3-D Transformations of Images in Scanline Order

Ed Catmull and Alvy Ray Smith
Lucasfilm Ltd.
P.O. Box 7
San Anselmo, CA 94960

Published in: SIGGRAPH '80 Conference Proceedings, Jul 14-18, 1980, Seattle, WA, edited by James J. Thomas, 279-285. This document was reentered by Alvy Ray Smith in Microsoft Word form on Mar 24, 1999. Spelling and punctuation are generally preserved, but trivially minor spelling errors are corrected. Otherwise additions or changes made to the original are noted inside square brackets. The following note accompanies the original document:

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

©1980 ACM 0-89791-021-4/80/0700-0279 $00.75

Abstract

Currently texture mapping onto projections of 3-D surfaces is time consuming and subject to considerable aliasing errors. Usually the procedure is to perform some inverse mapping from the area of the pixel onto the surface texture. It is difficult to do this correctly. There is an alternate approach where the texture surface is transformed as a 2-D image until it conforms to a projection of a polygon placed arbitrarily in 3-space. The great advantage of this approach is that the 2-D transformation can be decomposed into two simple transforms, one in horizontal and the other in vertical scanline order. Sophisticated light calculation is also time consuming and difficult to calculate correctly on projected polygons. Instead of calculating the lighting based on the position of the polygon, lights, and eye, the lights and eye can be transformed to a corresponding position for a unit square which we can consider to be a canonical polygon. After this canonical polygon is correctly textured and shaded it can be easily conformed to the projection of the 3-D surface.

KEY WORDS AND PHRASES: texture mapping, scanline, algorithm, spatial transforms, 2-pass algorithm, stream processor, warping, bottleneck, foldover

CR CATEGORY: 8.2

Introduction

Texture mapping is an immensely powerful idea now being exploited in computer graphics. It was first developed by one of the authors [2] and extended by Blinn [1] who produced some startling pictures. In this paper we present a

new approach to texture mapping that is potentially much faster than previous techniques and has fewer problems.

The two chief difficulties have been aliasing and the time it takes to do the transformation of a picture onto the projection of some patch. Usually the procedure is to perform some inverse mapping of a pixel onto a surface texture (Fig. 1). It is difficult to do this correctly because the inverse mapping does not happen in scanline order and also because we must integrate under the whole inverse image in order to prevent sampling errors.

We present in this paper an approach that does the mapping in scanline order both in scanning the texture map and in producing the projected image. Processing pixels in scanline order allows us to specify hardware that may work at video rates. We emphasize, however, that the approach is valuable for software as well as hardware implementations.

One of the key concepts we use is that of a “stream processor”. Pixels enter the stream processor at video rate, are modified or merged in some way with another incoming stream of pixels and then sent to the output (Fig. 2). This concept has been implemented by several manufacturers for image processing. A generalization of the concept would be to allow the framebuffers to feed the streams in either horizontal or vertical scanline order.

We will show here that the class of transformations that can be applied to streams is much broader than previously believed. For example, an image in a framebuffer may be rotated by some arbitrary angle even though the data is sent through the processor in scanline order only. While this concept has been known for some time [3, 4], we show here that the technique can be generalized to perspective projections. Further generalizations include quadric and bivariate curved surfaces. The ability to transform a whole raster image very quickly lets us consider doing shading calculations on a unit square where the calculations may be more amenable to stream processing and then transforming the results.

When we say “scanline order”, we use a slightly broader meaning than normal. Usually this means that the order of the pixels is from left to right across a scanline and that the scanlines come in top to bottom order. We broaden the definition to include vertical scanline order. In addition, the scanlines may also occur in bottom to top or right to left order. This gives us trivially a 90 degree rotate and flopping a picture over in one pass through the picture.

Example: Simple Rotation

For illustration, we present the simple case of rotation. We would like to rotate an entire image in the framebuffer. The rotation matrix is:

$$[x' \quad y'] = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where x and y refer to coordinates in the original picture and x', y' are the new coordinates (c = cos, s = sin).

We want to transform every pixel in the original picture. If we hold y constant and move along x then we are transforming the data in scanline order but the results are not coming out in scanline order. Not only is this inconvenient, it is also difficult to prevent aliasing errors.

There is an alternate method for transforming all of the pixels, and that is to evaluate only the x' of the first pass and then the y' in a second pass.

So again hold y constant, but just evaluate x':

$$[x' \quad y] = [cx - sy \quad y].$$

We now have a picture that has been skewed and scaled in the x direction, but every pixel has its original y value. See Fig. 4, where Fig. 4a is the original picture and Fig. 4b is after the horizontal scanline computation.

Next we can transform the intermediate picture by holding x' constant and calculating y. Unfortunately, the equation $y' = sx + cy$ can't be used because the x value for that vertical scanline is not the right one for the equation. So let us invert x' to get the correct x. we need x in terms of x'.

