Texture mapping

Polygon mapping
- Forwards and backwards methods

The use of intermediate surfaces
- Plane
- Cylinder
- Sphere

Bump mapping

Problems - aliasing

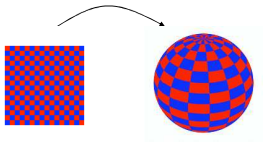
Texture mapping

Three types of mapping

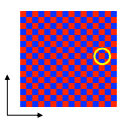
- Texture image mapping
  - Uses images to fill inside of polygons
- Environment ("reflection" mapping)
  - Uses a picture of the environment for texture maps
- Bump mapping
  - Emulates altering normal vectors during the rendering process

Texture image mapping

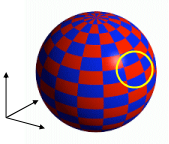
- Inputs:
  - 2D image of a texture
  - 3D surface onto which to map the texture (intermediate)
  - 2D projection of the 3D surface
- Process:
  - Relate a spot on the texture with 2D projection of a vertex



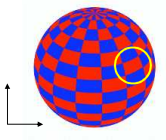
Texture image mapping



2D texture map

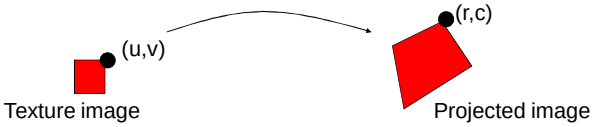


3D surface



2D projection of the surface

Texture image mapping
Projective mapping



General formulas:

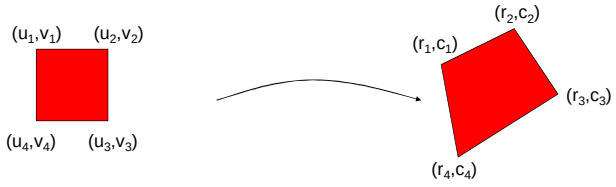
Homogeneous Coordinates of point (u,v)

$$1.\quad \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} r \\ c \\ 1 \end{bmatrix}$$

$$2.\quad \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} U/W \\ V/W \end{bmatrix}$$

usually can be assumed to be 1

Texture image mapping
Projective mapping



- Every polygon pixel is given two sets of coordinates
  - Projected image coordinates (r,c) (row and column)
  - Texture coordinates (u,v) (location in texture image from which the pixel's colour is to be "grabbed").

Texture image mapping
Projective mapping

- Two approaches
- Forward mapping
  - Copy pixel at texture source (u,v) to image destination (r,c)
  - Easy to compute but may leave holes
- Backward mapping
  - For image pixel (r,c) grab texture pixel at (u,v)
  - No holes but computation is harder

Backward mapping

Projective mapping

$u = (a_{11}r + a_{12}c + a_{13})/(a_{31}r + a_{32}c + 1)$

$v = (a_{21}r + a_{22}c + a_{23})/(a_{31}r + a_{32}c + 1)$

a square can be mapped to an arbitrary quadrilateral

Inputs: (r,c)
Outputs: (u,v)

There are a total of 8 parameters in this mapping
$(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32})$

The projective mapping is the most general 2D linear map

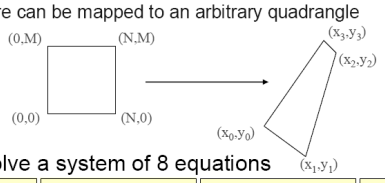
Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

How do we compute the parameters for a given polygon?

