Texture Mapping

2D Texture Space

↓ **Parameterization**

3D Object Space

↓ **Model Transformation**

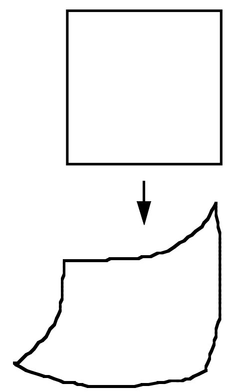
3D World Space

↓ **Viewing Transformation**

3D Camera Space

↓ **Projection**

2D Image Space



Barr/Catmull-Smith Framework

- **Parameterization - Deformation of parameter space**
- **Deformation - Bending and twisting**
- **Non-linear camera projections - Omnimax/Fisheye**

The whole process may be modeled as a complicated non-linear deformation of parameter space. Solve!!

Implement an image synthesis machine as based on an image warping engine.

Mapping Algorithms

Forward mapping (splatting, blots, ADO)

```
for all (u,v)
  (x,y) = f(u,v)
  image[x][y] = texture[u][v]
```

Backward mapping (RE)

```
for all (x,y)
  (u,v) = f^-1(x,y)
  image[x][y] = texture[u][v]
```


Comparison

Forward Mapping

- \+ Efficiently implemented in current architectures
- \+ Texture applied in object space, before shading
- \+ Forward transform may be simpler than backward
- \- Resampling
  - May leave gaps in image
  - Footprint, but must normalize
  - Adaptive sampling - micropolygons

Backward Mapping

- +- Similar to ray tracing
- \- Backward transform may be difficult to invert
- \- Resampling
  - Spatially varying filter

Both routinely used in volume rendering

Polygonal Mapping

Parameterization may be written as a linear map

$$P = P_0 + uP_1 + vP_2$$

The transformation from texture space to image space be performed with a single 2d projective matrix

$$(x, y, w) = P((V(M(T(u,v,1))))) = (PVMT)(u,v,1) \qquad \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

The inverse transformation from image space to texture space is also a 2d projective matrix

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

Incremental Texture Mapping Loop

```
x = xmin;
unum = A*x+B*y+C;
vnum = D*x+E*y+F;
den  = G*x+H*y+I;
for( ; x<xmax; x++ ) {
  image[x][y]=texture(unum/den,vnum/den);
  unum += A;
  vnum += D;
  den  += G;
}
```

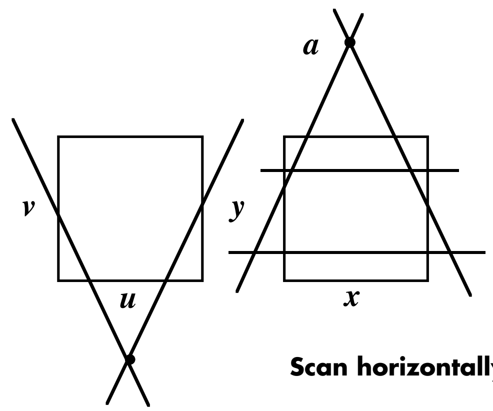
$$u = \frac{unum}{den} = \frac{Ax + By + C}{Gx + Hy + I}$$

$$v = \frac{vnum}{den} = \frac{Dx + Ey + F}{Gx + Hy + I}$$

$$x' = \frac{x}{w} = \frac{au + bv + c}{gu + hv + i}$$

$$y' = \frac{y}{w} = \frac{du + ev + f}{gu + hv + i}$$

Horizontal and Vertical Polygons



$$x = \frac{au}{a-v}$$

$$y = \frac{1}{a-v}$$

Scan horizontally: 1 divide per scanline

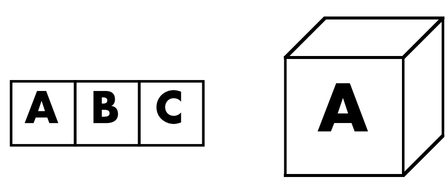
Affine Texture Map

```
x = xmin;
u = A*x+B*y+C;
v = D*x+E*y+F;
for( ; x<xmax; x++ ) {
  image[x][y]=texture(u,v);
  u += A;
  v += D;
}
```

