# POLYGON SHADOW GENERATION

by

Peter Atherton, Kevin Weiler and Donald Greenberg

Program of Computer Graphics

Cornell University

ABSTRACT

A general purpose method for generating shadows using a polygonal coordinate data base is presented. The method is based on an object space polygon clipping hidden surface removal algorithm. Output from the program is in the same three-dimensional polygon format as the input. Thus, a shadowed data environment may be easily created and viewed from any observer position with no additional depth sorting time required for the hidden surface removal process. Shadows can also be cast by more than one light source. Since the shadows are generated in object space, the results can be used for both visual display and numerical analysis.

COMPUTING REVIEWS CLASSIFICATION: 3.2, 4.9, 4.40, 4.41

KEYWORDS: Shadow Generation, Hidden Surface Removal, Polygon Clipping, Graphics

## I. INTRODUCTION

A shadow is the darkness cast by an object intercepting light. It falls from the side opposite the source of light. Theoretically, when the observer's position is coincident with that of the light source, no shadows are visible. Shadows become visible when the viewer's position moves away from the source of illumination.

The addition of shadows to a perspective image vastly improves the depth perception of the display. Furthermore, shadows provide valuable positional information and improve the ability of the observer to comprehend complex spatial environments. However, computation times and algorithmic complexity for shadow generation have prevented many implementations. The shadow creation method presented is a natural extension of an object space hidden surface removal algorithm which uses polygon area sorting and is described in the third section. A major advantage of this method is that both the input and output are in the form of a three-dimensional polygon data structure. This characteristic means that the shadow definitions can be used for the purposes of both display and analysis.

## II. SURVEY OF EXISTING ALGORITHMS

Several classes of algorithms for shadow generation have been previously presented. Each of these approaches has inherent limitations which may restrict their application and use. The raster scan method for shadow image creation was first implemented in 1970 by Kelley and Bouknight[4,5,7] although a similar procedure for line drawing images has been presented by Appel.[1]

Oriented to a raster type display scope, the Kelley and Bouknight method scans an object row by row to determine visibility. Each time a polygon boundary is crossed, a depth sort is made to determine which polygonal surface is nearest the observer (Figure 1). Since the color of a polygon does not change across its surface, the only display information necessary is the location of the "key squares," those raster units in each row where a color change takes place.

Shadows may be added to an image simply by running two concurrent scanning operations, one to determine visible surfaces and one in image space to determine shadow existence. Before scanning, a list is created for each polygon linking it to any other polygon that might cast a shadow upon it.

In 1973 Nishita and Kakamae presented a method for shadow generation based on a convex polyhedron clipping algorithm.[8] This program maintained some of the benefits of the raster scan display method while improving on the accuracy and versatility of the shadow definitions.

The data input base consists of convex polyhedra, each of which may be composed of several convex polygons. Hidden surface removal from any chosen point of view is accomplished by determining the silhouette contours of each polyhedron and using them to define its clipping border. Objects which lay behind a selected polyhedron are clipped to the window defined by the polyhedron's outside boundaries.

The generation of shadowed images by this polyhedron clipping method is accomplished in two basic

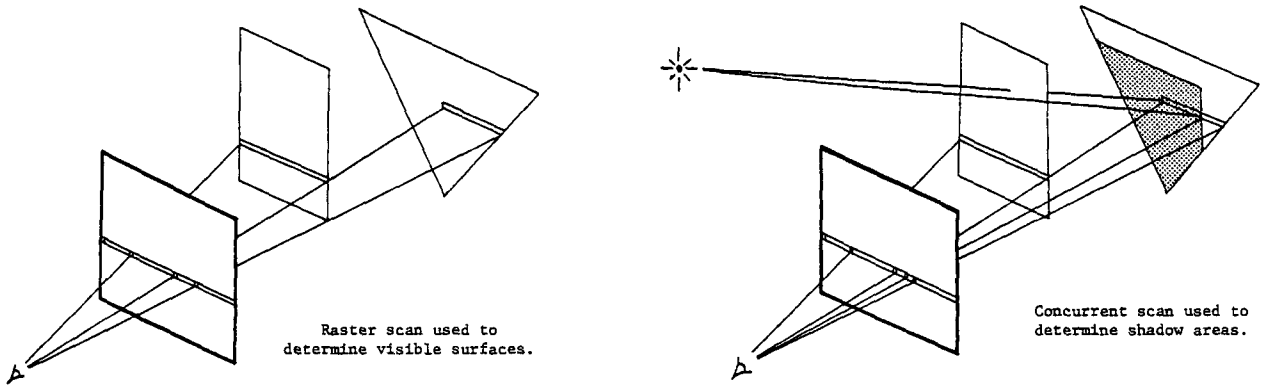



Figure 1. Concurrent scanning method of shadow display.