Recall $x' = cx - sy$, so

$$x = \frac{x'}{c} + \frac{sy}{c}.$$

Plug this into $y' = sx + cy$ to get:

$$y' = \frac{sx' + y}{c}.$$

Now we transform the y value of the pixels in the intermediate picture in vertical scanline order to get the final picture, Fig. 4c.

The first pass went in horizontal scanline order on input and output. The second was vertical in both. So in two passes the entire picture was rotated.

Before we generalize, two points should be noted.

(1) A 90 degree rotate would cause the intermediate picture to collapse to a line. It would be better to read the scanlines horizontally from the source buffer and write them vertically to effect that rotate. It follows that an 80 degree rotate should be performed by first rotating 90 degrees as noted then by –10 degrees using the 2-pass algorithm.

(2) The rate at which pixels are read from the input buffer is generally different than the rate at which they are sent to the output buffer. If we're not careful we could get sampling problems. However, since all of the pixels pass through the processor, it is not difficult to filter and integrate the incoming values to get an output value.

Next we generalize to:

$$[x' \quad y'] = [X(x,y) \quad Y(x,y)].$$

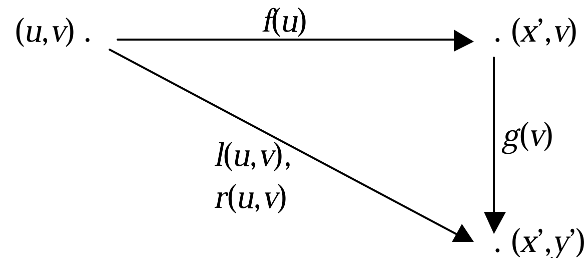
This generalization will include perspective. Whatever the transformation is, we shall show that we can do the x transforms first, followed by the y transforms, but in order to do the transforms we must be able to find the inverse of x'. This

may be very difficult to do and *x*' may even have multiple values. So we present first a more formal way of talking about the method before addressing some of the difficulties.

The 2-Pass Technique

We are interested in mapping the 2-D region bounded by a unit square into a 3-D surface which is projected back into 2-D for final viewing. Since the unit square (by which we mean the enclosed points also) may be represented by point samples in a digital framebuffer, and since a framebuffer is typically arranged in rows and columns, we are interested in row-ordered or column-ordered implementations of these mappings. The technique we now present is a means of decomposing a 2-D mapping into a succession of two 1-D mappings, or scanline-ordered mappings, where a scanline may be either horizontal (a row) or vertical (a column). The technique is quite general as we shall show subsequently.

This figure illustrates the 2-pass technique:



We want to map the set of points $\{(u,v): 0 < u < 1, 0 < v < 1\}$ in the unit square into the set $\{(x',y')\}$ where the desired mapping is given as an arbitrary pair of functions

$$x' = l(u,v)$$
$$y' = r(u,v)$$

We wish to replace this pair of functions with the pair

$$x' = f(u)$$
$$y' = g(v)$$

where it is understood that $f(u)$ is applied to all points in the unit square before $g(v)$ is applied to any of them. We call the application of *f* the h-pass (for horizontal) and the application of *g* the v-pass (for vertical).

In general, there will be a different $f(u)$ for each value of *v*, so *f* might be thought of as a function of (*u*, *v*). We prefer however to think of *v* as a parameter which selects a particular $f(u)$ to be applied to all *u* on scanline *v*. To emphasize when *v* is being held constant like this, we will use the notation $\tilde{v}$. Thus $\tilde{v}$ is an index into a table of horizontal mappings. Similarly, there will in general be a different $g(v)$ for each vertical scanline *x*' (where the prime indicates that the h-pass

has already occurred). We will use the notation $\tilde{x}'$ to indicate a given vertical scanline just prior to the v-pass.

In this section, we will always have the v-pass follow the h-pass. This is just a convenience. The decomposition into the other order proceeds similarly, and we will have occasion to choose one order over the other in a later section.

An algorithm for the decomposition of *l*, *r* into *f*, *g* is the following:

(1) $f(u) = l(u, \tilde{v})$ is the function *f* for scanline $\tilde{v}$.

(2) Solve the equation $l(u,v) - \tilde{x}' = 0$ for *u* to obtain $u = h(v)$ for scanline $\tilde{x}'$.

(3) $g(v) = r(h(v), v)$ is the function *g* for scanline $\tilde{x}'$.

We simply take $f(u)$ as defined in (1) and show that $g(v)$ in (3) is consistent with it. The h-pass takes the set of points $\{(u,v)\}$ into the set $\{(x', v)\}$. We desire a function which may be applied at this time to scanline $\tilde{x}'$. But being given $\tilde{x}'$ is equivalent to being given the equation

$$\tilde{x}' = l(u, v) .$$

If this equation can be rearranged to have the form $u = h(v)$, then $r(h(v), v)$ is a function of *v* only and is the desired *g*.

Thus solving the equation $l(u,v) - \tilde{x}' = 0$ for *u* is the key to the technique. We shall show some cases where this is simple, but in general it is not. An iterative solution such as provided by Newton-Raphson iteration could be used but is expensive. We shall treat these problems in the following sections.