$$u = Ax + By + C$$

$$v = Dx + Ey + F$$

Parallel Projections



Linearly Interpolating Texture Coords

Implicit Texture Transformation

The Basic Idea

1. Projective transformation between spaces
2. Create homogeneous coordinates at vertices
3. Interpolate homogeneous coordinates
4. Divide through by w to get true coordinates

$$\begin{tikzcd} \hat{Q} \arrow[r, "T^{-1}"] & P \arrow[d] \\ Q \arrow[u] & \hat{P} \arrow[l, "T"] \end{tikzcd}$$

$$P = T^{-1}\hat{Q} = T^{-1}\frac{Q}{Q_w} = T^{-1}T\frac{\hat{P}}{Q_w} = \frac{\hat{P}}{Q_w}$$

Generalized Interpolation

$$(x,y,z,1,nx,ny,nz,r,g,b,a,u,v,...,1)$$

Parameters affinely related $\downarrow$ **Transform and Clip**

$$(xw,yw,zw,w,nx,ny,nz,r,g,b,a,u,v,...,1)$$

$\downarrow$ **Project**

$$(xw/w,yw/w,zw/w,w,nx/w,ny/w,nz/w,r/w,g/w,b,a/w,u/w,v/w,....,1/w)$$

$\downarrow$ **Interpolate**

Normal - Phong
Color - Gouraud

...

Both already approximations $\downarrow$ **Project**

...

Explicit Texture Transformation

Pages from Paul

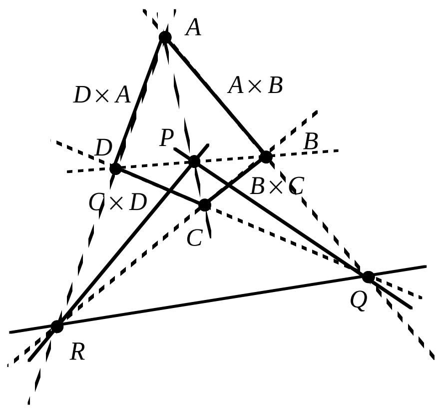
The idea

Quadrilateral-Quadrilateral Map

Quadrilateral-Square Map

Use for fake 3D!

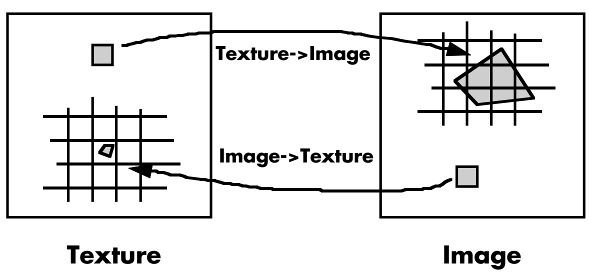
The Quadrangle



$$P = (B \times D) \times (A \times C)$$
$$Q = (A \times B) \times (C \times D)$$
$$R = (D \times A) \times (B \times C)$$

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} P & Q & R \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Filtering



Two cases:

1. **Magnification / Interpolation**
2. **Minification / Decimation / Averaging**

Resampling Process

Discrete Image

↓ Reconstruction filter

Continuous Image

↓ Transformation/Warp

Continuous Image

↓ Bandlimiting filter

Continuous Image

↓ Sample

Discrete Image

Resampling

Can merge reconstruction, transform, filter

If *magnify* **-> Bandlimiting subsumed by reconstruction**
-> Reconstruct in destination space with transformed filter

If *minify* **-> Reconstruction subsumed by bandlimiting**
-> Average in source space with transformed filter

Note:

1. **Affine: Shift invariant filter**
2. **Perspective/Deformation: Shift varying filter**

Fast Filtering Algorithms

Direct convolution

Time proportional to the area

Precompute partial convolutions

Approximate filtering

- **Resolution sets**
- **Mipmap "multum in parvo"**
  Dungan, Williams
- **Summed area tables**
  Crow
- **Repeated integration filtering**
  Perlin, Heckbert

Resolution Sets

CATS - Coherent Access Textures

Ideal for uniform sampling

of parametric surfacez

MipMaps

Interpolation

Principle of Texture Thrift

Using a multiresolution pyramidal representation, the amount of texture information needed by the renderer is proportional to the size of the output image, and not the size of the input texture database.

Summed Area Tables