ANAMORPHIC IMAGE PROCESSING

By

Steven Yelick

Submitted in Partial Fulfillment
of the Requirements for the
Degree of

Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1980

Signature of Author ______________________________
Department of Electrical Engineering
May 9, 1980

Certified by ______________________________
Andrew Lippman
Reaserch Associate
Thesis Supervisor

Accepted by ______________________________
Professor David Adler
Chairman, Undergraduate Thesis Committee

√

Table of Contents

Anamorphic Image Processing

By

Steven E. Yelick

submitted to the Department of Electrical Engineering and Computer Science on May 9, 1980 in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Abstract

Anamorphic processes are a class of visual transformations that distort or stretch an image. This thesis explores three such transformations which are projective in nature. In all three the target image is rendered in linear perspective, that is, space projected to a point through a viewing plane.

Thesis Supervisor: Andrew Lippman, Research Associate

Introduction

Throughout history man has attempted to represent the three dimensions of his environment on two dimensional surfaces. Surfaces have proven to be ideal for communication, reproduction, and as visual memory. Encoding the lost dimension presents the same problem to the artist, cartographer, draftsman, or photographer. How each resolves this problem is dependent on those qualities of three-space (such as dimensionality, color or linearity) that must be preserved. To the artist and photographer, the preservation of apparent spatial relationships is paramount.

During the Renaissance, linear perspective became the representation deemed most realistic by mathematicians. Debate raged as to whether art need follow the rules of perspective or not. As mathematicians systematized projective geometry, artists rebelled, saying that strict adherence to perspective was "anti-artistic." With the development of photography, the split between art and mathematics was complete. The camera provided an image production system that, unlike the brush and canvas, invariably created a perspective view. Photographs became the two-dimensional reality against which all representational art was compared.

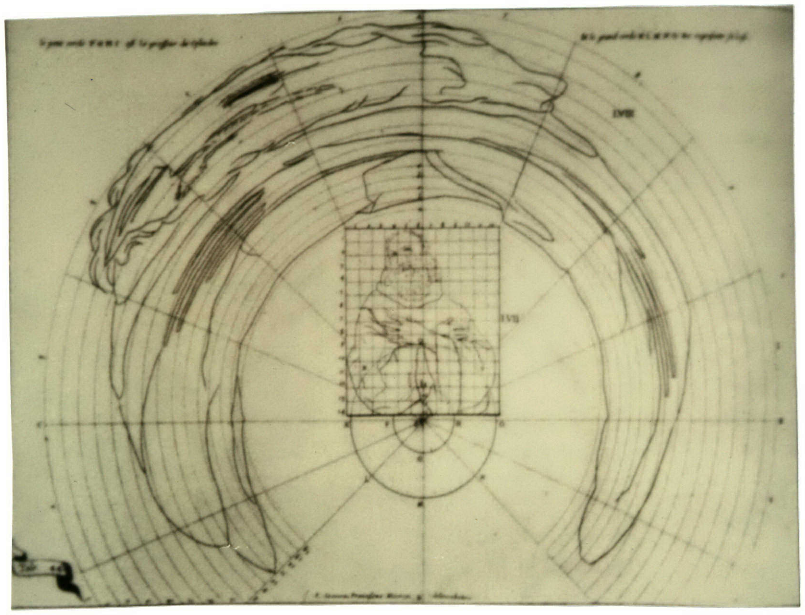


Figure 1

Integrally entwined with the development of perspective was that of anamorphic art. Paintings and engravings that, when viewed edgewise or with special mirrors, revealed new views became quite popular during the sixteenth and seventeenth centuries. Figure 1 shows an engraving by Jean-Francois Niceron that when reflected in a cylindrical mirror, appears like the normal view in the center.

During the late 1700s, a Scottish painter named Robert Barker invented an optical refinement of perspective that allowed a 360 degree field of view. The resulting views, called panoramas, were an instant success, and many feared that painting itself was threatened. The fear proved unfounded since panoramic painting soon lost its popularity.

Recently, there has been a new found interest in panoramic imaging, especially in the context of movie mapping. The development of the optical video disc provides the ability to randomly access a large number of video images. This access can allow a user to interactively sample a set of images that simulate travel in an unfamiliar space, i.e., the movie map. Panoramic viewing is particularly powerful because the spatial relationships of all objects visible from a single location can be portrayed.

Linear Perspective

Of all the pictorial qualities that are valued in visual representations, preservation of straight lines has been given the most attention. The three main projective geometries that preserve straight lines are axonometric, oblique, and perspective. In this section I will show that axonometric and oblique projections are merely special (albeit simplified) cases of perspective.

Oblique projections are limited in that a face of the object being projected must be parallel with the view plane. The lines of projection are parallel and intersect the view plane at an oblique angle, thereby providing a view of object depth. The face of the object that is parallel to the view plane is called the principal face, and oblique projection is useful for the class of objects whose principal face is easily definable. Figure 2

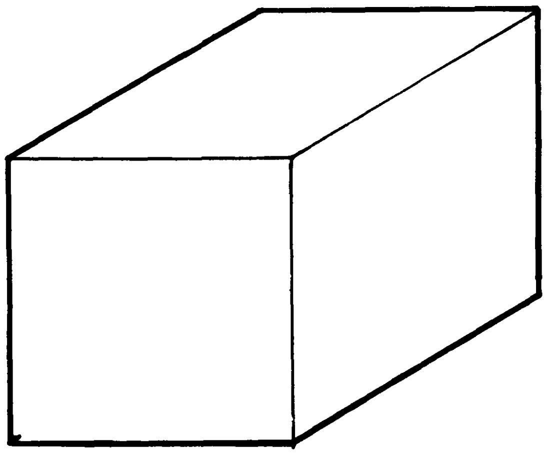
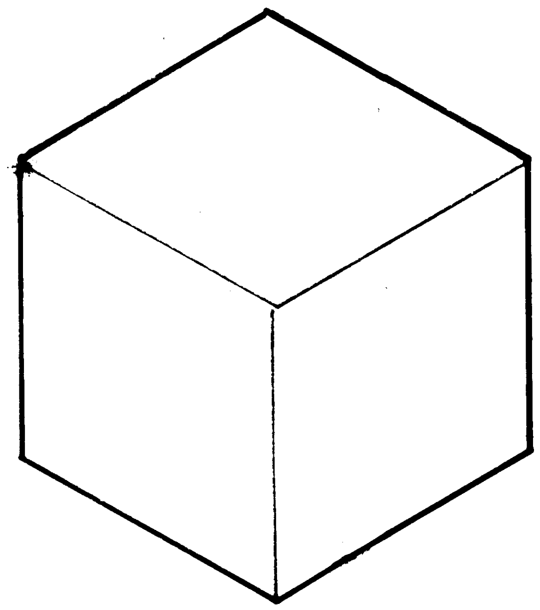


Figure 2

shows an oblique cube projection. Note that dimensionality and angularity are completely preserved in the vertical and horizontal, but only at the expense of considerable depth distortion.

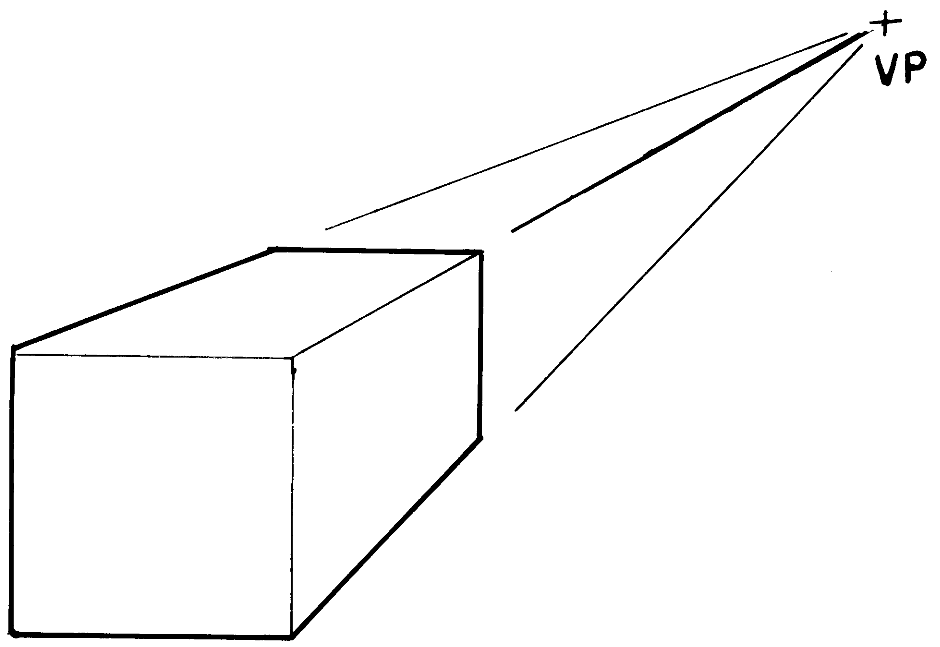
In contrast to oblique projection, an object in an axonometric projection can assume any angle to the view plane, but the parallel lines of projection intersect the view plane perpendicularly. Figure 3 shows an axonometric cube. Dimensionality and angularity are lost, but depth distortion is less apparent.

Figure 3



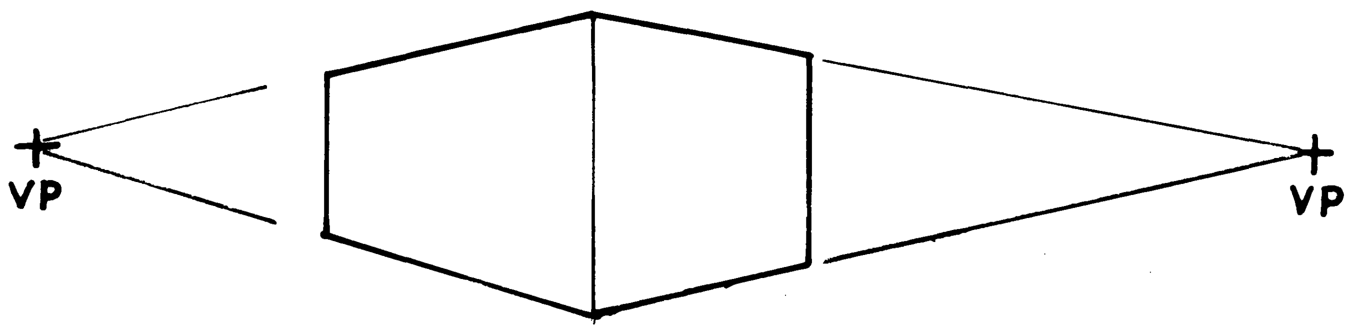
Oblique and axonometric views are created with parallel projection. Perspective, on the other hand, simulates what the eye sees by projecting to a single point through the view plane. Perspective views are classified by the number of points towards which parallel lines converge. Figure 4 is a one point perspective. Horizontal and vertical lines are parallel with the view plane, much like the oblique view, but depth lines look undistorted. We interpret perspective views so often that the parallel depth lines may actually appear to diverge in an oblique view.

Figure 4



Two point perspective allows an object to be viewed from any horizontal angle. As Figure 5 shows, only vertical lines remain parallel. Because both vanishing points are so close together, depth seems greatly exaggerated.