It should be noted that we have placed no restrictions on functions *l*, *r*. So the 2-pass technique can be applied to a large class of picture transformations and distortions, only a few examples of which will be presented here. In particular, we henceforth restrict our attention to ratios of polynomials.

We shall illustrate the 2-pass technique by applying it, in detail, to the case of a rectangle undergoing affine transformations followed by a perspective transformation and projection into 2-space. Then, in less detail, we will treat bilinear and biquadratic patches under the same type of transformation. This should serve to indicate how the method can be extended to higher degree surfaces.

The Simple Rectangle

Consider the (trivial) parametric representation of a rectangle given by $x(u,v) = u$, $y(u,v) = v$, $z(u,v) = 0$, $w(u,v) = 1$. The class of transformations we apply are exactly those which can be represented by a 4x4 matrix multiplying a 3-space vector represented in homogeneous coordinates as indicated below:

$$[x \quad y \quad z \quad w] \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} = [x'' \quad y'' \quad z'' \quad w''] .$$

Then projection into 2-space is accomplished by dividing through the homogeneous coordinate w”:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} \frac{x''}{w''} & \frac{y''}{w''} & \frac{z''}{w''} \end{bmatrix}$$

Replacing x, y, z, and w with their parametric forms and expanding the equations above gives

$$x' = \frac{au + bv + d}{mu + nv + p} = l(u, v)$$
$$y' = \frac{eu + fv + h}{mu + nv + p} = r(u, v) .$$

We are interested in the 2-D projection only so we shall ignore z' from here on.

The functions l, r in this case represent an ordinary linear transformation of the unit square, followed by a perspective projection. Notice that they are both rational linear polynomials—i.e., a linear polynomial divided by a linear polynomial.

Applying the 2-pass algorithm to the functions l, r gives:

(1) The h-pass function for scanline $\tilde{v}$ is

$$f(u) = \frac{Au + B}{Cu + D}$$

where $A = a$, $B = b\tilde{v} + d$, $C = m$, $D = n\tilde{v} + p$.

(2) $u = h(v)$ is obtained by solving

$$\tilde{x}' = \frac{au + bv + d}{mu + nv + p}$$

for u:

$$u = Ev + F$$

where $E = \frac{b - n\tilde{x}'}{m\tilde{x}' - a}$ and $F = \frac{d - p\tilde{x}'}{m\tilde{x}' - a}$.

(3) Thus

$$g(v) = \frac{e(Ev + F) + fv + h}{m(Ev + F) + nv + p} = \frac{Gv + H}{Iv + J}$$

is the v-pass function for scanline $\tilde{x}'$, where $G = f + eE$, $H = h + eF$, $I = n + mE$, $J = p + mF$.

Fig. 5 shows the results of applying this f, g pair. Fig. 5a is the original rectangular texture. Fig. 5b is its appearance after the h-pass, and Fig. 5c is the result of the v-pass.

Following are several points about this computation:

(1) The sampled image (Fig. 5a) was reconstructed with a first-order filter (the so-called Bartlett window) then resampled with a zeroth-order filter (the Fourier window). This is only minimal use of sampling theory. A piece of

hardware or software for production quality work would certainly employ more sophisticated filtering. Our figures look surprisingly nice despite use of the low-order filters mentioned above. (The edges are not antialiased, however.)

(2) Clipping is natural. The f function generates the final value of x'. If this value should fall outside the limits of the output buffer then it does so with no loss. The g function, which operates only on the scanlines output by f, will never need values clipped in the h-pass.

(3) The 2-pass technique does not avoid the ordinary problems of perspective projections. For example, the transformation can blow up if the denominator of either f or g goes to zero. This corresponds to the usual problem of wrap-around through infinity and requires the normal solution of clipping before transformation.

(4) There is a problem introduced by the 2-pass technique not encountered before. This is what we call the "bottleneck problem". We shall discuss this in greater detail and offer a solution to it in the next section, then return to the examples.

Bottleneck

With the perspective transformation we have a problem analogous to that of the 90 degree rotate, that is, it is possible to have an intermediate picture collapse. In the case of rotation the solution was simple: rotate the texture 90 degrees and change the transformation by that amount. The solution for the perspective case is the same, however it is more difficult to tell from the transformation matrix when a problem will occur.

We base our criteria on the area of the image in the intermediate picture. There are four possible ways to generate an intermediate picture:

1. transform x first
2. transform y first
3. rotate 90 degrees and transform x first
4. rotate 90 degrees and transform y first

In each case the area is easily found by integrating the area between $x'(0, y)$ and $x'(1, y)$ where

$$x' = \frac{ax + by + c}{dx + ey + f}$$

and y varies from 0 to 1. This gives

$$area = K \ln\left(1 + \frac{e}{d + f}\right) - k \ln\left(1 + \frac{e}{f}\right)$$

where $K = \frac{(ce - bf) + (ae - bd)}{e^2}$, and $k = \frac{cd - bf}{e^2}$. We use the method that gives the maximum intermediate area.

