Shading

Color Interpolation

- **Flat shading**

```
glBegin(GL_TRIANGLE);
  glColor3f(r,g,b);
  glVertex3f(x1,y1,z1);
  glVertex3f(x2,y2,z2);
  glVertex3f(x3,y3,z3);
glEnd();
```

- **Gouraud shading**

```
glBegin(GL_TRIANGLE);
  glColor3f(r1,g1,b1); glVertex3f(x1,y1,z1);
  glColor3f(r2,g2,b2); glVertex3f(x2,y2,z2);
  glColor3f(r3,g3,b3); glVertex3f(x3,y3,z3);
glEnd();
```


Copyright © Pat Hanrahan

Polygon Interpolation

This lecture:

- **Gouraud shading**
  - **Interpolate color**
- **Texture mapping**
  - **Interpolate texture coordinates**
  - **Lookup color at that position in a texture map/image**

Other possibilities:

- **Z-buffering**
  - **Interpolate z**
  - **Resolve hidden surfaces at each pixel**
- **Phong shading**
  - **Interpolate normal**
  - **Compute new color at each pixel**

Copyright © Pat Hanrahan

Texture Mapping

Texture Coordinates at vertices

```
glBegin(GL_TRIANGLE);
   glTexCoord2f(u1,v1); glVertex3f(x1,y1,z1);
   glTexCoord2f(u2,v2); glVertex3f(x2,y2,z2);
   glTexCoord2f(u3,v3); glVertex3f(x3,y3,z3);
glEnd();
```

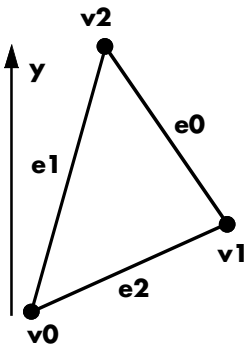
Other variations

- Directions, 3D
- Current texture transformations

Copyright © Pat Hanrahan

Recall: Triangle Tiler

```
scany( vert v[3]) {

  ybot = ceil(v[0].y);
  ymid = ceil(v[1].y);
  ytop = ceil(v[2].y);

  differencey(&v[0],&v[2],&e1,ybot); l=e1;
  differencey(&v[0],&v[1],&e2,ybot); r=e2;
  differencey(&v[1],&v[2],&e0,ymid);

  for( y=ybot; y<ytop; y++ ) {
      if( y >= ymid ) r = e0;
      scanx(&l,&r,y);
      l.x += l.dxdy; l.v += l.dvdx; ...
      r.x += r.dxdy; r.v += r.dvdx; ...
  }
}
```



Assume:
1. v's sorted
2. single l edge

Copyright © Pat Hanrahan

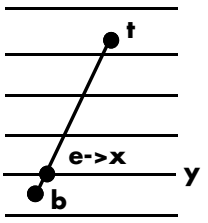
Horizontal Span

```
scanx( edge *l, edge *r, int y)
{
  lx = ceil(l->x);
  rx = ceil(r->x);

  if( lx < rx ) {

    differencex(l,r,&s,lx);

    for( x=lx; x<rx; x++ ) {
      fragment(x,y);
      s.v += s.dvdx; ...
    }
}
```

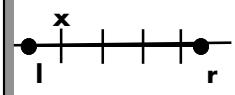

Copyright © Pat Hanrahan

Increments

```
differencey(vert *b, vert *t, edge *e, int y)
{
  value recdy = 1.0 / (t->y - b->y);

  e->dxdy = (t->x - b->x) * recdy;
  e->x    = b->x + (y - b->y) * e->dxdy;

  e->dvdy = (t->v - b->v) * recdy;
  e->v    = b->v + (v - b->v) * e->dvdy;
  ...
}
```