steps (Figure 2). In the first step, a view is taken in the direction of the infinite light source. Using the polyhedron clipper, all the hidden surfaces, which are surfaces that are in shadow, are found. The entire scene is then transformed to a selected view point, and all hidden surfaces are removed by a raster scan method similar to that used by Bouknight and Kelley.

A third algorithm for generating shadows is based on the concept of computing the surface defining the volume of space swept out by the shadow of an object, its umbra. The umbra surfaces are then added to the data and treated as invisible surfaces which, when pierced, cause a transition into or out of an object shadow. This shadow volume approach was presented by F. Crow in 1977.[6]

For any polyhedron, the shadow volume can be completely described for a given light source position (Figure 3). The contour edges of the original object, as seen from the light source, are first computed. Then all planes defined by the light source and the contour edges constitute the bounding surface of the shadow volume. The "near" surface of the shadow volume is defined by the silhouette edges of the object casting the shadow. The "far" surface is at an infinite distance. This volume is then clipped by the frustrum of vision (or viewbox) and added to the environment data base. Any hidden surface algorithm can then be used to create the display. The shadow data is treated in the same manner as the original data except that it is invisible. In the depth order calculations, any plane behind a front facing shadow surface is in shadow. The method can be coupled with several hidden surface algorithms and has the capability of effectively creating shadow volumes when the illuminating light source is placed within the original environment.

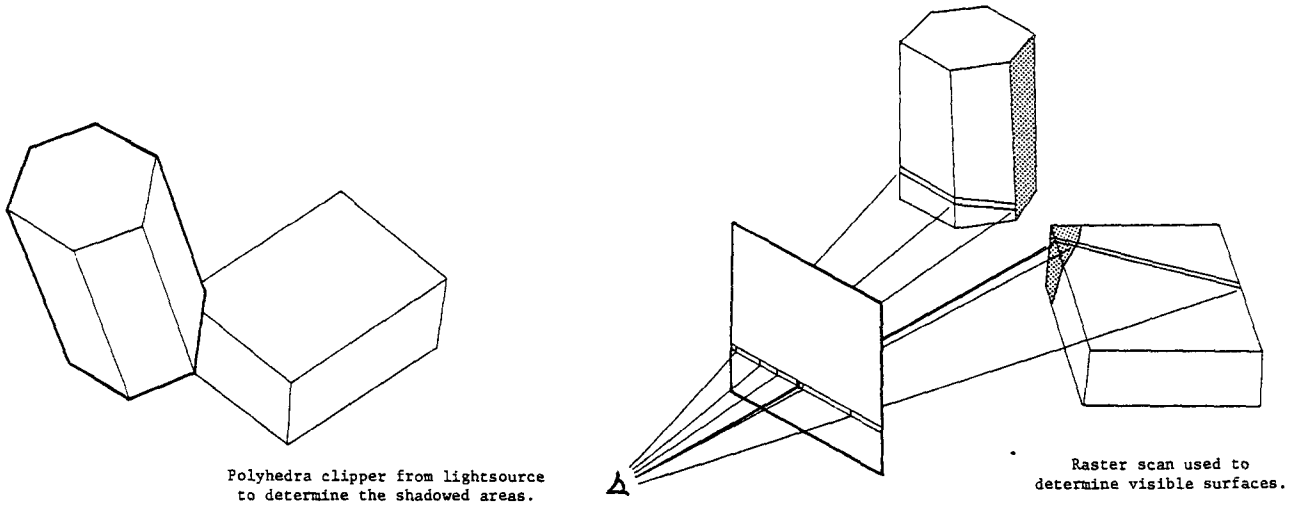



Figure 2. Polyhedra clipper and raster scan method of shadow display.