Figure 5



Three point perspective gives full freedom of viewing angle. All lines converge to their respective vanishing points, as Figure 6 shows. Because perspective

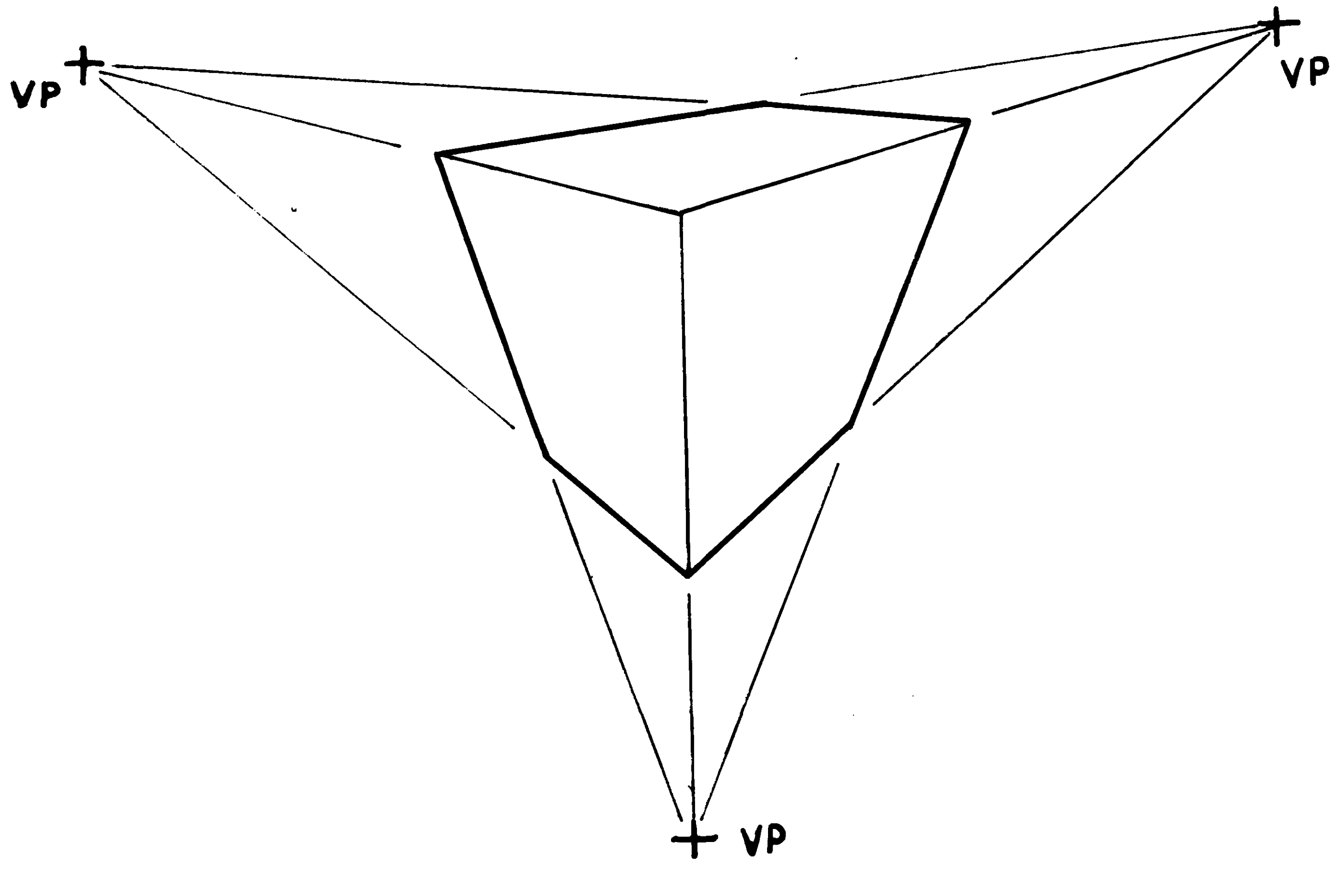


Figure 6

is taught as three different ways of drawing space, it is seldom pointed out that all three involve exactly the same transformation. The different perspectives arise merely from the placement of the view plane with respect to the object. One and two point perspectives can, in fact, be understood as three point perspective with one or two of the vanishing points infinitely distant from the view. Lines receding towards infinity will appear parallel. With this in mind, if we assume all three vanishing points to be located at infinity, the resulting view is axonometric. Oblique views are a less pure extrapolation, since the depth "vanishing" point, even though located at infinity, must move angularly with respect to the other two.

To equate perspective, axonometric projection, and oblique projection necessatates introducing the notion of vanishing points as virtual locations rather than physical intersections. This is not difficult to envision if one considers that the distance to the intersection of parallel lines is infinite.

As mentioned before, the apparent depth of a scene depends on the separation of vanishing points. That separation, in turn, is determined by the distance of the view point from the view plane. Decreasing the focal length decreases the separation of the vanishing points, providing a wider angle of view. Figure 7 compares the perspective model of of projecting through a plane with the optical reality of projection onto film. It is clear that focal length and any distance measured from the center of projection form a right triangle, so horizontal view angle is described by the equation:

$$\theta = 2\tan^{-1}\left(\frac{w}{2f} \right)$$

θ - view angle

w - view plane width

f - focal length

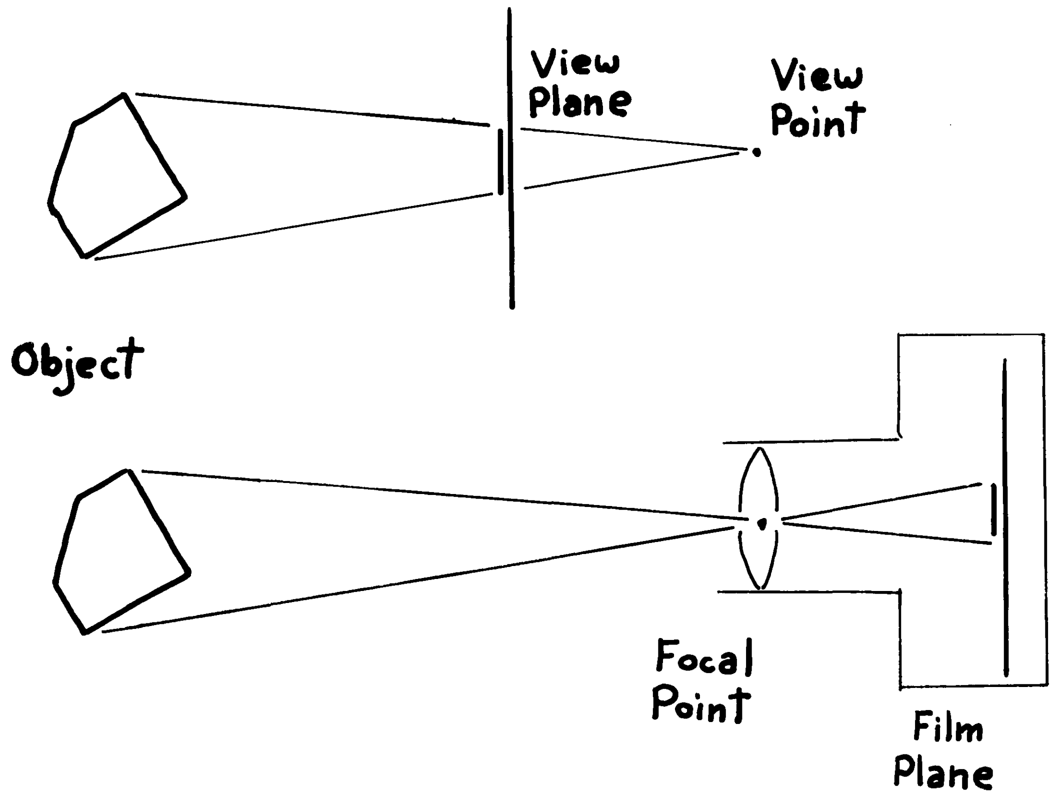


Figure 7

The images produced by perspective become quite distorted when the focal length is less than half the view plane width (lens view angle measurements are usually measured with respect to the diagonal rather than the horizontal). This corresponds to a viewing angle of approximately 90 degrees, which means that two vanishing points could both be visible in the image. As the view angle grows larger, edge distortion increases rapidly, and the need for a different projection becomes acute.

Curvilinear Perspective

Preservation of straight lines in views much greater than 90 degrees is difficult since edge distortion becomes quite apparent. The quality of straight lines remaining straight, on which so much emphasis was placed in normal perspective, falls victim to the desire for panoramic viewing.

Extreme wide angle views appear stretched and distorted at the edges because equal angles correspond to increasingly larger lengths as the distance from the view plane center increases. Cylindrical projection makes equal horizontal angles appear as equal distances by wrapping the view plane around the view point. Figure 8 is an engraving by

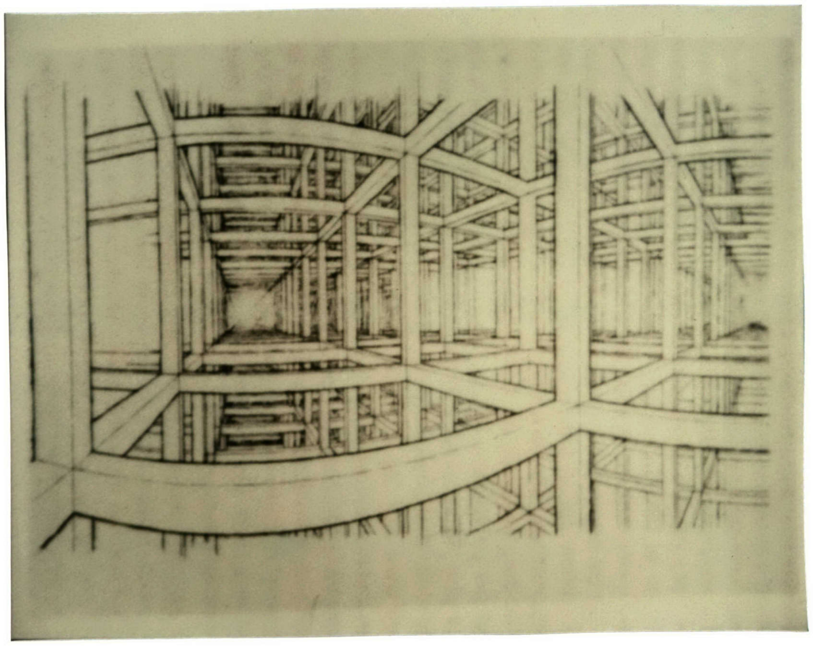


Figure 8

M.C. Escher of a rectilinear grid in cylindrical perspective. Two vanishing points are visible in the image, and because vertical lines are projected linearly, they appear straight. Only lines that are parallel with the axis of the view cylinder will be straight, so that if the grid had been angled up or down, all lines would be curved.

Cylindrical projection is ideal for panoramas, since the view cylinder can easily be cut and laid flat. The globuscope, a type of rotating slit camera, provides an imaging system that incrementally exposes vertical "slices" of space, thereby creating a cylindrical view. Because the exposure is incremental, the scene must remain stationary or horizontal blurring will occur.

In much the same way that three point perspective relates to two point perspective, spherical perspective relates to cylindrical perspective. Figure 9 is an

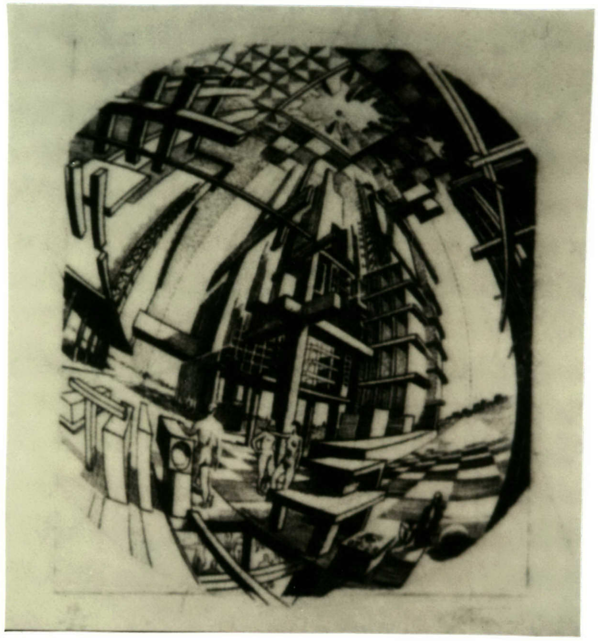


Figure 9

engraving by Albert Flocon in spherical projection. All three vanishing points are visible, and lines in all three dimensions are curved. Cylindrical projection required slicing the view cylinder and laying it flat. Similarly, an additional spatial transformation must be made to map the view sphere onto a flat surface. This mapping is exactly the problem that has faced cartographers for centuries.