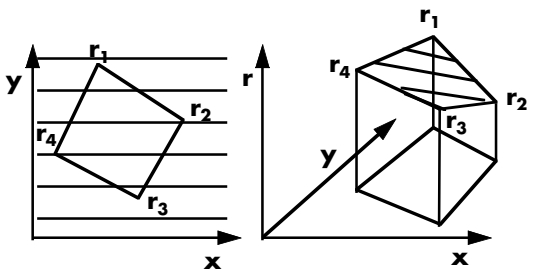



```
differencex(edge *l, edge *r, span *s, int x)
{
  value = recdx = 1.0 / (r->x - l->x);

  s->dvdx = (r->v - l->v) * recdx;
  s->v    = l->v + (x - l->x) * s->dvdx;
  ...
}
```



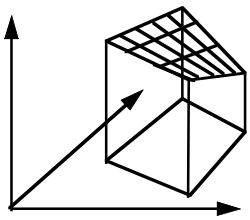
Copyright © Pat Hanrahan

Gouraud Shading



Gouraud Interpolation

Note: Not rotationally invariant

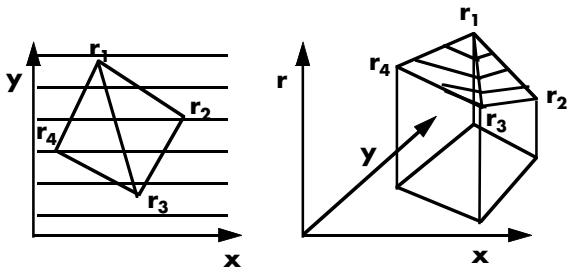


Ruled Surface:

Hyperboloid!

Copyright © Pat Hanrahan

Triangle Interpolation

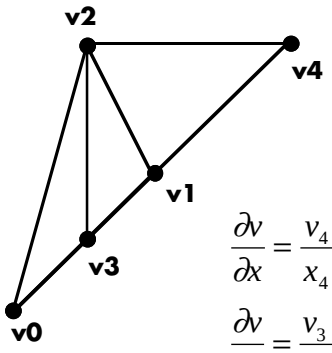


Gouraud Interpolation on Triangles

Note: Rotationally invariant

Copyright © Pat Hanrahan

Triangle Setup



$$\frac{\partial v}{\partial x} = \frac{v_4 - v_2}{x_4 - x_2} = \frac{(v_1 - v_0)(y_2 - y_0) - (v_2 - v_0)(y_1 - y_0)}{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)}$$

$$\frac{\partial v}{\partial y} = \frac{v_3 - v_0}{y_3 - y_0} = \frac{(v_2 - v_0)(x_1 - x_0) - (v_1 - v_0)(x_2 - x_0)}{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)}$$

Copyright © Pat Hanrahan

Texture Mapping <=> Image Warp

2D Texture Space

↓ **Parameterization**

3D Object Space

↓ **Model Transformation**

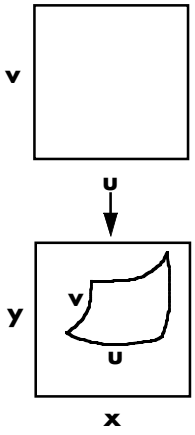
3D World Space

↓ **Viewing Transformation**

3D Camera Space

↓ **Projection**

2D Image Space



Copyright © Pat Hanrahan

Polygonal Texture Mapping

1. **Parameterization may be written as a linear transformation from texture space to object space**
2. **Modeling and viewing transformation from object space to camera space**
3. **Projection from camera space to screen space**

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underbrace{\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}}_{\text{TO}} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \underbrace{\begin{pmatrix} . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix}}_{\text{OC}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

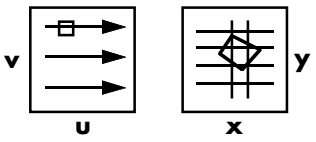
$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \underbrace{\begin{pmatrix} . & . & . \\ . & . & . \\ . & . & . \end{pmatrix}}_{\text{TS=TO•OC•CS}} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Copyright © Pat Hanrahan

Mapping Algorithms

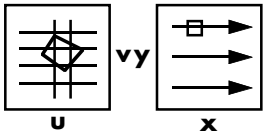
Forward mapping (video effects)

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



Backward mapping (workstations)