There are several restrictions with the shadow image creation programs described. The major limitation of the double scanning method and the shadow volume method is that the shadow surfaces are not defined in object space. This precludes the use of the created shadows for accurate computational purposes. Furthermore, scan-line algorithms based on raster display devices determine their depth priority in image space. This limits output portability (e.g., vector displays) and will become less efficient as the display resolution increases. Additional problems, unique to the individual algorithms, are the imposed limitations on the environment description, the potentially large increases in the environment data base, the required maintenance of non-visible polygons as possible shadow casting elements, and the necessity for recalculating shadows for each image.

To overcome these difficulties, a different approach to shadow image generation based on our hidden surface removal method using polygon area sorting has been developed.[2,10,11] This can be accomplished with relative ease since the polygon form of the output is the same as the polygon form of the input.

## III. POLYGON AREA SORTING HIDDEN SURFACE REMOVAL ALGORITHM

A program to remove hidden surfaces by polygon area sorting has been developed at Cornell's Laboratory of Computer Graphics.[10,11] The basic concept of a polygon sorting hidden surface removal algorithm is that all surfaces that lay behind each unique polygonal area and within its borders are removed. The algorithm proceeds from front to back across the transformed object space, producing portions of the final image along the way and temporarily reversing direction only when an initial depth sort error is detected. Output from the algorithm never overlaps on the vertical image plane since each visible area has had all polygons behind it removed. This polygonal area may itself be subdivided recursively if there is an error in the initial depth sort.

The hidden surface removal algorithm involves four basic steps:

1) a preliminary rough depth sort
2) a two-dimensional comparison of the currently most forward polygon, or template, to the remaining polygons
3) removal of polygons that exist behind the template and within its borders
4) a recursive comparison when an error in the preliminary depth sort has occurred.

At the heart of the hidden surface removal process is a polygon clipper. This algorithm considers two polygons at a time, a template or clipping polygon and a subject polygon. The two polygons are compared and the surfaces of the subject polygon existing within the borders of the clipping polygon are designated. Even though the polygon clipper works essentially in two dimensions, all depth information is accurately preserved maintaining the precise three-dimensionality of the polygons.

The polygon clipper is capable of clipping a concave subject polygon with holes to the borders of a concave clipping polygon with holes. This generality is necessary since even when a scene is restricted to convex polygons, a clipping sequence could quickly yield concave areas and holes. Surface details such as texture or color differences can be described as polygons within the boundaries of a parent polygon. These surface details will have a minimal effect on the hidden surface removal process.

## IV. POLYGON SHADOW ALGORITHM

The procedure for creating an image containing shadows consists of two major parts. The first is the creation of the shadow descriptions as dictated by the particular object orientations and light source position. The second is the determination of visible surfaces with their associated shadow descriptions and is dependent upon the observer's position.

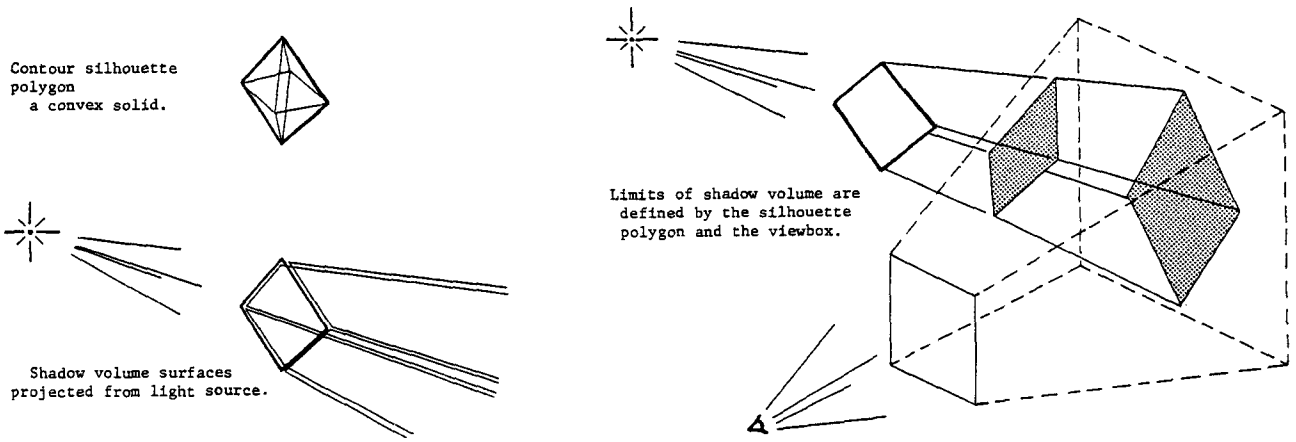



Figure 3. Shadow volume method of shadow display.