The Bilinear Patch

The preceding class of transformations of the rectangle does not generate all quadrilaterals—e.g., nonplanar quadrilaterals. Since in general we cannot guarantee that all quadrilaterals are planar, we generalize to the bilinear patch.

The general bilinear patch (Fig. 3a) has a parametric representation

$$x(u,v) = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix}$$

where $a_{00} = (x_3 - x_2) - (x_1 - x_0)$, $a_{01} = x_1 - x_0$, $a_{10} = x_2 - x_0$, $a_{11} = x_0$. There are similar representations for $y(u,v)$, $z(u,v)$, and $w(u,v)$, where b_{ij}, c_{ij}, and d_{ij} correspond respectively to the a_{ij} for $x(u,v)$.

As in the preceding example we transform a bilinear patch with a 4x4 matrix multiply followed by a projection into 2-space. Hence we shall again ignore z' (but see discussion of foldover below). The transformation may be represented by the following matrix equation:

$$\begin{bmatrix} x'' & y'' & z'' & w'' \end{bmatrix} = \begin{bmatrix} x & y & z & w \end{bmatrix} \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} =$$

$$\begin{bmatrix} uv & u & v & 1 \end{bmatrix} \begin{bmatrix} a_{00} & b_{00} & c_{00} & d_{00} \\ a_{01} & b_{01} & c_{01} & d_{01} \\ a_{10} & b_{10} & c_{10} & d_{10} \\ a_{11} & b_{11} & c_{11} & d_{11} \end{bmatrix} \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} = \begin{bmatrix} uv & u & v & 1 \end{bmatrix} \begin{bmatrix} A & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{bmatrix}.$$

After the homogeneous divide

$$x' = \frac{Auv + Bu + Cv + D}{Muv + Nu + Ov + P} = l(u,v)$$
$$y' = \frac{Euv + Fu + Gv + H}{Muv + Nu + Ov + P} = r(u,v).$$

The 2-pass algorithm gives:

(1) $f(u) = \dfrac{A'u + B'}{C'u + D'}$ for scanline $\tilde{v}$, where $A' = A\tilde{v} + B$, $B' = C\tilde{v} + D$, $C' = M\tilde{v} + N$, $D' = O\tilde{v} + P$.

(2) For vertical scanline $\tilde{x}'$, it can be shown that

$$g(v) = \frac{A''v^2 + B''v + C''}{D''v^2 + E''v + F''}$$

where $A'' = EE' + GG'$, $B'' = EF' + FE' + GH' + HG'$, $C'' = FF' + HH'$, $D'' = ME' + OG'$, $E'' = MF' + NE' + OH' + PG'$, $F'' = NF' + PH'$ with $E' = C - O\tilde{x}'$, $F' = D - P\tilde{x}'$, $G' = M\tilde{x}' - A$, $H' = N\tilde{x}' - B$.

Fig. 6 shows a planar bilinear patch representing the texture in Fig. 5a twisted about its center point. For this particular example, the h-pass is the identity function $f(u)=u$ and hence is not shown. Fig. 7 shows the h-pass and v-pass for a nonplanar patch transformation. Note the foldover. Fig. 5a is the source texture again.

All of the considerations discussed for the simple rectangle apply here also. In addition we have new problems introduced due to the higher complexity of the surface. A bilinear patch may be nonplanar, so from some views it may be double valued. That is, a line from the viewpoint through the surface may intersect the surface twice. In terms of the scanline functions, $g(v)$ can map scanline $\tilde{x}'$ back over [itself.] We call this problem "foldover". It occurs at a silhouette edge of the projected surface. The solution is to compute z' for $v=0$ and for $v=1$. The endpoint of scanline $\tilde{x}'$ which maps into the z' farthest from the [viewpoint] is transformed first, so that later points overwrite points that would be obscured anyway. For antialiasing purposes, the location of the foldover point must be remembered and an appropriate weight computed for combining the pixel there with a background.

The Biquadratic Patch

The highest order patch we shall discuss here is the biquadratic patch (Fig. 3b). It is particularly interesting because surface patches on quadric surfaces (e.g., ellipsoids) may be represented as biquadratic patches. The parametric equation of x for a biquadratic patch has form

$$x(u,v)=\begin{bmatrix} u^2 & u & 1 \end{bmatrix}\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} v^2 \\ v \\ 1 \end{bmatrix}$$

and similarly for $y(u,v)$, $z(u,v)$, and $w(u,v)$.

$$\begin{bmatrix} x'' & y'' & z'' & w'' \end{bmatrix}=\begin{bmatrix} u^2v^2 & u^2v & u^2 & uv^2 & uv & u & v^2 & v & 1 \end{bmatrix}\begin{bmatrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \\ a_5 & b_5 & c_5 & d_5 \\ a_6 & b_6 & c_6 & d_6 \\ a_7 & b_7 & c_7 & d_7 \\ a_8 & b_8 & c_8 & d_8 \end{bmatrix}.$$

It can be shown, in a manner analogous to that used for the bilinear patch, that $f(u)$ is a ratio of quadratic polynomials and $g(v)$ is a ratio of 4th-degree polynomials. Actually there are two v-pass functions—say $g_i(v)$, $i=0$ or 1—one corresponding to each of two solutions of a quadratic equation encountered in

the derivation. The fact that there are two v-pass functions requires an explanation. We now turn to this and other considerations which have been added because of the introduction of higher degree surfaces.

