different than our parameterization for triangle intersection

\[ p(\beta, \gamma) = t_0 + \beta (t_2-t_0) + \gamma (t_2-t_0) \]

\[ \begin{align*}
  t_0 & \quad p(\beta, \gamma) & \quad t_2 \\
  \end{align*} \]

2d parameterization of surface

\[ p(u, w) \]

\[ (0,0,0) \]

find vectors in tangent plane

find vectors in tangent plane

find vectors in tangent plane

find vectors in tangent plane

\[ u = (t_1-t_0)/|t_2-t_0| \] -- this is a little different than our parameterization for triangle intersection

\[ p(\beta, \gamma) = t_0 + \beta (t_1-t_0) + \gamma (t_2-t_0) \]

\[ \begin{align*}
  t_0 & \quad p(\beta, \gamma) & \quad t_1 & \quad t_2 \\
  \end{align*} \]

\[ u = (t_1-t_0)/|t_2-t_0| \] -- this is a little different than our parameterization for triangle intersection

\[ p(\beta, \gamma) = t_0 + \beta (t_1-t_0) + \gamma (t_2-t_0) \]

\[ \begin{align*}
  t_0 & \quad p(\beta, \gamma) & \quad t_1 & \quad t_2 \\
  \end{align*} \]

\[ u = (t_1-t_0)/|t_2-t_0| \] -- this is a little different than our parameterization for triangle intersection

\[ p(\beta, \gamma) = t_0 + \beta (t_1-t_0) + \gamma (t_2-t_0) \]

\[ \begin{align*}
  t_0 & \quad p(\beta, \gamma) & \quad t_1 & \quad t_2 \\
  \end{align*} \]

We can compute \( P_u \) and \( P_w \) for our surfaces.
triangle: parametric form

\[ P_u = \lim_{\delta \to 0} \left( (t_0 + \beta \delta u + w) - (t_0 + \beta u + w) \right) / \| \delta u \| = \beta u \]

find vectors in tangent plane

\[
\begin{align*}
Q_u &= P_u + \left( \frac{df(u,w)}{du} n(u,w) + f(u,w) \frac{dn(u,w)}{du} \right) \\
Q_w &= P_w + \left( \frac{df(u,w)}{dw} n(u,w) + f(u,w) \frac{dn(u,w)}{dw} \right)
\end{align*}
\]

bump map derivative

convolution kernel

<table>
<thead>
<tr>
<th>-1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

change in \( u \) | change in \( w \)

find vectors in tangent plane

\[
\begin{align*}
Q_u &= P_u + b_u(u,w) n(u,w) + f(u,w) \frac{dn(u,w)}{du} \\
Q_w &= P_w + b_w(u,w) n(u,w) + f(u,w) \frac{dn(u,w)}{dw}
\end{align*}
\]

we'll ignore this because:

(a) it is small,

(b) it is computationally difficult, and

(c) results look ok if we do.

we can do this!
bump mapping

1. Compute derivatives of surface $P_u$ and $P_v$
2. Compute derivatives of bump map $b_u(u,v)$ and $b_v(u,v)$
3. Take cross products and add

Computation

1. Compute derivatives of surface $P_u$ and $P_v$
2. Compute derivatives of bump map $b_u(u,v)$ and $b_v(u,v)$
3. Take cross products and add

cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

Texture specifies transparency of surface

2d parameterization of surface

take the cross product

take partial derivatives of $Q(u,v)=P(u,v)+f(u,v)n(u,v)$ with respect to $u,v$

$Q_u = P_u + b_u(u,v)n(u,v)$
$Q_v = P_v + b_v(u,v)n(u,v)$

take cross product

$Q_u \times Q_v = (P_u \times P_v) + b_u(u,v)P_u \times n(u,v) + b_v(u,v)P_v \times n(u,v)$

We'll assume our bumpy surface is locally smooth so normals are well-defined.
cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

jittering: anti-aliasing technique

- cast several rays through pixel neighborhood into scene
- for each:
  - find intersection point (if any) that is closest to eye
  - compute luminance at intersection
- compute average luminance

See paper mentioned in assignment

cheap tricks

- texture mapping
- procedural texture mapping
- bump mapping
- transparency mapping
- depth of field
- lens effects
- jittering
- soft shadows

jittering: antialiasing technique

Run rt for example
soft shadows

run rt for examples

soft shadows

use jittering in occlusion test

ray tracing

- simple ray casting
- recursive ray tracing
- modeling transforms
- cheap tricks
- optimizations