By using the general purpose polygon clipping hidden surface removal algorithm previously described, the process of generating shadowed images can be made relatively simple (Figure 4). Shadow descriptions are found by viewing the environment from the position of the light source. A hidden surface removed view from the light source position will delineate the illuminated polygons which are those areas not in shadow. Once defined, these illuminated polygons are added to the original environment and treated as surface details on their original source polygons. This general method is suitable for both point light sources and parallel light sources.

This approach has several distinct advantages. First, since the polygonal output of the hidden surface removal routines is the same as the input, the same logic can be used for the shadow generation and the image display. Second, by maintaining the three-dimensional shadow polygon output, it is possible to compute shadow areas and thus their effect on such phenomena as energy utilization. Third, by adding the shadows to the data base in the form of details attached to "parent" planes, the computational time for the hidden surface removal sorting process does not increase. Fourth, many views can be generated requiring only one original shadow generation cycle. Lastly, shadow views with multiple light sources require only a single pass through the hidden surface removal program from the viewpoint of each light source. At present, the general polygon shadow generation procedure is only limited by the requirement of a polygonal planar data base. It has proven to be flexible, device independent, and has run efficiently on a large variety of environments.

For each display frame that is to be produced, there is a set of transformation matrices which are used to manipulate the environment coordinate data. These transformation matrices are of two types, view matrices and shadow matrices. The view matrices transform the environment to any selected view. There are two shadow matrices which are devoted to the creation of a shadow data base consisting of the original polygonal coordinate definitions and their associated lighted detail polygons (Figures 4 and 5).

The first of these shadow matrices is used to transform the entire object environment to the viewpoint of the light source. A copy of the transformed environment is made for later use. Hidden surfaces are removed from the object environment leaving only the illuminated polygons. The second shadow matrix is then used to transform the entire copy of the object environment to any environment orientation including the original orientation. The lighted polygons are also transformed by the second shadow matrix and then added to copied polygonal data as lighted details to derive a shadowed coordinate data file. Once the shadowed data file is created, only one view matrix is needed to transform it to any desirable viewing position.

## V. FITTING THE ENVIRONMENT INTO THE VIEWBOX

In performing the computations for the display of shadows, it is important that the object be entirely contained by the frustrum of vision emanating from the light source. Areas of the object that exist outside of the viewing area will be clipped and removed, and thus falsely interpreted to be in shadow. Therefore, the entire object must be within the boundaries of the viewing area.

By performing shadow calculations in object space with the polygon clipping method of hidden surface removal, the precision may be extended to the machine limits, rather than the display limits. If the coordinate values are stored in integer format, the maximum accuracy of the shadow calculations can be obtained when the following three criteria are met:

1) The boundaries of the viewbox of the frustrum of vision are set to correspond to the maximum machine limits (e.g., for a 16 bit computer, this corresponds to $\pm$32,767).
2) The object environment is centered within the viewbox.
3) The object environment is then scaled as large as possible to fit within the viewbox.

To accomplish this, the extreme three-dimensional coordinates of the original object are used to form the minimum rectangular solid containing the entire environment. The centroid of this volume is then centered in the viewbox and scaled as large as possible with the constraint that all portions of the bounding volume remain within the viewbox window.

## VI. DISPLAY OF SHADOWED IMAGES

Since the three-dimensional polygon coordinate data is maintained to the limits of machine precision, images created by the hidden surface removal system can be displayed accurately on many different peripheral devices. The two basic types of displays used are hidden line removed vector displays and hidden surface removed halftone displays (Figure 4).

The vector displays are only concerned with drawing the lines or borders of each polygon and are inherently faster than the halftone displays. Furthermore, the display is more accurate due to the available resolution of the standard vector displays. Details may be visualized easily, but the depth perception is not nearly as effective as with the halftone displays.