First, we present a technique for reducing $g_i(v)$ from a ratio of 4th-degree forms to a rational quadratic polynomial like $f(u)$. Presumably we could implement the $g_i(v)$ as they stand. However, besides being computationally nasty, they are difficult to interpret and hence hide many pitfalls. For example, the foldover problem discussed in the bilinear case could occur three times in a scanline with attendant antialiasing problems. We prefer to introduce a method which, at the cost of more memory, greatly reduces the complexity of the $g_i(v)$. It can also be applied to the simple rectangle and bilinear patch. Its utility comes however in extending the methods of this paper to higher degree—e.g., to the transformation of bicubic patches in perspective—which we reserve for a future paper.

The notion is that during the h-pass we have already computed the u's which we need in the v-pass. It is the recomputation of these u's (step (2) in the 2-pass algorithm) which makes the v-pass more difficult than the h-pass. We propose a high-precision framebuffer (e.g., 16 bits per pixel) to hold the u's as they are computed during the h-pass. Thus, if u_j maps into x'_j under $f(u)$, then at location x'_j in one framebuffer we store the intensity computed from the neighborhood of u_j in the source picture and in another framebuffer (the one with higher precision) the value u_j itself. Then during the v-pass on scanline $\tilde{x}'_j$, we merely lookup in the extra framebuffer the value of u_j mapped by the h-pass into the current pixel, say $(\tilde{x}'_j, y)$, on the vertical scanline. It will be the u_j at $(\tilde{x}'_j, y)$ in the extra framebuffer.

A difficulty which arises is that the h-pass function $f(u)$ can cause a foldover on horizontal scanlines. This means that the intensity computed at location x'_j is a function of one of two different u_j's. Our solution is to have two auxiliary location framebuffers and one additional intensity framebuffer. During the h-pass a scanline is computed in an order where the deepest points are generated first, as discussed in the bilinear case. As each u_j is determined, it is written into only one of the location framebuffers and the corresponding intensity is written into one of the intensity framebuffers only. This occurs until the u_j corresponding to the point of foldover occurs. From this point on all u_j's are stored in only the other location framebuffer and the corresponding intensities in the other intensity framebuffer. The final image is a combination of the two intensity framebuffers. In general, we believe this to be a difficult hidden surface problem and do not treat it further here.

The simplification produced by the addition of the three extra framebuffers reduces the $g_i(v)$ to

$$g(v) = \frac{B_0'v^2 + B_1'v + B_2'}{D_0'v^2 + D_1'v + D_2'}$$

where, for example, $B_0' = B_0u_j^2 + B_3u_j + B_6$ with u_j being obtained by table lookup. The problem of which $g_i(v)$ is to be used is replaced with the problem of computing in two framebuffers and solving the hidden surface problem implied.

Notice that $g(v)$ may cause foldover in both intensity framebuffers, but in any one framebuffer there is only a single foldover per vertical scanline instead of the triple foldover implied by the original $g_i(v)$ and no auxiliary framebuffers.

We have claimed that the use of additional memory makes unnecessary the determination of the u_j, the inverses of x' under $f(u)$. To make this strictly true, we must make the following observations. A typical way to implement the function $f(u)$ is to step along x' in equal increments (e.g., one pixel increments) and compute the inverse image u. The neighborhood of u is then used to compute the intensity at location x'. Of course, this defeats the whole purpose of avoiding inverses, assuming they can be computed at all. we propose "straightahead" mapping for implementing $f(u)$ to avoid inverses altogether. The idea here is to step along u in equal increments, computing $x' = f(u)$ after each increment. Let x_j' be a value of x' for which we wish to know its inverse image. Let u_i be values of u at the equal increment points used as samples of u. Then when $x_i' = f(u_i)$ is less than x_j' and $x_{i+1}' = f(u_{i+1})$ is greater than x_j', we either

(1) iterate on the interval $[u_i, u_{i+1}]$ to obtain the desired inverse image u_j, or
(2) approximate u_j by $u_j = u_i + a(u_{i+1} - u_i)$, where $a = x_j' - x_i'$.

The figures used to illustrate this paper were generated using the approximation (2) above. Filtering and sampling require integration of the intensity function of u. This integration requires computation at each u_i, so the cost, if any, of straightahead implementation is a small addition to that already required.