If less than half of the view sphere is being shown, a parallel projection perpendicular to the view plane can be used. This mapping is known as orthographic, and the displacement form the center of the image is related to angle by the equation:

$$y = d \sin(\Phi)$$

y - distance from center

d - diameter of view

Φ - angle from perpendicular

Orthographic mapping of more than a hemisphere will result in two angles mapped onto the same position on the image. Equidistant projection averts this problem by mapping distance as a linear function:

$$y = d \cdot \frac{2\Phi}{v}$$

y - distance from center

d - diameter of view

Φ - angle from perpendicular

v - maximum view angle

This transformation is neither a parallel projection nor projection through the view plane to a point. Projection to any single point would result in a highly non-linear transformation. Only if the view point moves along the perpendicular as a function of angle can the transformation be visualized as purely projective.

Interpolative Views

Panoramic viewing is useful because it shows the spatial relationships of objects that are widely dispersed through a scene. If the entire panorama isn't available on one image (in cylindrical or spherical perspective), the ability to rotate view direction arbitrarily can be nearly as powerful. An apparent intermediate view can be simulated by projecting two angularly displaced views onto the new view plane, as shown in Figure 10. The two

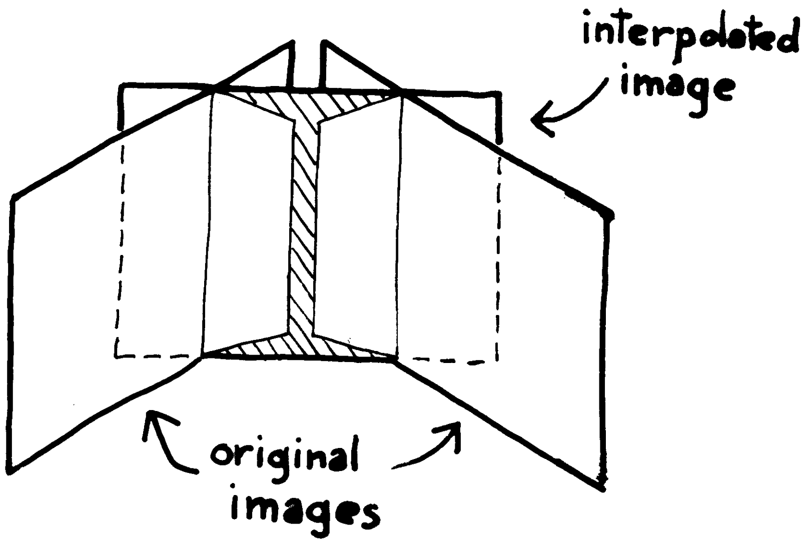


Figure 10

original views must be the correct focal length away from the view point and must retain their angular separation. The new view may be projected with any focal length, and can be rotated around the view point arbitrarily. Thus, from separate images recorded at the same location and at specific angles, a picture can be created for any angle of view.

Figures 11 and 14 are the original views from which Figures 12 and 13 are created. Figure 12 is 30 degrees rotated from Figure 11, Figure 13 is rotated 60 degrees. The originals came from 16 mm footage shot in Aspen, Colorado. Four orthogonal 16 mm pictures were taken simultaneously using a 5.5 mm lens. The horizontal angle of view is calculated to be 86 degrees using the angular view formula (the width of the image on 16 mm film is 10.2 mm). This explains the black mullion in the center of the interpolated views; neither original image covers those four degrees. This problem can be overcome either by increasing view angle for each original, or by decreasing angular displacement and using more views.

The software that projects the new view works in three steps for each point.

1. Calculate the equation of the line passing from the view point to the point on the new view plane.

2. Calculate the equation of the old view plane. The old view plane must be rotated around the view point by the view angle.