$u = (a_{11}r + a_{12}c + a_{13})/(a_{31}r + a_{32}c + 1)$

$v = (a_{21}r + a_{22}c + a_{23})/(a_{31}r + a_{32}c + 1)$

a square can be mapped to an arbitrary quadrangle

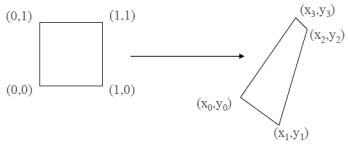


Answer: Solve a system of 8 equations

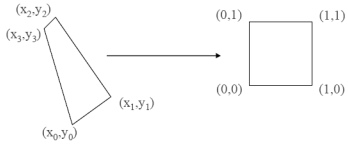
Eqs. for corner #0	Eqs. for corner #1	Eqs. for corner #2	Eqs. for corner #3
$0 = \frac{a_{11}x_0 + a_{12}y_0 + a_{13}}{a_{31}x_0 + a_{32}y_0 + 1}$	$N = \frac{a_{11}x_1 + a_{12}y_1 + a_{13}}{a_{31}x_1 + a_{32}y_1 + 1}$	$N = \frac{a_{11}x_2 + a_{12}y_2 + a_{13}}{a_{31}x_2 + a_{32}y_2 + 1}$	$0 = \frac{a_{11}x_3 + a_{12}y_3 + a_{13}}{a_{31}x_3 + a_{32}y_3 + 1}$
$0 = \frac{a_{21}x_0 + a_{22}y_0 + a_{23}}{a_{31}x_0 + a_{32}y_0 + 1}$	$0 = \frac{a_{21}x_1 + a_{22}y_1 + a_{23}}{a_{31}x_1 + a_{32}y_1 + 1}$	$M = \frac{a_{21}x_2 + a_{22}y_2 + a_{23}}{a_{31}x_2 + a_{32}y_2 + 1}$	$M = \frac{a_{21}x_3 + a_{22}y_3 + a_{23}}{a_{31}x_3 + a_{32}y_3 + 1}$

Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

- Case #1: Solve system for the unit-square-to-quadrilateral case

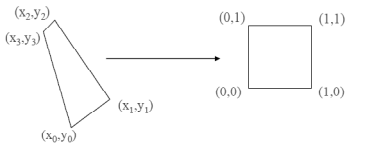


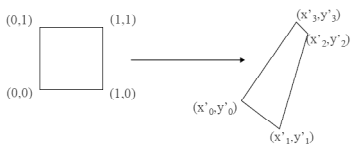
- Case #2: Solve system for the quadrilateral-to-unit-square case



Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

Quadrilateral-to-quadrilateral case solved by combining previous 2 special cases:

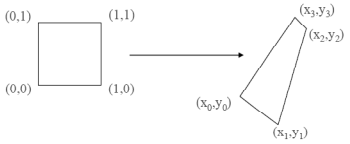




Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

The Projective Mapping: Case #1 Solution

(Wolberg, 1990)



Step 1: Define the following:

$\Delta x_1 = x_1 - x_2 \qquad \Delta y_1 = y_1 - y_2$

$\Delta x_2 = x_3 - x_2 \qquad \Delta y_2 = y_3 - y_2$

$\Delta x_3 = x_0 - x_1 + x_2 - x_3 \qquad \Delta y_3 = y_0 - y_1 + y_2 - y_3$

Step 2a: Solution if $\Delta x_3 = 0$ and $\Delta y_3 = 0$:
(Affine case, where square maps to parallelogram)

$a_{11} = x_1 - x_0 \qquad a_{21} = x_2 - x_1 \qquad a_{31} = x_0$

$a_{12} = y_1 - y_0 \qquad a_{22} = y_2 - y_1 \qquad a_{32} = y_0$

$a_{13} = 0 \qquad a_{23} = 0$

Step 2b: Solution if $\Delta x_3 \neq 0$ or $\Delta y_3 \neq 0$:
(Projective case, where square maps to a quadrilateral that is not a parallelogram)

$$a_{13} = \frac{\begin{vmatrix} \Delta x_3 & \Delta x_2 \\ \Delta y_3 & \Delta y_2 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}} \qquad a_{23} = \frac{\begin{vmatrix} \Delta x_1 & \Delta x_3 \\ \Delta y_1 & \Delta y_3 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$$

$a_{11} = x_1 - x_0 + a_{13}x_1 \qquad a_{21} = x_3 - x_0 + a_{23}x_3$

$a_{12} = y_1 - y_0 + a_{13}y_1 \qquad a_{22} = y_3 - y_0 + a_{23}y_3$

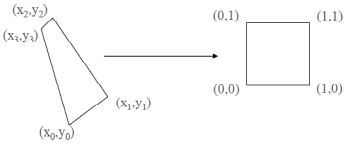
$a_{31} = x_0 \qquad a_{32} = y_0$

recall that $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$

Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

The Projective Mapping: Case #2 Solution

(Wolberg, 1990)