Simplifications

Much of the heavy machinery in the examples above becomes unnecessary in the following two special cases:

No perspective: It is easy to see that the division at each output pixel is unnecessary in this case—i.e., the scanline mappings are polynomials instead of ratios of polynomials.

Planar patch: If the patch is known to be 2-D then, regardless of the order of its bounding curves, there can be no foldover problem with the rigid-body transformations considered here. (Lines can completely reverse direction however

(Fig. 6).) Hence no extra framebuffers are needed. The problem simplifies substantially, becoming a 2-D "warp" of a rectangular texture.

For example, a planar biquadratic patch under affine projection only (no perspective) has scanline functions of form $f(u) = au^2 + bu + c$ and $g(v) = dv^2 + ev + f$ and can be accomplished in only one framebuffer.

Shading

People who have implemented hidden surface programs with sophisticated lighting models have discovered that the time spent for the lighting calculation is much greater than the time spent solving the hidden surface problem. We propose here that it may be faster to perform the light calculations on a square canonical polygon and then to transform the results.

Typically, normals for a polygon are determined and then interpolated across segments. The normal at each pixel is dotted with vectors to the lights and eye in some function to find the shading.

While framebuffers have been used to store intensities and depth values, they can also be used to store normal values. The normal values can be kept in a buffer at arbitrary resolution. The stream processor can then interpolate or approximate those normals to get normals at a higher resolution, normalize them, dot them with other streams of normals, and use the dot products in intensity calculations. The approach is:

1. Transform eye and lights relative to canonical polygon.
2. The canonical polygon normal framebuffer is filled with normals at some resolution (say 4 by 4).
3. Generate a high resolution array of normals using cubic splines (we used b-splines) first in the vertical direction, then the horizontal.
4. Normalize the normals.
5. The stream of normals is dotted with a stream of light vectors and/or eye vectors to implement the lighting function.
6. The results are transformed into position into the final framebuffer yielding the shaded polygon.
7. If we are also doing texture mapping, then the intensity of each pixel in the texture is used as the color in the lighting function and the results [are transformed as before.]

The approximation of normals with cubic curves can be done in a stream processor by using difference equations. Each overlapping set of four values can be used to generate a difference equation with a matrix multiply. Then the difference equation is used to generate all of the values. Until normalization, x, y, and z may be treated alike and independently.

Conclusions

We have presented what we believe to be a powerful new way of looking at 3-D surface rendering in computer graphics. It is based on the old notion of transforming to a canonical form, where the difficult work may be performed with relative ease, then transforming back. The success of this notion in 3-D surface graphics depends on the ease of realization of the transformations to and from canonical form. We have shown that a stream processor and the 2-pass decomposition technique give a technologically feasible realization of the notion for modern computer graphics.

There is much work to be done to fully explore this approach. This paper begins the exploration of this territory and points out several of the difficulties peculiar to it.

Acknowledgements

Although we do not know the details of their work, we are aware that Larry Evans and Steve Gabriel have been pursuing an apparently similar line of research and wish to acknowledge them here. Mike Shantz and his colleagues at De Anza Systems Inc. have also done independent work on separable transformations. They have implemented second-order polynomial coordinate transformations in hardware.

The photographs for this paper were prepared by David DiFrancesco at the Jet Propulsion Lab (JPL). The source picture in all cases was generated by Turner Whitted of Bell Labs for SIGGRAPH '79 and is used with his permission and that of the CACM.

Computing facilities at JPL were generously provided by Jim Blinn and Bob Holzman. Facilities were also provided by Tom Ferrin and Bob Langridge of the University of California at San Francisco.

References

[1] James F. Blinn, *Simulation of Wrinkled Surfaces*, SIGGRAPH Proceedings, Aug 1978, 286-292.

[2] Edwin Catmull, *Computer Display of Curved Surfaces*, IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structures, Los Angeles, May 1975.

[3] Steven A. Coons, *Transformations and Matrices*, Course Notes No. 6, University of Michigan, Nov 26, 1969.

[4] A. Robin Forrest, *Coordinates, Transformations, and Visualization Techniques*, University of Cambridge, Computer Laboratory CAD Document 45, Jun 1969.