3. Solve for the intersection of the line in step

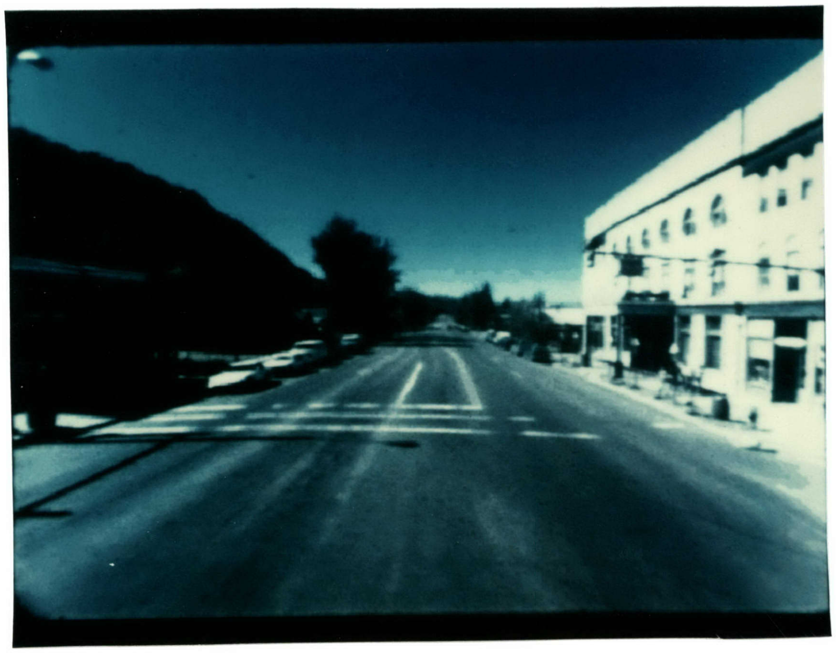


Figure 11

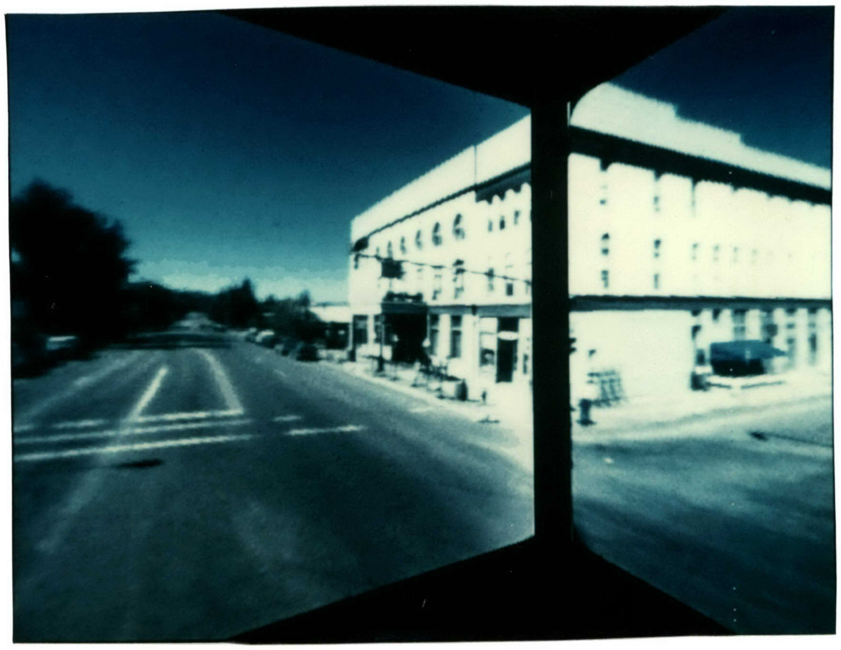


Figure 12

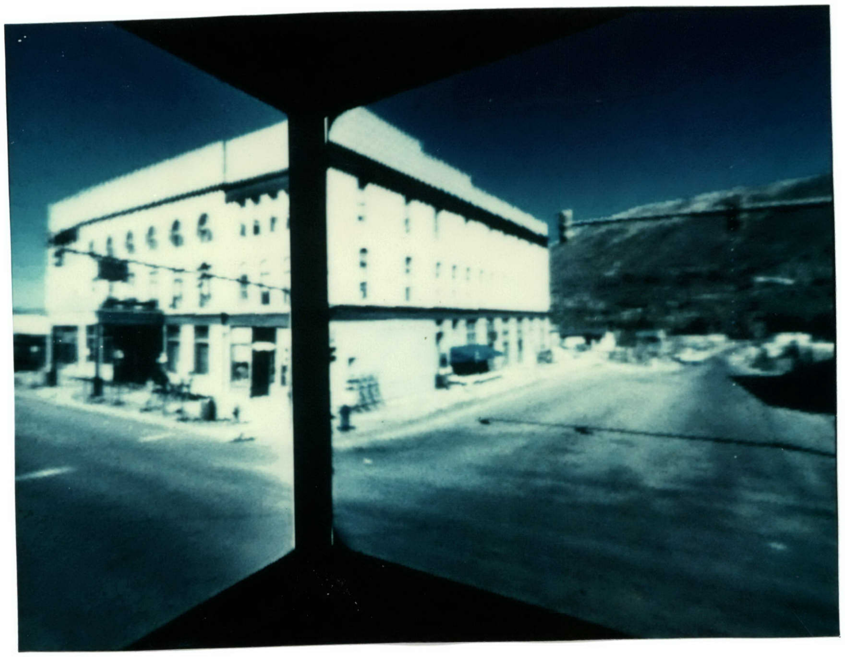


Figure 13

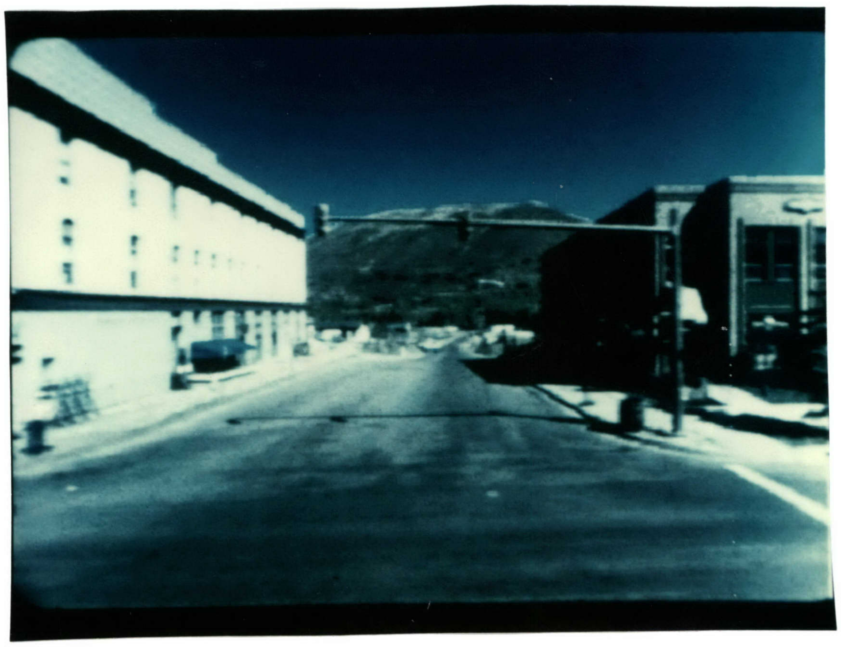


Figure 14

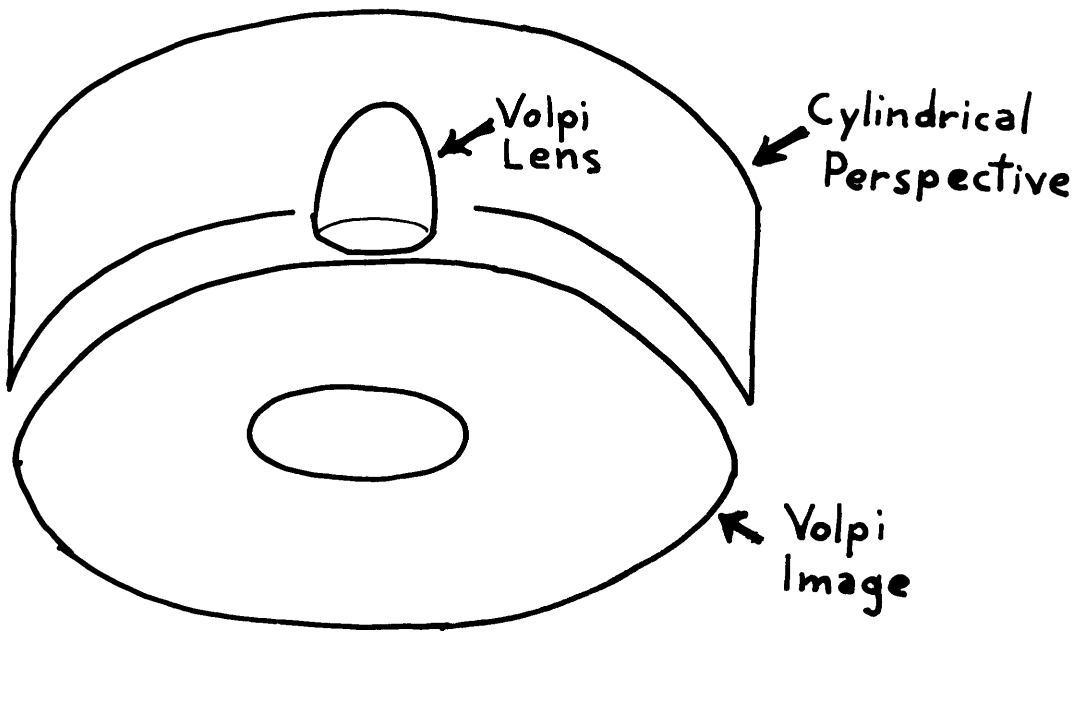
1 and the plane in step 2 to find the sample position in the old view.

An important feature of movie mapping is the ability to change travel direction under user control. Interpolative viewing was used to synthesize turn footage for those corners that were skipped during filming of Aspen. Ultimately, real time interpolation could provide a movie map user with complete view control.

The Volpi Lens

In order to represent an entire panorama in a single view, an anamorphic mapping must be used. A special panoramic lens manufacture by Volpi (Switzerland) performs such a mapping by transforming a cylindrical panorama into a ring (Figure 15). Figure 16 is a Volpi image taken in Boston, Massachusetts, behind the Prudential Building.

Figure 15



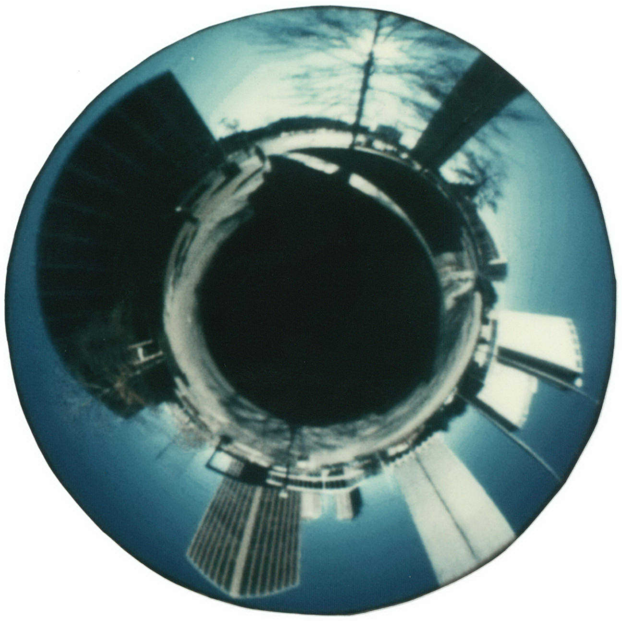


Figure 16

Because information about the exact transformation made by the Volpi lens was not available, photos were made of a cylindrical grid. The lens (Figure 17) is a parabloid of glass that uses both refraction and reflection

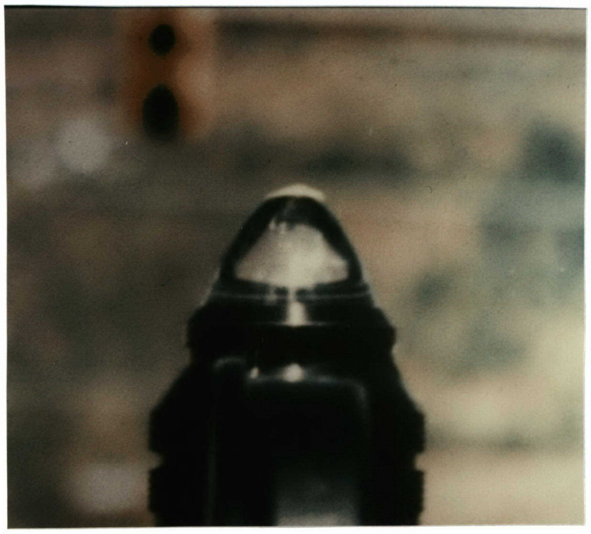


Figure 17

to project the image. It was thought that the lens provided a 60 degree vertical view, and the grid photos revealed that for an image with a 3 to 4 aspect ratio, the horizontal view angle should be 108 degrees.

By radially sampling a Volpi image, a cylindrical perspective is produced. Unfortunately, the vertical angular mapping is not linear. The grid photos were used to calculate the vertical distortion. Before measurements were made of the grid, a correction to turn the vertical measurements into angular measurements was needed. That correction was made by assuming that half the vertical distance subtends and angle of 30 degrees. The resulting conversion is:

$$\Theta = \sin^{-1}(d \cdot \sin(\tfrac{\pi}{6})/m)$$

Θ - angle from horizontal

d - distance from horizontal

m - maximum distance from horizontal

A plot was then made of radial distance versus angular displacement, as shown in Figure 18. That curve was approximated by the equation:

$$y = \frac{\sin(\pi x - \frac{\pi}{2}) + 1}{2}$$

y - corrected vertical distance

x - vertical distance

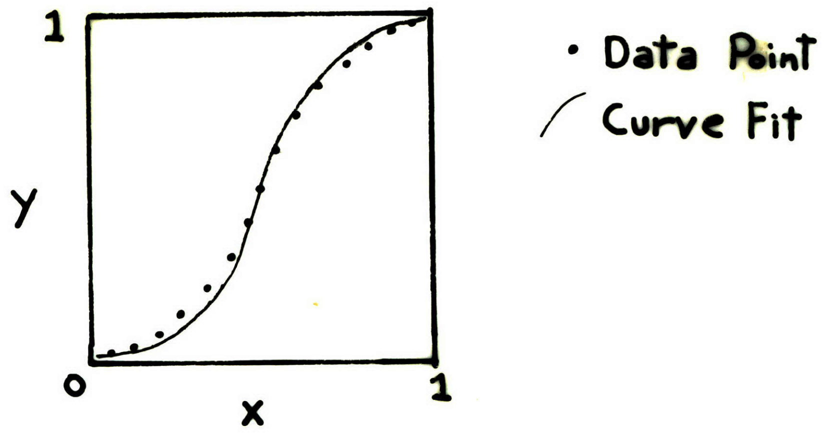


Figure 18

When this equation is applied to vertical sampling of the original photo, a corrected cylindrical perspective like Figure 19 is produced. Note that curvature is still present. To create a linear perspective, the view cylinder must be projected onto a view plane. That projection is

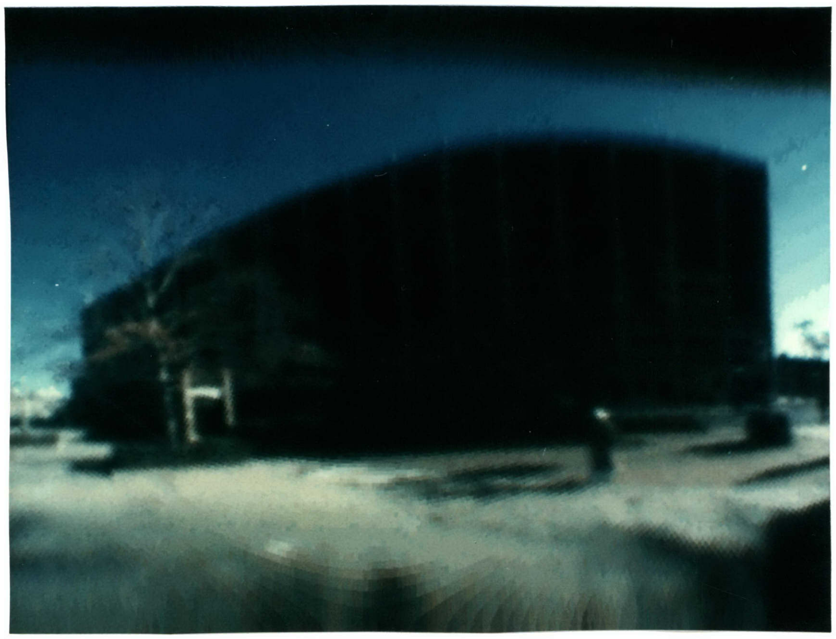


Figure 19

described by the equations:

$$xlin = f \cdot \tan\left(\frac{x}{f}\right)$$

$$ylin = y\left(\sqrt{\frac{x^2+f^2}{f}}\right)$$

xlin, ylin - linear perspective coordinates

x, y - cylindrical perspective coordinates

f - focal diameter

The focal diameter can be calculated from the horizontal view angle:

$$f = \frac{360v}{2\pi\Theta}$$

When these equationa are applied, views like Figures 20 and 21 result. The building in Figure 20 is no longer curved, and lines in Figure 21 recede towards their respective vanishing points.