Step 1: Compute the square-to-quadrilateral parameters as specified in Case #1:

$a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}$

Step 2: Compute the intermediate parameters A'_{11}-A'_{33} as follows:

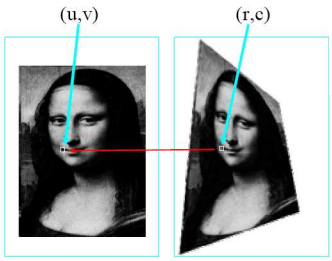
$$A'_{11} = a_{22} - a_{23}a_{32} \quad A'_{12} = a_{13}a_{32} - a_{12} \quad A'_{13} = a_{12}a_{23} - a_{13}a_{22}$$
$$A'_{21} = a_{23}a_{31} - a_{21} \quad A'_{22} = a_{11} - a_{13}a_{31} \quad A'_{23} = a_{13}a_{21} - a_{11}a_{23}$$
$$A'_{31} = a_{21}a_{32} - a_{22}a_{31} \quad A'_{32} = a_{12}a_{31} - a_{11}a_{32} \quad A'_{33} = a_{11}a_{22} - a_{12}a_{21}$$

Step 3: Compute the 8 quad-to-square parameters:

$$A_{ij} = \frac{A'_{ij}}{A'_{33}}$$

Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

General Linear Backward Mapping



Backward mapping algorithm:

for (r,c) = polygon pixel

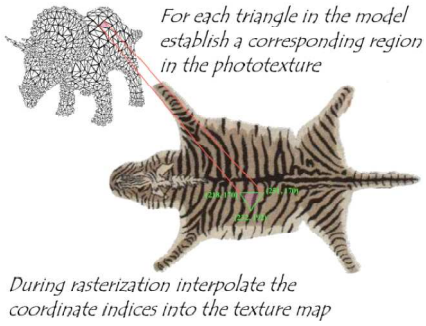
$u = (a_{11}r + a_{12}c + a_{13})/(a_{31}r + a_{32}c + 1)$

$v = (a_{21}r + a_{22}c + a_{23})/(a_{31}r + a_{32}c + 1)$

copy pixel at source (u,v) to destination (r,c)

Source: http://www.utm.toronto.edu/~arnold/320/04s/lectures/15/lecture-15.pdf

Texture image mapping

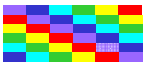


http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

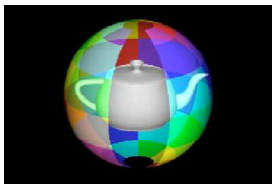
Inverse mapping using an intermediate surface

Two-part mapping:

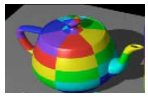
Texture image



Intermediate parametric surface
e.g. plane, cylinder, sphere



Projected image of a 3D object



Source: http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/

Inverse mapping using an intermediate surface

Texture image

Intermediate parametric surface
e.g. plane, cylinder, sphere

3D object

2D screen projection

$T(u,v)$

$T'(x_i,y_i,z_i)$

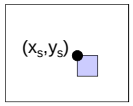
$O(x_w,y_w,z_w)$

$P(x_s,y_s)$

Backward mapping

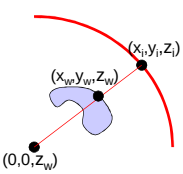
Inverse mapping – example for a cylinder

$P(x_s,y_s)$



Point (x_s,y_s) in the screen image ...