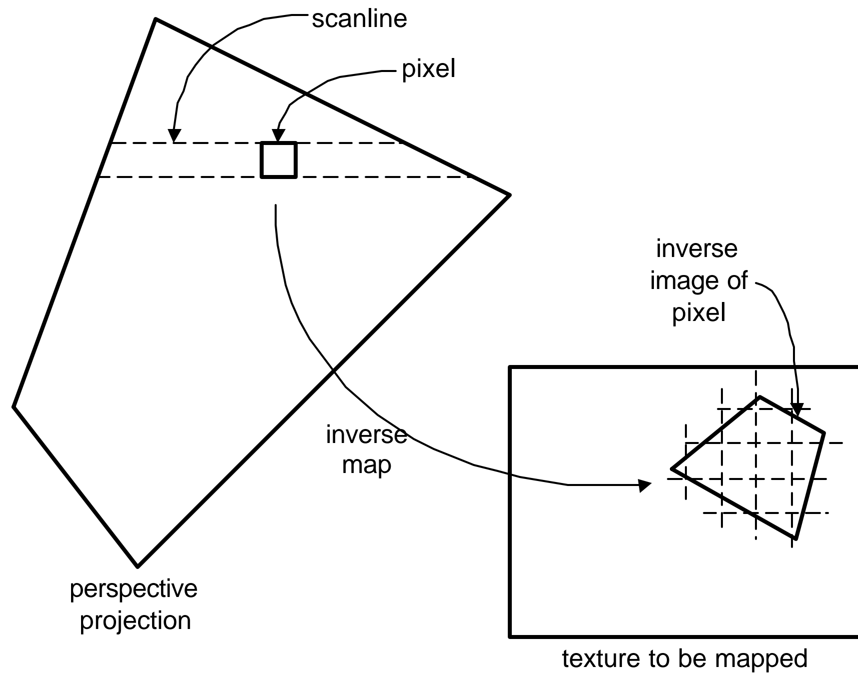


Fig. 1. Texture mapping.

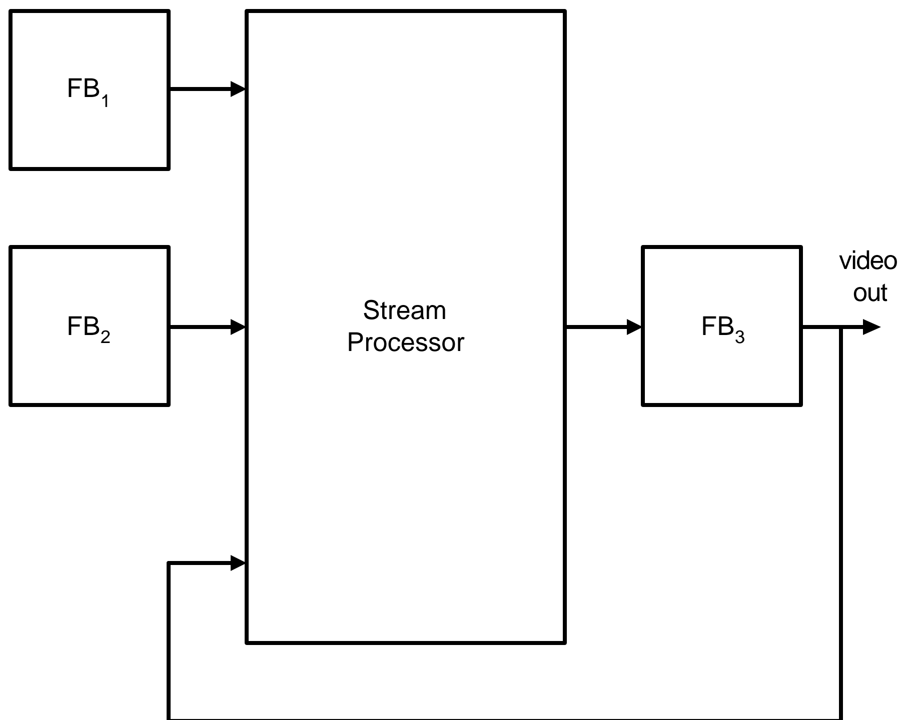


Fig. 2. Stream processor.

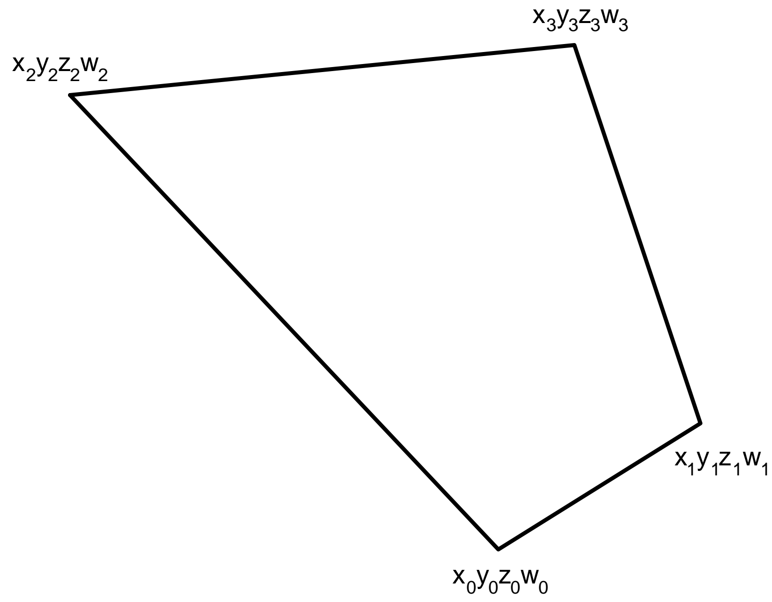


Fig. 3a. Bilinear patch.