Figure 20

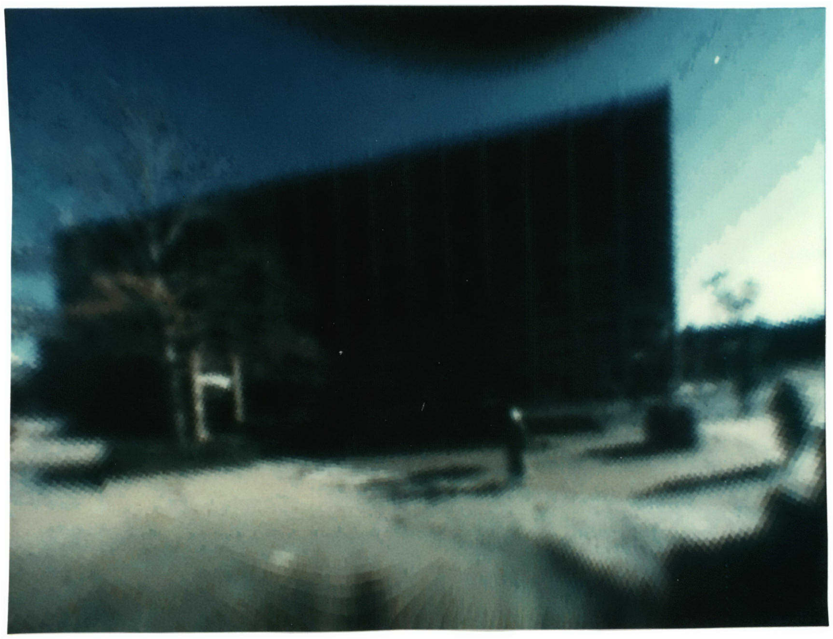
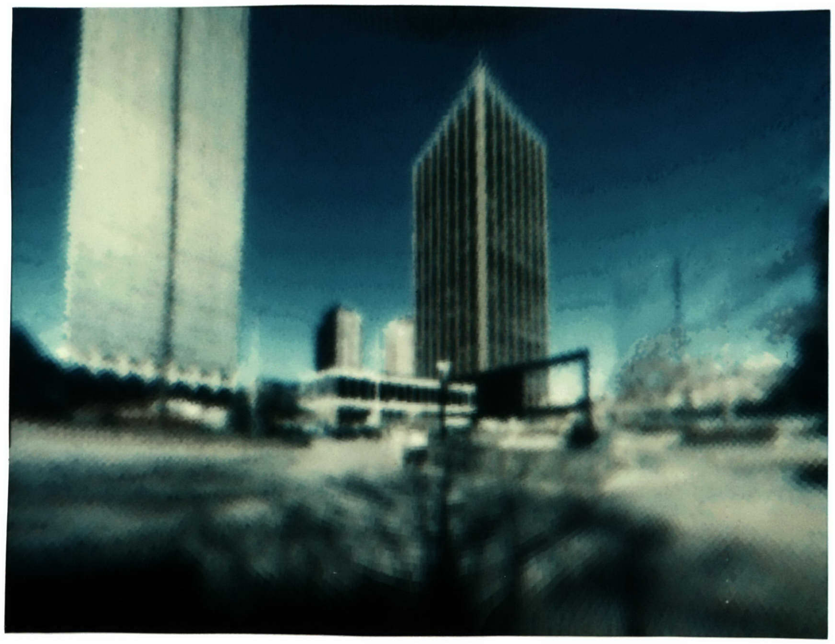


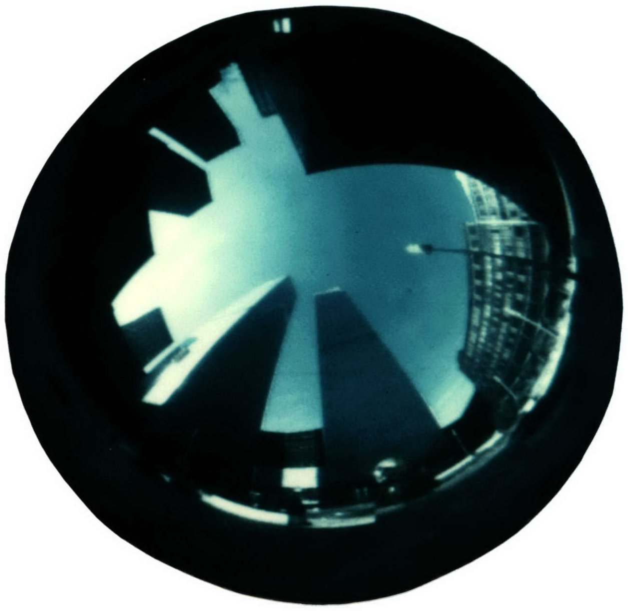
Figure 21



The Fish-eye Lens

While the Volpi lens produces a panoramic view for relatively flat geometries, vertical viewing is desirable for city panoramas. Figure 22 is a photo taken in New York City of the World Trade Center with a 6 mm fish-eye.

Figure 22



The 6mm lens provides a 220 degree view angle on 35 mm film, and uses an equidistant mapping. Due to perturbations in lens design, the actual angular mapping is given by:

$$y = (0.109)\Theta - (0.369 \times 10^{-4})\Theta^2 - (0.265 \times 10^{-7})\Theta^3$$

y - distance from image center in mm

Θ - angle from center

The transformation into linear perspective is more straightforward than for the Volpi lens since the mapping characteristics are known exactly. While approximately 10 degrees of view are lost near the ground, 120 degrees of view are gained across the top of the lens.

Once the center of the new view is selected, the position of the view plane can be calculated using the equidistant mapping formula. Each point in the view plane is translated into spherical coordinates Θ and π. The coordinates to sample the image are given by the equations:

$$x = \frac{\Phi\cos(\Theta)}{110\text{rad}}$$

$$y = \frac{\Phi\sin(\Theta)}{110\text{rad}}$$

x, y - sample coordinates

Φ, π - spherical coordinates

rad - image radius

Figures 23 and 24 are linear projections of Figure 22. Notice that the subject of view can be arbitrary. Both of these views use a focal length that approximates a 15 mm lens. Figure 25 is a view with a focal length of 8mm. The edge distortion is extreme, note specifically the building on the right.

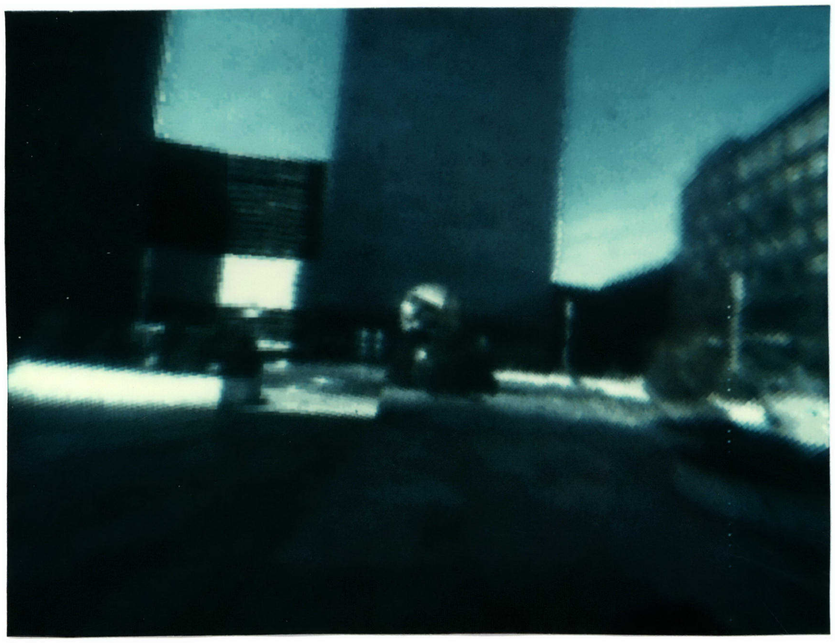


Figure 23

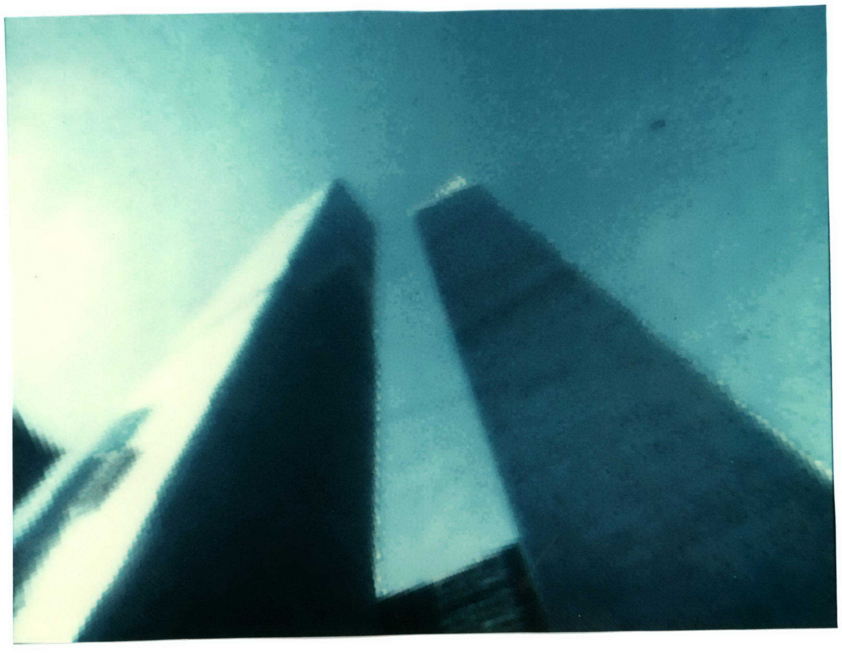


Figure 24

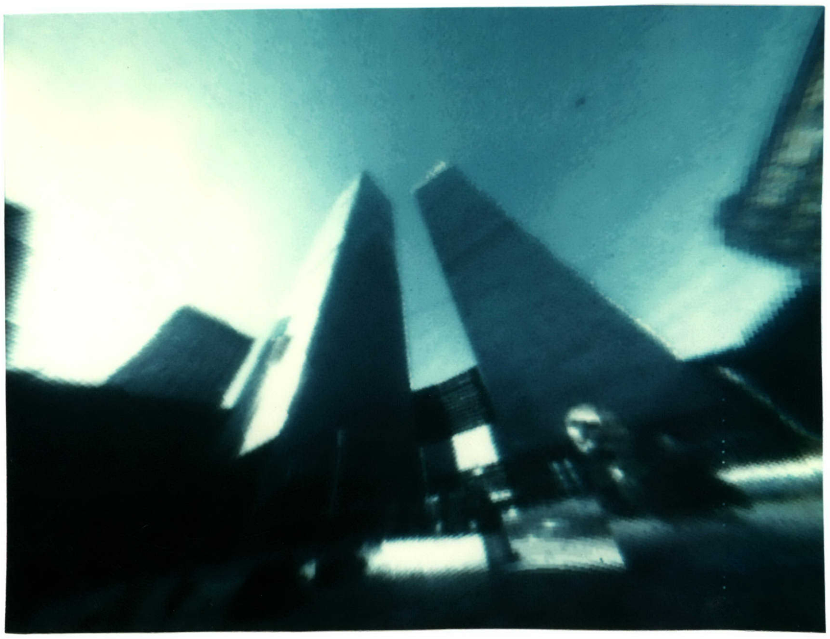


Figure 25

Conclusion

The major drawback of anamorphic processing in the video domain is resolution. The problem is especially acute with Volpi images, where the important information is radially compressed since it is in the center of the picture, and also vertically compressed. Panoramic viewing is really most useful when the entire panorama is visible. In that case spatial correspondence would overwhelm the viewer to such an extent that resolution problems would seem inconsequential.

Panoramas offer compact storage of visual relationships that separate images can only approach. Separate images, however, give increased resolution. One could concievably create panoramic data structures from multiple views that, when accessed, could provide normal or even enhanced resolution.

While anamorphoses has been understood for centuries, anamorphic imagery in photography is relatively unexplored. No longer is linear perspective looked upon as visual reality. Rather, it is just one of the ways that space can be represented on a surface.

Bibliography

1. Descargues, Pierre. Perspective, Harry N. Abrams, Inc., New York, 1977.

2. Giachino, J.W., and Beukema, Henry J. Engineering-Technical Drafting and Graphics, American Technical Society, Chicago, Illinois, 1961.