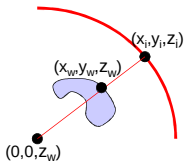
$O(x_w,y_w,z_w)$



... related to point (x_w,y_w,z_w) on a 3D object

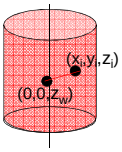
Inverse mapping – example for a cylinder

$O(x_w,y_w,z_w)$



... projected onto a point (x_i,y_i,z_i) on a cylinder surface

$T'(x_i,y_i,z_i)$



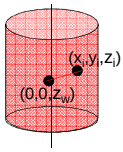
$$(\theta, h) = (\tan^{-1}(y_w/x_w), z_w)$$

$$x_i = r\cos\theta$$
$$y_i = r\sin\theta$$
$$z_i = h$$

r - cylinder radius
$h = z_i = z_w$

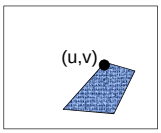
Inverse mapping – example for a cylinder

$T'(x_i,y_i,z_i)$



... de-referenced to a point (u,v) on a texture patch

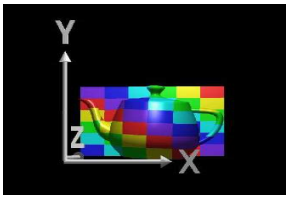
$T(u,v)$

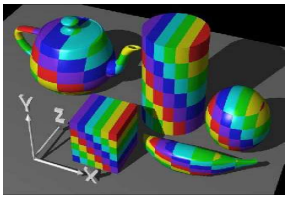


$$(u, v) = (\frac{r}{c_1}(\theta - \theta_0), \frac{1}{c_2}(h - h_0))$$

c_1, c_2 – scaling factors
h_0, θ_0 – origin of the texture on the cylinder
r - cylinder radius
$h = z_i = z_w$

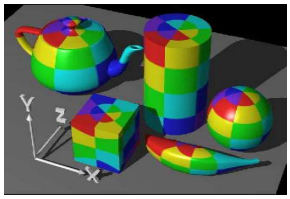
Texture mapping: plane





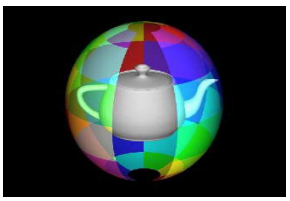
Source: http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/

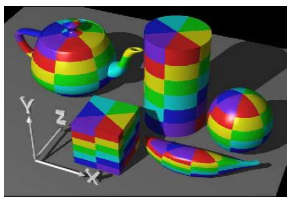
Texture mapping: cylinder



Source: http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/

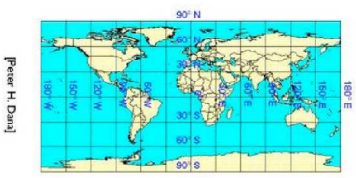
Texture mapping: sphere





Source: http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/

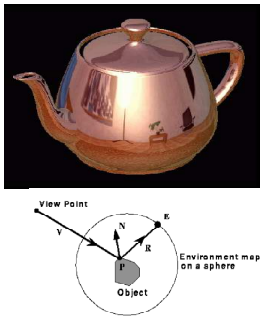
Texture mapping: sphere



Source: http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/

Texture mapping: environment maps

- Instead of using the ray from the surface point to the projected texture's centre, we use the direction of the reflected ray to index a texture map
- This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.

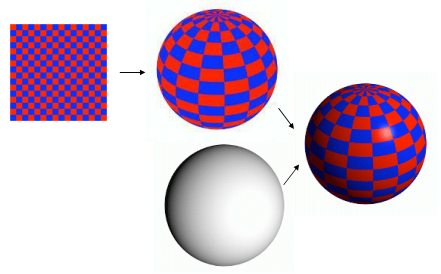


http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

Texture mapping as shading

- Texture mapping can be used to alter some or all of the constants in the illumination equation
- Texture is used as the final colour for the pixel

Texture mapping as shading



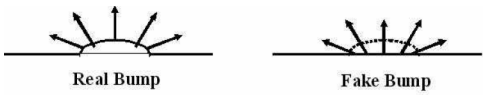
$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{lights} I_i \left(k_d \left(\hat{N} \cdot \hat{L} \right) + k_s \left(\hat{V} \cdot \hat{R} \right)^{n_{shiny}} \right)$$