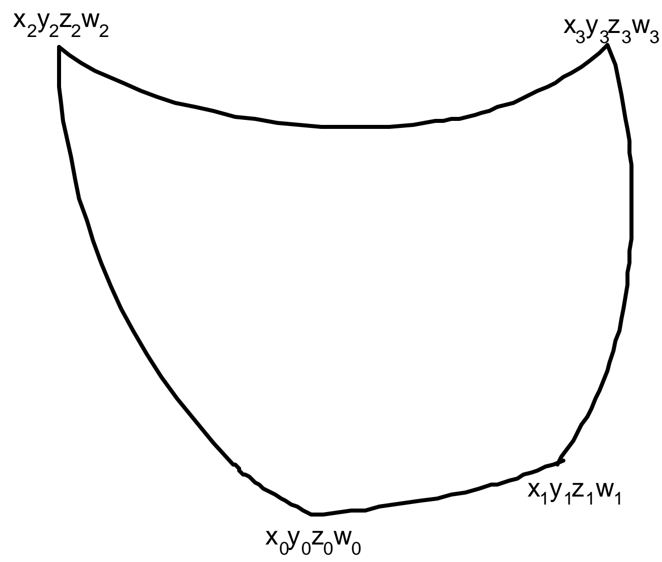


Fig. 3b. Biquadratic patch.

Fig. 4a. Source texture by Turner Whitted. (By permission of CACM).

Fig. 4b. Simple rotate h-pass.

Fig. 4c. Simple rotate v-pass.

Fig. 5a. Source texture by Turner Whitted. (By permission of CACM).

Fig. 5b. Simple rectangle in perspective, h-pass.

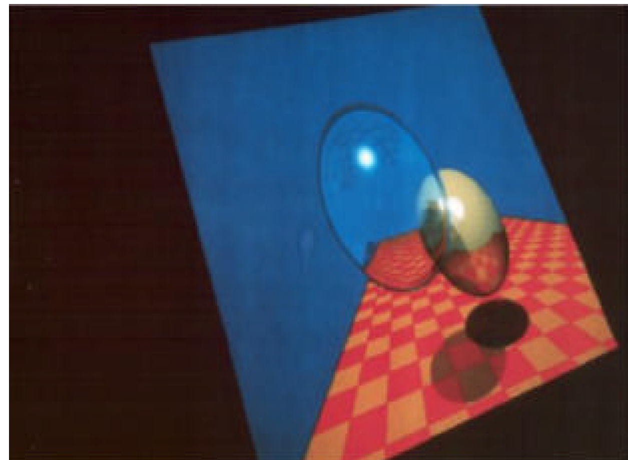


Fig. 5c. Simple rectangle in perspective, v-pass

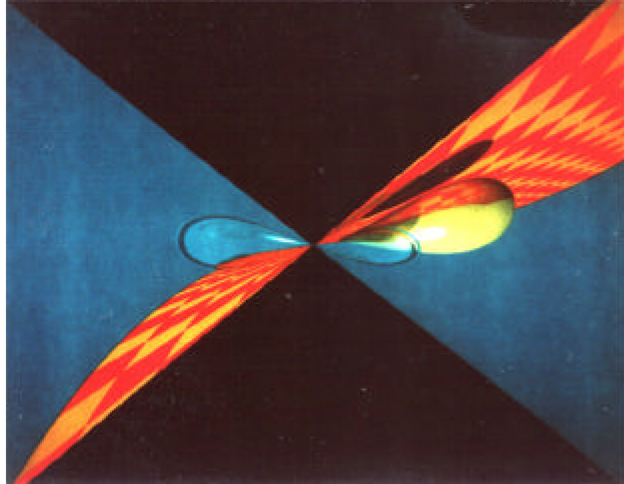


Fig. 6. Planar bilinear patch twisted about midpoint.

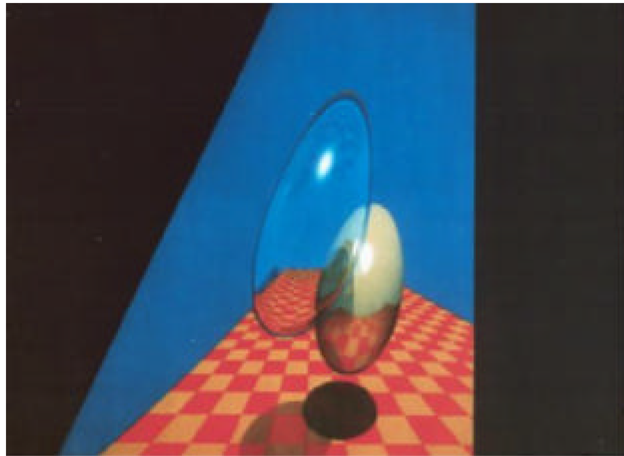


Fig. 7a. Nonplanar bilinear patch h-pass.

Fig. 7b. Nonplanar bilinear patch v-pass.