3. Karp, Jeffrey M. Letter to Hans-Christian Lischewski, Nikon Incorporated, Garden City, New York, December 27, 1979.

4. Newman, William M. Principles of Interactive Computer Graphics, McGraw-Hill Book Company, New York, 1979.

5. Nikon, Specification Sheet for the Nikkor 50mm f/1.8 Lens, Japan, 1979.

6. Richardson, Peter, and Adler, R.K. Map Projections: For Geodesists, Cartographers, and Geographers, American Elsevier Publishing Company, New York, 1972.

Appendix

1. All photos in this thesis are polaroid hardcopies of digital images stored on the Architecture Machine.

2. Program Listings:

   Twirl - image interpolation

   Turnwrap - cylindrical perspective-Volpi unwrapping

   Secwrap - linear perspective-Volpi unwrapping

   Fishy - Fish-eye straightening

```
twirl: proc;

/*
Interpolates a view between two adjacent frames.
For use with aspen footage.
Steve Yelick Feb 26, 1980
*/

dcl 1 split,
      2 top fix(7) init(0),
      2 bot fix(7);
dcl hw fix defined split;
dcl (i,j,flag,h,ox,oy,dx,dy,num,cruel) fix;
dcl (addif,dif,inc,here,there,step) fix;
dcl O fix(31);
dcl (lex,rex,tey,bey)[0:3] flt;
dcl (lext,rext,teyt,beyt) flt;
dcl (le1x,le1y,re1x,re1y,te1x,te1y,be1x,be1y) flt;
dcl (le2x,le2y,re2x,re2y,te2x,te2y,be2x,be2y) flt;
dcl (incy,samy1,samy2) flt;
dcl (lrot,rrot,f,vangle,h11y) flt;
dcl (lcos,lsin,rcos,rsin,px,pz,nx,nz,dis,d,t,tx,ty,tz,temx,temz) flt;
dcl pa fix;
dcl (vf,vfsqrd,vfactor,fsqrd,xsqrd,picangle,samx,samy) flt;
dcl (xorg,x,y,rad) flt;
dcl start fix;
dcl (vert,numa,offa)[0:479] fix;
dcl big[0:0] fix(7) based;      /* really 0:30719 */
dcl p[0:9] ptr;
dcl dn char(168)vary;
dcl (string,strsav,str1,str,name) char(32)vary;
dcl dot[0:3] char(1);
dcl (rightview,view) fix;
dcl (pat,mat)[0:255] bit(24);
dcl ch[0:4] char(1);
dcl (systemao,break) cond;

dcl camera$attach entry;
dcl camera entry (fix);
dcl cursor entry(fix,fix);
dcl (ram$refcm,ram$defcm) entry(bit(24),fixed,fixed);
dcl tab$big entry (fix,fix,fix,bit(8));
dcl (ram$write_block) entry (fix,fix,fix,fix,fix);
dcl (math$atan,math$sqrt,math$cos,math$sin) entry (flt) rtns (flt);
dcl hcs$initiate entry (char(168)vary,char(32)vary,ptr,fix(31));
dcl hcs$terminate entry (ptr);
dcl (com_error,askn,ioa,ioan) entry options (variable);
dcl scs$cl entry options(variable);

    ch[0]="0";
    ch[1]="1";
    ch[2]="2";
    ch[3]="3";
    ch[4]="4";
    call camera$attach;
    on systemao begin; call ioan("SYSTEMAO"); end;
    do oy = 0 to 479;
        num = oy/48;
        numa[oy] = num;
        offa[oy] = (oy-num*48)*640;
```

```
     end;
     call askn ("What is the view? (must be 8 bit) ",string);
dot[0]="e";
dot[1]="s";
dot[2]="w";
dot[3]="n";
do view=0 to 3;
     call scs$cl("loadpic "||string||"."||dot[view]||" -nz");
     call ioa("Tap left edge");
     call cursor(i,j);
     lex[view]=i;
     call ioa("Tap right edge");
     call cursor(i,j);
     rex[view]=i;
     call ioa("Tap top edge");
     call cursor(i,j);
     tey[view]=j;
     call ioa("Tap bottom edge");
     call cursor(i,j);
     bey[view]=j;
end;

do view=0 to 3;
     rightview=view+1;
     if rightview=4 then rightview=0;
     le1x=lex[view]; le2x=lex[rightview];
     re1x=rex[view]; re2x=rex[rightview];
     te1y=tey[view]; te2y=tey[rightview];
     be1y=bey[view]; be2y=bey[rightview];
     str=string||"."||dot[view];
     str1=string||"."||dot[rightview];
     do pa=0 to 80 by 10;
     vangle=pa;
     picangle=85;
     vfactor=1;

     hl1y=(be1y-te1y)/2.+te1y;

     rad=(picangle/2.)/360.*2*3.14159;

     vf=(re1x-le1x)/2.;
     vf=vf/(math$sin(rad)/math$cos(rad));

     f=vfactor*320./(math$sin(rad)/math$cos(rad));
     picangle=rad*2.;
     vangle=(vangle/360.)*2.*3.14159;
     start=0;
     flag=0;
     inc=0;

     call scs$cl("loadcm >pix>"||str1||">"||str1);
     call ram$refcm(mat,0,255);
     call scs$cl("loadcm >pix>"||str||">"||str);
     call ram$refcm(pat,0,255);
     call ram$defcm(mat[255],511,511);
     call ram$defcm(pat,0,255);
     call ram$defcm(mat,256,510);

     lrot=vangle;
     rrot=vangle-(3.14159/2.);
```

```
      i=rrot/(2.*3.14159)*360.;
      call ioa("Pictures are ^i degrees apart.",i);
      lcos=math$cos(lrot); lsin=math$sin(lrot);
      rcos=math$cos(rrot); rsin=math$sin(rrot);
      fsqrd=f*f;
      vfsqrd=vf*vf;
      lext=le1x; rext=re1x; teyt=te1y; beyt=be1y;
      px=vf*(-lsin);
      pz=vf*(lcos);
      nx=-lsin;
      nz=lcos;
      strsav=str;
call ioa("down to out");
out: if flag^=0 then do;
            vf=(re2x-le2x)/2.;
            vf=vf/(math$sin(rad)/math$cos(rad));
            lext=le2x; rext=re2x; teyt=te2y; beyt=be2y;
            px=vf*(-rsin);
            pz=vf*rcos;
            nx=-rsin;
            nz=rcos;

            lcos=rcos;
            lsin=rsin;
            strsav=str;
            str=str1;

            h1ly=(be2y-te2y)/2.+te2y;
            inc=256;

            do cruel=0 to 9;
                  call hcs$terminate (p[cruel]);
            end;
      end;

      dn = ">pix>"||str;
      do cruel = 0 to 9 by 2;
call ioa("cruel fucking loop ^i",cruel);
            name = str||".pic"||ch[cruel/2];
            call hcs$initiate (dn,name,p[cruel],0);
call ioa("the initiate actually occured");
            if 0<0 then do;
                  call com_error (0,"twirl",name);
                  return;
            end;
call ioa("Lets play pointers!!");
            p[cruel+1] = addrel(p[cruel],30720);
            call ioa("Still there? Here ew go again");
      end;

      call ioa("done with pic file intialization.");

      on break begin; goto done; end;

      d=px*nx+pz*nz;

      do i = start to 639;

            x=i-320.;
```

```
          xsqrd=x*x;
          dis=math$sqrt(240.*240.+xsqrd+fsqrd);
          tx=x/dis;
          ty=240./dis;
          tz=f/dis;
          t=d/(nx*tx+nz*tz);
          temx=tx*t;
          temz=tz*t;
          samx=temx*lcos+temz*lsin+320. ;

          if flag=0 then if samx>relx then do
               start=i;
               flag=1;
               goto out;
          end;

          samy1=-ty*t+h11y;
          samy2=ty*t+h11y;
          incy=(samy2-samy1)/480.;
          samy=samy1;

          do j = 0 to 479;

               samy=samy+incy;
               ox=samx; oy=samy;

               if ox=dx then if oy=dy then do; /* same spot as last time? */
                    vert[j] = hw+inc;
                    goto skip_the_shit;
               end;
               hw = 0;
               if oy>=teyt then if oy<=beyt then if ox>=lext then if ox<=rext then do;
                    num = numa[oy];        /* remember: (-23)/48=0 */
                    bot = p[num]->big[offa[oy]+ox];
                    dx = ox;
                    dy = oy;
                    vert[j]=hw+inc;
                    goto skip_the_shit;
               end;
               vert[j] = hw;

skip_the_shit:

          end;

           call ram$write_block (i,0,i,479,vert);

     end;

done:

     do cruel = 0 to 9 by 2;
          call hcs$terminate (p[cruel]);
     end;
str=strsav;
/*go to loop;
*/
call camera(11);
```

```
     end;
end;
getpt: proc (x,y);
dcl (x,y,z) fix;
dcl stat bit(8);
     do z=0 while (z=0);
          call tab$big (x,y,z,stat);
     end;
     do while (z<0);
          call tab$big (x,y,z,stat);
     end;
     call ioan ("\g");
end getpt;

end twirl;
```

```
turnwrap: proc;

/*
     Volpi unwrapping program for synthesized turn filming.

          by Steve Yelick 22 Feb 80
*/

dcl 1 split,
      2 top fix(7) init(0),
      2 bot fix(7);
dcl hw fix defined split;
dcl (ax,z,line,h,x,y,ox,oy,cx,cy,tx,ty,px,py,num,cruel) fix;
dcl (here,there,step) fix;
dcl O fix(31);
dcl (savx,savy) fix;
dcl ang fix;
dcl (U,F,T,X1,X2,Y1,Y2,X,Y,DX,DY,R1,R2,R3,R4,ANGLE,DANGLE) flt;
dcl (A,SINA,COSA,SINT,COST) flt;
dcl (numa,offa)[0:479] fix;
dcl vert[0:479,0:9] fix;
dcl big[0:0] fix(7) based;     /* really 0:30719 */
dcl p[0:9] ptr;
dcl dn char(168)vary;
dcl (str,name) char(32)vary;
dcl up char(1);
dcl (break,systemao) cond;
dcl (correct,correct2,correct3) flt;
dcl D[0:479] flt;
dcl (i,j) fix;
dcl xrad[1:6] flt;
dcl (B,EE,FY,FAX,E) flt;
dcl yrad[1:6] flt;
dcl (sx,sy)fix;

dcl camera$attach entry;
dcl camera entry(fix);
dcl char[0:4] char(1);
dcl cursor entry(fix,fix);
dcl ram$defcm entry(bit(24),fix,fix);
dcl (ram$rect,ram$write_block) entry (fix,fix,fix,fix,fix);
dcl (math$atan,math$sqrt,math$cos,math$sin) entry (flt) rtns (flt);
dcl hcs$initiate entry (char(168)vary,char(32)vary,ptr,fix(31));
dcl hcs$terminate entry (ptr);
dcl (com_error,askn,ioa,ioan) entry options (variable);
dcl scs$cl entry options(variable);

     char[0]="0";
     char[1]="1";
     char[2]="2";
     char[3]="3";
     char[4]="4";
     call camera$attach;          /* attach the Matrix */
     on systemao begin; call ioan ("SYSTEMAO "); end;
     do oy = 0 to 479;            /* initialize lookup tables */
          num = oy/48;
          numa[oy] = num;
          offa[oy] = (oy-num*48)*640;
     end;
     call askn ("which picture to unwrap? (must be 8 bit) ",str);
     dn = ">pix>"||str;
     do cruel = 0 to 9 by 2;     /* initiate picture segments */