Phong's Illumination Model

http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

Bump mapping

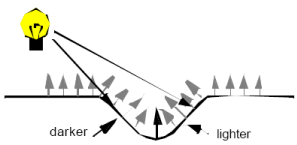
- Methods for making a surface look rough
- Several approaches
  - Perturb the surface vertices (real bumps)
  - Perturb the surface normals (fake bumps)

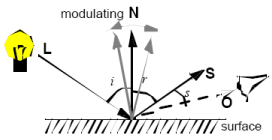


http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

Bump mapping

- Surface normal vectors (e.g. computed using Phong method) for each pixel simulate 'bumps' or other small texture irregularities
- By artificially altering the surface normal vectors, we alter the amount of light reaching the observer

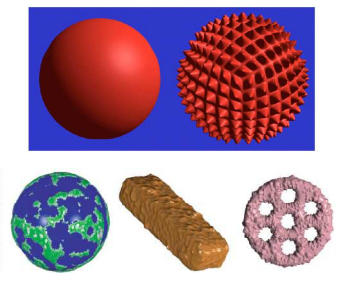




http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

Bump mapping

- Use texture map to actually displace surface points



http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

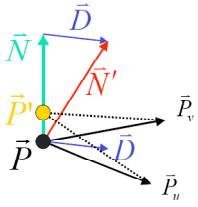
Bump mapping

If Pu and Pv are orthogonal and N is normalized:

$\vec{P} = [x(u,v), y(u,v), z(u,v)]^T$ Initial point

$\vec{N} = \vec{P}_u \times \vec{P}_v$ Normal

$\vec{P}' = \vec{P} + B(u,v)\vec{N}$ Simulated elevated point after bump



$\vec{N}' \simeq \vec{N} + \underbrace{B_u\vec{P}_u + B_v\vec{P}_v}_{\vec{D}}$

Variation of normal in u direction

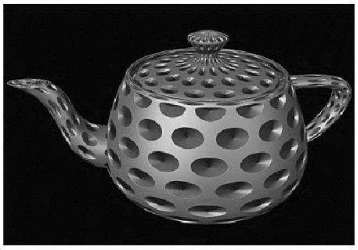
$B_u = \frac{B(s-\Delta,t) - B(s+\Delta,t)}{2\Delta}$

$B_v = \frac{B(s,t-\Delta) - B(s,t+\Delta)}{2\Delta}$

Variation of normal in v direction

Compute bump map partials by numerical differentiation

Bump mapping

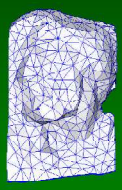


http://www.avl.iu.edu/~ewernert/b581/lectures/15.1/bumpmap1.jpg

Bump Mapping

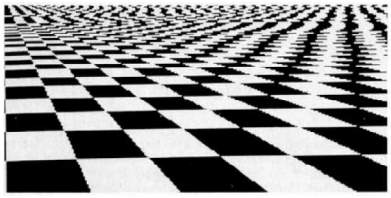
- In CG, we can perturb the normal vector without having to make any actual change to the shape.
- What would be the problems with bump mapping?

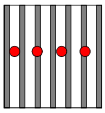

Original model (5M)


Simplified (500)


Simple model with bump map

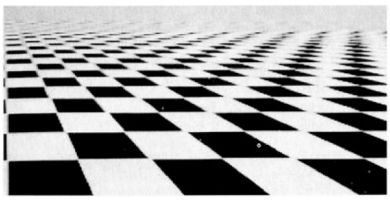
Problems - aliasing




sampling point ?

http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

Anti-aliasing methods

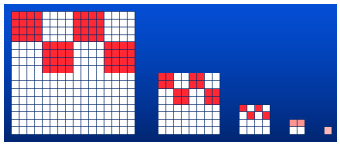


- Interpolation (bi-linear, tri-linear)
- Filtering
- Multi-resolution sampling (e.g. MIP maps)

http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf

MIP-mapping

- MIP = Multim in Parvo (many things in a small place)
- Idea: store texture as a pyramid of progressively lower-resolution images, filtered down from original
- Points further away are sampled from a lower-resolution MIP-map



http://www.cs.drexel.edu/~david/Classes/CS431/Lectures/CGII_Pres7.pdf