For color raster displays, all visible surfaces of the environment must be rendered. This is achieved with the aid of a set of software routines which can render an arbitrary concave polygon with holes with a selected color. Colors and shades can be interactively selected or automatically computed for each polygon surface. For black and white images, the shade of gray selected for a particular polygon is dependent upon the angle between a ray extending from the light source to the polygon and the normal of the polygon. Strictly speaking, shadowed surfaces would be rendered black. To aid in image visualization, the shadowed surfaces utilize a darker gray range of the gray scale than that used for the lighted surfaces. To produce a color image, the same type of intensity scale is applied to the particular ratio of basic hues (red, green

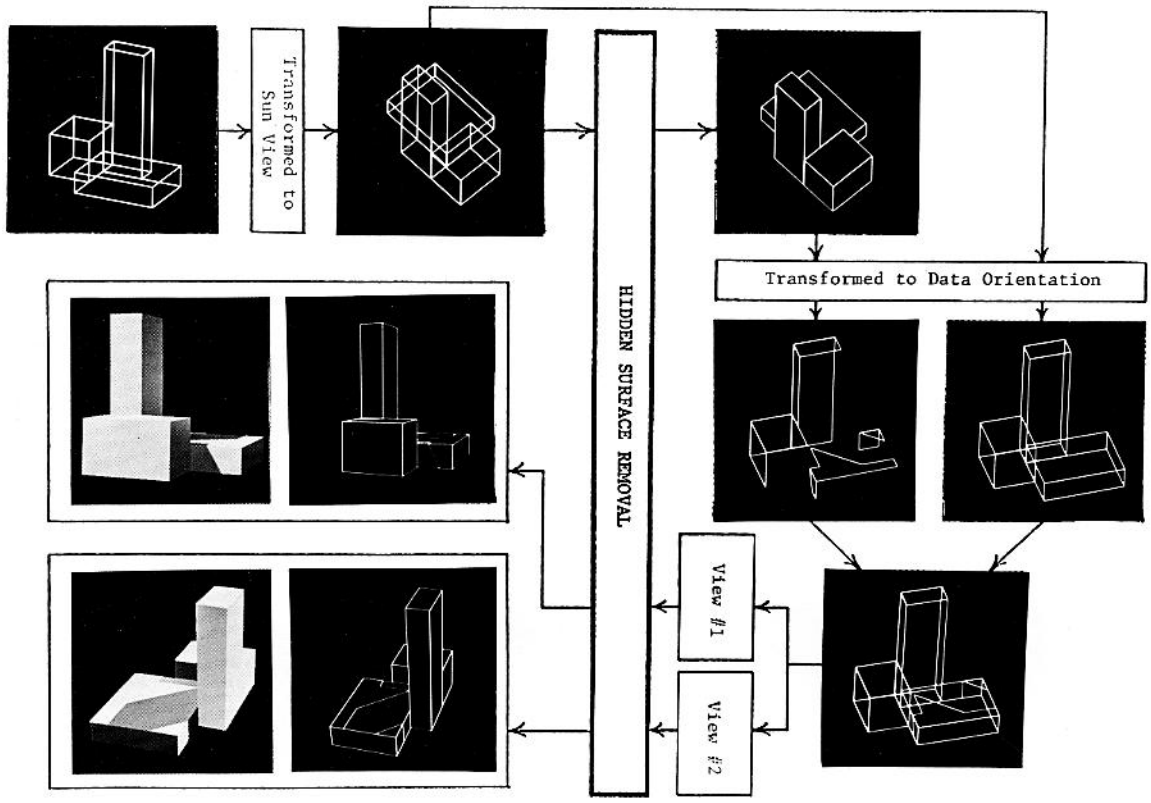



Figure 4. Shadow Creation and Display Process.