          name = str||".pic"||char[cruel/2];
          call hcs$initiate (dn,name,p[cruel],O);
          if O<0 then do;
               call com_error (O,"turnwrap",name);
               return;
          end;
          p[cruel+1] = addrel(p[cruel],30720);
     end;
     call askn ("was the camera pointing up or down? (u or d) ",up);


     do i=1 to 3;                     /* three points on inner radius */
          call ioa("Tap Inner Radius");
          call cursor(tx,ty);
          xrad[i]=tx;yrad[i]=ty;
          call ioa("x=^f,y=^f",xrad[i],yrad[i]);
     end;
     do i=4 to 6;                     /* three points on outer radius */
          call ioa("Tap Outer Radius");
          call cursor(tx,ty);
          xrad[i]=tx;yrad[i]=ty;
/*        call ioa("x=^f,y=^f",xrad[i],yrad[i]);*/
     end;
     call ioa("Tap left edge ");      /* point to start unwrapping */
     call cursor(sx,sy);

                                      /* calculate the center */
     call center(xrad[1],yrad[1],xrad[2],yrad[2],xrad[3],yrad[3],DX,DY);
     tx=DX;ty=DY;
     call ioa("center1 x=^i y=^i",tx,ty);
     call center(xrad[4],yrad[4],xrad[5],yrad[5],xrad[6],yrad[6],DX,DY);
     cx=DX;cy=DY;
     call ioa("center2 x=^i,y=^i",cx,cy);
     cx=(cx+tx)/2;
     cy=(cy+ty)/2;

     call ram$defcm("ffffff"b4,500,500);/* mark center of image */
     call ram$rect(cx,cy-5,cx,cy+5,500);
     call ram$rect(cx-5,cy,cx+5,cy,500);
     DX=xrad[2]-cx; DY=yrad[2]-cy;
     R1=math$sqrt(DX*DX+DY*DY);
     DX=xrad[5]-cx; DY=yrad[5]-cy;
     R2=math$sqrt(DX*DX+DY*DY);


     h = 480;                         /* hieght in pixels */
     if up="u" then do; here = 0; there = h-1; step = 1; end;
     else do; here = h-1; there = 0; step = -1; end;
     T = (108.84/180.)*3.1415926/640.;/* horizontal angle per pixel */
     COST=math$cos(T);
     savx=sx;
     savy=sy;
     SINT=math$sin(T);
do ang=0 to 36;                       /* take a picture every 10 degrees */
     call ioa("View #^i",ang);
     X1=savx-cx; Y1=savy-cy;          /* set up sample radius */
     R3=math$sqrt(X1*X1+Y1*Y1);
     X1=X1*(R2/R3);
     Y1=-Y1*(R2/R3);


     px = -1;
     py = -1;
     on break begin; goto done; end;


     do x = 0 to 639;                /* horizontal pixel loop */

          line=x-((x/10)*10);
/*        call ioan ("^3i ",x);*/
/*        call ram$cursor (1,0,1,cx+X1,cy+Y1);*/
          X = X1;
          Y = Y1;
          if x<6400/108 then do;     /* linear unwrapping */
               savy=cy-Y;
               savx=cx+X;
          end;
          do y = here to there by step;/* vertical pixel loop */
               ox = cx+X;
               oy = cy-Y;

               if ox=px then if oy=py then do; /* same spot as last time? */
                    vert[y,line] = hw;
                    goto skip_the_shit;
               end;
               hw = 0;
               if oy>=0 then if oy<=479 then if ox>=0 then if ox<=639 then do;
                    num = numa[oy];       /* remember: (-23)/48=0 */
                    bot = p[num]->big[offa[oy]+ox];
                    px = ox;
                    py = oy;
               end;
               vert[y,line] = hw;
skip_the_shit:


               if x=0 then do;    /* initialize vertical distartion tables */
                    correct=1-y/(h-1.0);

/*                  D[y]=-0.012+0.529*correct-1.048*correct*correct+*/
/*                       4.486*correct*correct*correct+*/
/*                       1.385*correct*correct*correct*correct-*/
/*                       8.957*correct*correct*correct*correct*correct+*/
/*                  4.599*correct*correct*correct*correct*correct*correct;*/

                    if y>239 then
                         D[y]=(math$sin(3.14159*correct-3.14159/2.)+1.)/2.;
                         else D[y]=correct;

                    X=X1-(X1*(1-R1/R2))*(1-D[y]);
                    Y=Y1-(Y1*(1-R1/R2))*(1-D[y]);
               end;
               else do;
                    X=X1-(X1*(1-R1/R2))*(1-D[y]);/* calculate next sample */
                    Y=Y1-(Y1*(1-R1/R2))*(1-D[y]);
               end;
          end;

          X2 = (COST*X1+SINT*Y1);       /* next sample radius */
          Y1 = (COST*Y1-SINT*X1);
          X1 = X2;

          if line=9 then
           call ram$write_block (x-9,0,x,h-1,vert);/* write to screen */

     end;
call camera(11);        /* take a picture */
end;

done:
/*   call ram$cursor (0,0,0,0,0);*/
/*   call ram$cursor (1,0,0,0,0);*/
     do cruel = 0 to 9 by 2;
          call hcs$terminate (p[cruel]);
     end;



perpend:proc(x1,y1,x2,y2,xm,ym,slope);

/* subroutine to calculate the equation
of a line perpendicular to two points*/

dcl (x1,x2,y1,y2,xm,ym,slope)flt;


slope=-(x2-x1)/(y2-y1);
xm=(x2+x1)/2.;
ym=(y2+y1)/2.;

end perpend;


center: proc(x1,y1,x2,y2,x3,y3,xcen,ycen);

/* subroutine to calculate the center of a circle defined by 3 points */
dcl(x1,x2,x3,y1,y2,y3,xcen,ycen)flt;
dcl a[20,21] flt;
dcl (xa,ya,slopa) flt;
dcl (xd,yd) flt;
dcl (i,j) fix;



do i=1 to 2;
     if i=1 then do;
          xd=x1;
          yd=y1;
     end;
     else do;
          xd=x3;
          yd=y3;
     end;
     call perpend(xd,yd,x2,y2,xa,ya,slopa);
     a[i,1]=-slopa;
     a[i,2]=1;
     a[i,3]=ya-slopa*xa;
end;


call solve(2,a);
xcen=a[1,3];
ycen=a[2,3];

end center;


solve: proc(order,array);

/* subroutine to solve an n order array */

dcl array[20,21] flt;
dcl (order,m,n,l1,l2,r1,l3,x,y) fix;
dcl r2 flt;
m=order; n=order+1;

do x=1 to m;
     r1 = m;
     do l1=m-1 to x by -1;
          if (array[r1,x]*array[r1,x]) < (array[l1,x]*array[l1,x]) then r1=l1;
     end;
     do l2= n to x by -1;
          r2= fna(fna(array[r1,l2])/fna(array[r1,x]));
          array[r1,l2]= array[x,l2];
          array[x,l2]= r2;
     end;
     array[x,x]=1;
     do y = 1 to m;
          if x~=y then do;
               do l3= x+1 to n;
                    array[y,l3]= fna(array[y,l3])-fna(fna(array[x,l3])*
                         fna(array[y,x]));
               end;
               array[y,x]=0;
          end;
     end;
end;
end solve;

fna: proc(x) returns(flt);
dcl x flt; dcl y fix;


return(x);
end fna;


end turnwrap;
```

```
secwrap: proc;

/*
Linear perspective Volpi unwrapping program

     by Steve Yelick
*/

dcl 1 split,
      2 top fix(7) init(0),
      2 bot fix(7);
dcl hw fix defined split;
dcl (ax,z,line,h,x,y,ox,oy,cx,cy,tx,ty,px,py,num,cruel) fix;
dcl (here,there,step) fix;
dcl O fix(31);
dcl (U,F,T,X1,X2,Y1,Y2,X,Y,DX,DY,R1,R2,R3,R4,ANGLE,DANGLE) flt;
dcl (SINT,COST) flt;
dcl (numa,offa)[0:479] fix;
dcl vert[0:479,0:9] fix;
dcl big[0:0] fix(7) based;     /* really 0:30719 */
dcl p[0:9] ptr;
dcl dn char(168)vary;
dcl (str,name) char(32)vary;
dcl up char(1);
dcl (break,systemao) cond;
dcl (correct,correct2,correct3) flt;
dcl D[0:479] flt;
dcl (i,j) fix;
dcl xrad[1:6] flt;
dcl (A,B,EE,FY,FAX,E) flt;
dcl yrad[1:6] flt;
dcl (sx,sy)fix;


dcl solve entry(fix,flt);
dcl tab$big entry (fix,fix,fix,bit(8));
dcl (ram$cursor,ram$write_block) entry (fix,fix,fix,fix,fix);
dcl (math$atan,math$sqrt,math$cos,math$sin) entry (flt) rtns (flt);
dcl hcs$initiate entry (char(168)vary,char(32)vary,ptr,fix(31));
dcl hcs$terminate entry (ptr);
dcl (com_error,askn,ioa,ioan) entry options (variable);
dcl scs$cl entry options(variable);


     on systemao begin; call ioan ("SYSTEMAO "); end;
     do oy = 0 to 479;
          num = oy/48;
          numa[oy] = num;
          offa[oy] = (oy-num*48)*640;
     end;
     call askn ("which picture to unwrap? (must be 8 bit) ",str);
     dn = ">pix>"||str;
     do cruel = 0 to 9 by 2;
          name = str||".pic"||char(cruel/2,1);
          call hcs$initiate (dn,name,p[cruel],O);
          if O<0 then do;
               call com_error (O,"secwrap",name);
               return;
          end;
          p[cruel+1] = addrel(p[cruel],30720);
```

```
        end;
        call askn ("was the camera pointing up or down? (u or d) ",up);



        do i=1 to 3;
             call ioa("Tap Inner Radius");
             call getpt(tx,ty);
             xrad[i]=tx;yrad[i]=ty;
             call ioa("x=^f,y=^f",xrad[i],yrad[i]);
        end;
        do i=4 to 6;
             call ioa("Tap Outer Radius");
             call getpt(tx,ty);
             xrad[i]=tx;yrad[i]=ty;
/*           call ioa("x=^f,y=^f",xrad[i],yrad[i]);*/
        end;
        call ioa("Tap left edge at horizon");
        call getpt(sx,sy);
             call ioa("Tap another horizon point");
             call getpt(x,y);
        call ram$cursor(0,0,0,0,0);
        call ram$cursor(1,0,0,0,0);
        call scs$cl("tmr tab");
        call center(xrad[1],yrad[1],xrad[2],yrad[2],xrad[3],yrad[3],DX,DY);
        tx=DX;ty=DY;
        call ioa("center1 x=^i y=^i",tx,ty);
        call center(xrad[4],yrad[4],xrad[5],yrad[5],xrad[6],yrad[6],DX,DY);
        cx=DX;cy=DY;
        call ioa("center2 x=^i,y=^i",cx,cy);
        cx=(cx+tx)/2;
        cy=(cy+ty)/2;



        DX=xrad[2]-cx; DY=yrad[2]-cy;
        R1=math$sqrt(DX*DX+DY*DY);
        DX=xrad[5]-cx; DY=yrad[5]-cy;