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



Copyright © Pat Hanrahan

Texture Mapping Polygons in Perspective

Forward transformation: linear projective map

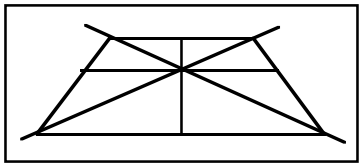
$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Backward transformation: linear projective map
(Inverse of a 3x3 matrix is a 3x3 matrix)

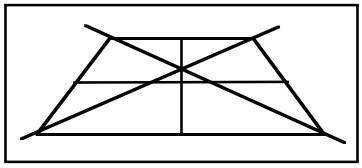
$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

Copyright © Pat Hanrahan

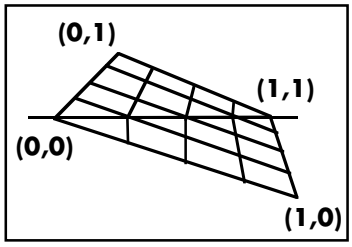
Linearly Interpolating Texture Coords



Correct Linear Perspective



Incorrect Perspective



Linear Interpolation, *Bad*
Perspective Interpolation, *Good*

Copyright © Pat Hanrahan

Implicit Texture Transformation

1. Linear interpolation: Map (x,y) with (u,v)

$$\begin{pmatrix} xw \\ yw \\ w \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \neq \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} xw/w \\ yw/w \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

2. Projective interpolation: Map (x,y) with (u/w,v/w,1/2)

$$\begin{pmatrix} xw \\ yw \\ w \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} xw/w \\ yw/w \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} u/w \\ v/w \\ 1/w \end{pmatrix}$$

$$\begin{pmatrix} u/w \\ v/w \\ 1/w \end{pmatrix} = \begin{pmatrix} uw \\ vw \\ w \end{pmatrix} = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Copyright © Pat Hanrahan

Generalized Interpolation

Vertex: *(x,y,z,1,u,v,1,...,r,g,b,a)*

Parameters affinely related

↓ **Transform and Clip Vertices**

(xw,yw,zw,w,u,v,1...,r,g,b,a)

↓ **Project Vertices**

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

↓ **Interpolate between vertices (difference and increment)**

"Hyperbolic Interpolation," J. Blinn

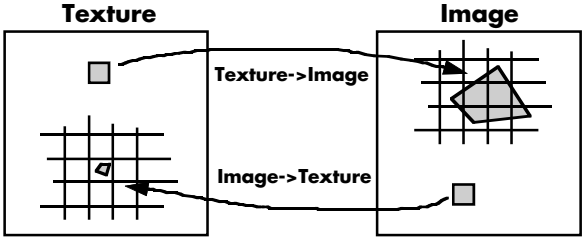
…

↓ **Project at Pixels (divide by [1/w])**

…

Copyright © Pat Hanrahan

Filtering Textures

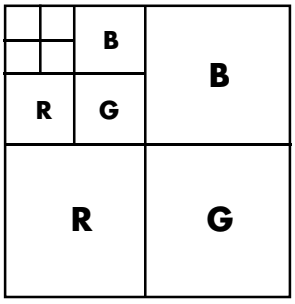


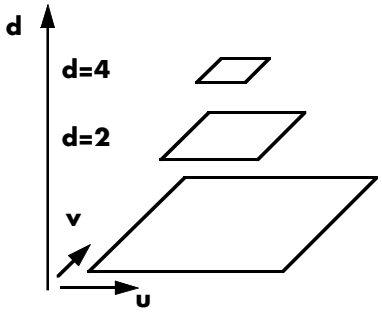
- Footprint changes from pixel to pixel: no single filter
- Resampling theory: two cases
  1. Magnification => Interpolation
  2. Minification => Averaging
- Constant cost per pixel

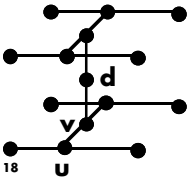
Copyright © Pat Hanrahan

MipMaps - L. Williams

Multum In Parvo = Many things in a small place







Trilinear mip-map interpolation

Copyright © Pat Hanrahan