        R2=math$sqrt(DX*DX+DY*DY);


        h = 480;
        if up="u" then do; here = 0; there = h-1; step = 1; end;
        else do; here = h-1; there = 0; step = -1; end;

        E=320./(math$sin(45./360.*2.*3.14159)/math$cos(45./360.*2.*3.14159));

             X2=x-cx; Y2=y-cy;
             R4=math$sqrt(X2*X2+Y2*Y2);
             x=480.*((R2-R4)/(R2-R1));
             X2=X2*(R2/R4);
             Y2=-Y2*(R2/R4);

        X1=sx-cx; Y1=sy-cy;
             sx=x;
        R3=math$sqrt(X1*X1+Y1*Y1);
        X1=X1*(R2/R3);
        Y1=-Y1*(R2/R3);
        sy=480.*((R2-R3)/(R2-R1));
        call ioan("Horizon = ^i",sy);
```

```
          call ioan("Horizon test = ^i",sx);

     X=X2-X1; Y=Y2-Y1;
     T=math$sqrt(X*X+Y*Y)/2.;
     ANGLE=2*math$atan(T/math$sqrt(R2*R2-T*T));
     DANGLE=0;

     px = -1;
     py = -1;
     on break begin; goto done; end;
     EE=0;

     do x = 0 to 639;

          line=x-((x/10)*10);
/*        call ioan ("^3i ",x);*/
/*        call ram$cursor (1,0,1,cx+X1,cy+Y1);*/
          X = X1;
          Y = Y1;


          do y = here to there by step;
               ox = cx+X;
               oy = cy-Y;
               if ox=px then if oy=py then do; /* same spot as last time? */
                    vert[y,line] = hw;
                    goto skip_the_shit;
               end;
               hw = 0;
               if oy>=0 then if oy<=479 then if ox>=0 then if ox<=639 then do;
                    num = numa[oy];        /* remember: (-23)/48=0 */
                    bot = p[num]->big[offa[oy]+ox];
                    px = ox;
                    py = oy;
               end;
               vert[y,line] = hw;
skip_the_shit:


               ax=x-320;

               FAX=ax;
               FY=y;
               EE=sy+(sx-sy)*(DANGLE)/ANGLE;
               z=EE-((EE-FY)*(E/math$sqrt(FAX*FAX+E*E)));


               if x=0 then do;
                    correct=1-y/(h-1.0);


                    D[y]=(math$sin(3.14159*correct-3.14159/2.)+1.)/2.;


                    X=X1-(X1*(1-R1/R2))*(1-D[z]);
                    Y=Y1-(Y1*(1-R1/R2))*(1-D[z]);
               end;
               else do;
                    X=X1-(X1*(1-R1/R2))*(1-D[z]);
                    Y=Y1-(Y1*(1-R1/R2))*(1-D[z]);
          end;

     end;
     ax=x-320;
     if ax<0 then j=ax+1;
          else j=ax-1;


     FAX=ax;
     FY=j;
     DANGLE=((45.*2.*3.14159)/360.)+math$atan(FY/E);

     A=math$sqrt(E*E+FAX*FAX);
     B=math$sqrt(E*E+FY*FY);
     T=-(1.-A*A-B*B)/(2.*B);
     COST=T/A;
     SINT=math$sqrt(1.-COST*COST);


     X2 = (COST*X1+SINT*Y1);
     Y1 = (COST*Y1-SINT*X1);
     X1 = X2;

     if line=9 then
      call ram$write_block (x-9,0,x,h-1,vert);

     end;

done:
/*   call ram$cursor (0,0,0,0,0);*/
/*   call ram$cursor (1,0,0,0,0);*/
     do cruel = 0 to 9 by 2;
          call hcs$terminate (p[cruel]);
     end;

getpt: proc (x,y);
dcl (x,y,z) fix;
dcl stat bit(8);
     do z=0 while (z=0);
          call tab$big (x,y,z,stat);
     end;
      do while (z<0);
          call tab$big (x,y,z,stat);
     end;
     call ioan ("\g");
end getpt;


perpend:proc(x1,y1,x2,y2,xm,ym,slope);

dcl (x1,x2,y1,y2,xm,ym,slope)flt;


slope=-(x2-x1)/(y2-y1);
xm=(x2+x1)/2.;
ym=(y2+y1)/2.;

end perpend;
```

```
center: proc(x1,y1,x2,y2,x3,y3,xcen,ycen);

dcl(x1,x2,x3,y1,y2,y3,xcen,ycen)flt;
dcl a[20,21] flt;
dcl (xa,ya,slopa) flt;
dcl (xd,yd) flt;
dcl (i,j) fix;



do i=1 to 2;
     if i=1 then do;
          xd=x1;
          yd=y1;
     end;
     else do;
          xd=x3;
          yd=y3;
     end;
     call perpend(xd,yd,x2,y2,xa,ya,slopa);
     a[i,1]=-slopa;
     a[i,2]=1;
     a[i,3]=ya-slopa*xa;
end;


call solve(2,a);
xcen=a[1,3];
ycen=a[2,3];

end center;

end secwrap;
```

```
fishy: proc;

/*
6mm fish-eye images projected to linear perspective

 Steve Yelick 4 March 80
*/

dcl 1 split,
      2 top fix(7) init(0),
      2 bot fix(7);
dcl hw fix defined split;
dcl (ax,z,line,h,x,y,ox,oy,cx,cy,tx,ty,px,py,num,cruel) fix;
dcl O fix(31);
dcl (XX,YY,ysin,ycos) flt;
dcl char[0:4] char(1);
dcl (f,vy,vx,vz,theta,stheta,yphi,phi,r,samx,samy,DX,DY,R1) flt;
dcl (X,Y) flt;
dcl (numa,offa)[0:479] fix;
dcl vert[0:479,0:9] fix;
dcl big[0:0] fix(7) based;      /* really 0:30719 */
dcl p[0:9] ptr;
dcl dn char(168)vary;
dcl (str,name) char(32)vary;
dcl up char(1);
dcl (break,systemao) cond;
dcl (correct,correct2,correct3) flt;
dcl D[0:479] flt;
dcl (i,j) fix;
dcl xrad[1:6] flt;
dcl (A,B,EE,FY,FAX,E) flt;
dcl yrad[1:6] flt;
dcl (sx,sy)fix;


dcl cursor entry(fix,fix);
dcl (ram$cursor,ram$write_block) entry (fix,fix,fix,fix,fix);
dcl (math$atan,math$sqrt,math$cos,math$sin) entry (flt) rtns (flt);
dcl math$atan2 entry(flt,flt) rtns(flt);
dcl hcs$initiate entry (char(168)vary,char(32)vary,ptr,fix(31));
dcl hcs$terminate entry (ptr);
dcl (com_error,askn,ioa,ioan) entry options (variable);
dcl scs$cl entry options(variable);

char[0]="0";
char[1]="1";
char[2]="2";
char[3]="3";
char[4]="4";

    on systemao begin; call ioan ("SYSTEMAO "); end;
    do oy = 0 to 479;
        num = oy/48;
        numa[oy] = num;
        offa[oy] = (oy-num*48)*640;
    end;
    call askn ("which picture to unwrap? (must be 8 bit) ",str);
    dn = ">pix>"¦¦str;
    do cruel = 0 to 9 by 2;
        name = str¦¦".pic"¦¦char[cruel/2];
```

```
     call hcs$initiate (dn,name,p[cruel],O);
     if O<0 then do;
          call com_error (O,"fishy",name);
          return;
     end;
     p[cruel+1] = addrel(p[cruel],30720);
end;

do i=1 to 3;
     call ioa("Tap Outer Radius");
     call cursor(tx,ty);
     xrad[i]=tx;yrad[i]=ty;
     call ioa("x=^f,y=^f",xrad[i],yrad[i]);
end;

call center(xrad[1],yrad[1],xrad[2],yrad[2],xrad[3],yrad[3],DX,DY);
cx=DX;cy=DY;
call ioa("center x=^i,y=^i",cx,cy);
DX=xrad[2]-cx; DY=yrad[2]-cy;
R1=math$sqrt(DX*DX+DY*DY);

call ioa("Tap subject");
call cursor(sx,sy);
yphi=-(math$sqrt((sx-cx)*(sx-cx)+(cy-sy)*(cy-sy))/R1)*110.*3.14159/180.;
call askn("What is the view focal length?(try 30)^f",f);
f=(f*320.)/17.5;
ysin=math$sin(yphi); ycos=math$cos(yphi);

XX=sx-cx;
YY=cy-sy;

stheta=math$atan2(YY,XX);

px = -1;
py = -1;
on break begin; goto done; end;

do x = 0 to 639;
     X=x;
     line=x-((x/10)*10);
     vx=x-320.;

     do y=0 to 479;
          Y=y;
          vy=(240.-Y)*ycos+f*ysin;
          vz=-(240.-Y)*ysin+f*ycos;

          theta=math$atan2(vx,-vy);

          phi=math$atan2(math$sqrt(vx*vx+vy*vy),vz);

          r=phi/(110.*3.14159/180.)*R1;

          samx=r*math$cos(theta+stheta)+cx;
          samy=cy-r*math$sin(theta+stheta);

          ox=samx; oy=samy;

          if ox=px then if oy=py then do; /* same spot as last time? */
               vert[y,line] = hw;
```

```
                    goto skip_the_shit;
                end;
                hw = 0;
                if oy>=0 then if oy<=479 then if ox>=0 then if ox<=639 then do;
                    num = numa[oy];        /* remember: (-23)/48=0 */
                    bot = p[num]->big[offa[oy]+ox];
                    px = ox;
                    py = oy;
                end;
                vert[y,line] = hw;
skip_the_shit:


            end;

            if line=9 then
             call ram$write_block (x-9,0,x,479,vert);

        end;

done:
/*    call ram$cursor (0,0,0,0,0);*/
/*    call ram$cursor (1,0,0,0,0);*/
      do cruel = 0 to 9 by 2;
          call hcs$terminate (p[cruel]);
      end;


perpend:proc(x1,y1,x2,y2,xm,ym,slope);

dcl (x1,x2,y1,y2,xm,ym,slope)flt;


slope=-(x2-x1)/(y2-y1);
xm=(x2+x1)/2.;
ym=(y2+y1)/2.;

end perpend;


center: proc(x1,y1,x2,y2,x3,y3,xcen,ycen);

dcl(x1,x2,x3,y1,y2,y3,xcen,ycen)flt;
dcl a[20,21] flt;
dcl (xa,ya,slopa) flt;
dcl (xd,yd) flt;
dcl (i,j) fix;


do i=1 to 2;
     if i=1 then do;
          xd=x1;
          yd=y1;
     end;
     else do;
          xd=x3;
          yd=y3;
     end;

     call perpend(xd,yd,x2,y2,xa,ya,slopa);
```

```
    a[i,1]=-slopa;
    a[i,2]=1;
    a[i,3]=ya-slopa*xa;
end;

call solve(2,a);

xcen=a[1,3];
ycen=a[2,3];

end center;

solve: proc(order,array);
dcl array[20,21] flt;
dcl (order,m,n,l1,l2,r1,l3,x,y) fix;
dcl r2 flt;
m=order; n=order+1;

do x=1 to m;
    r1 = m;
    do l1=m-1 to x by -1;
        if (array[r1,x]*array[r1,x]) < (array[l1,x]*array[l1,x]) then r1=l1;
    end;
    do l2= n to x by -1;
        r2= fna(fna(array[r1,l2])/fna(array[r1,x]));
        array[r1,l2]= array[x,l2];
        array[x,l2]= r2;
    end;
    array[x,x]=1;
    do y = 1 to m;
        if x~=y then do;
            do l3= x+1 to n;
                array[y,l3]= fna(array[y,l3])-fna(fna(array[x,l3])*
                    fna(array[y,x]));
            end;
            array[y,x]=0;
        end;
    end;
end;
end solve;

fna: proc(x) returns(flt);
dcl x flt; dcl y fix;


return(x);
end fna;

end fishy;
```

Acknowledgements

I would like to thank Andy Lippman for the opportunity to prepare this thesis, and for continuing direction during the work. I would also like to thank Paul Heckbert for programming assistance; Christian Lischewski for many of the original images; and Mike Naimark for raising pertinent questions.