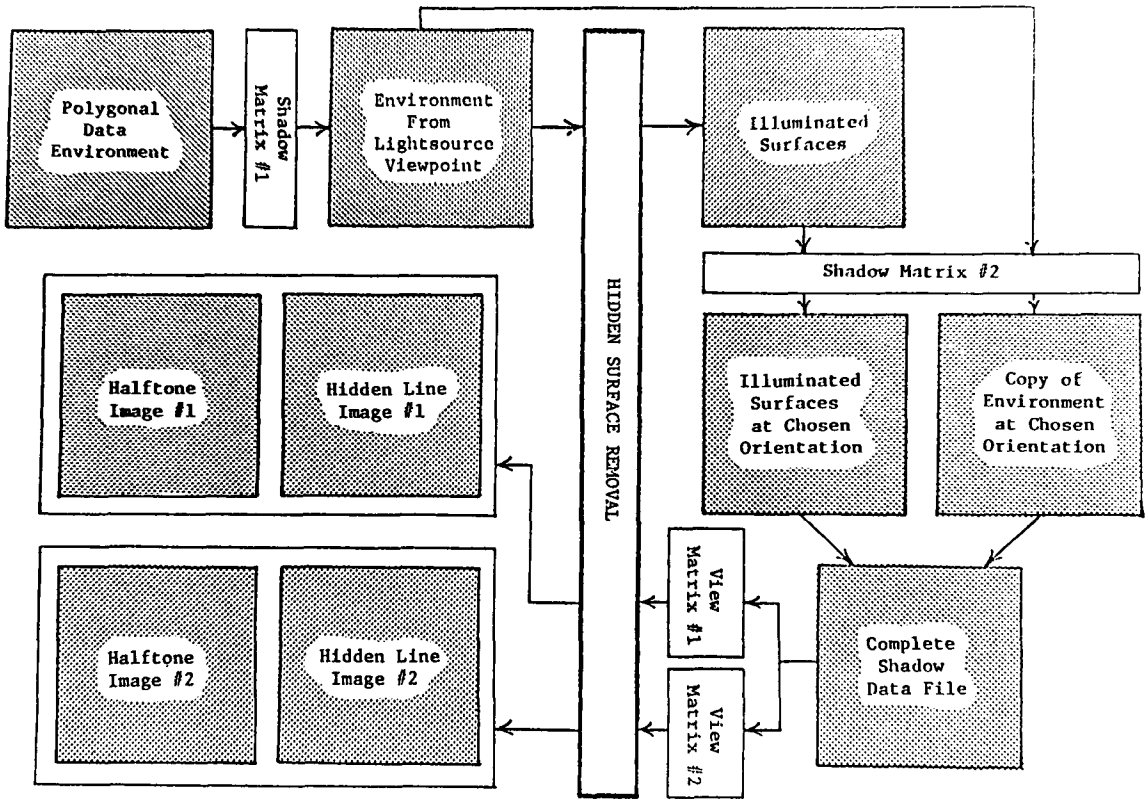
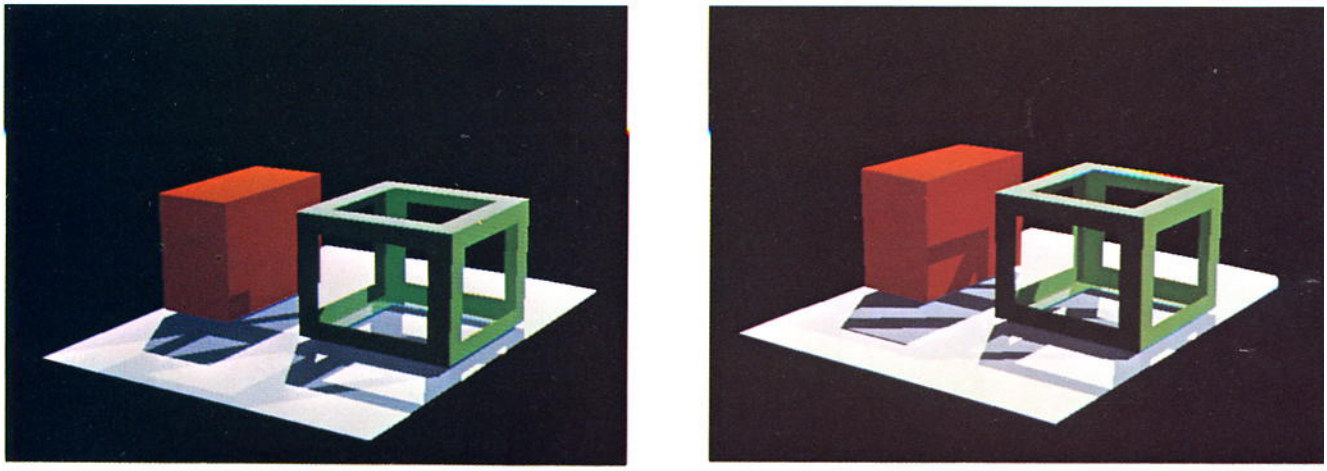
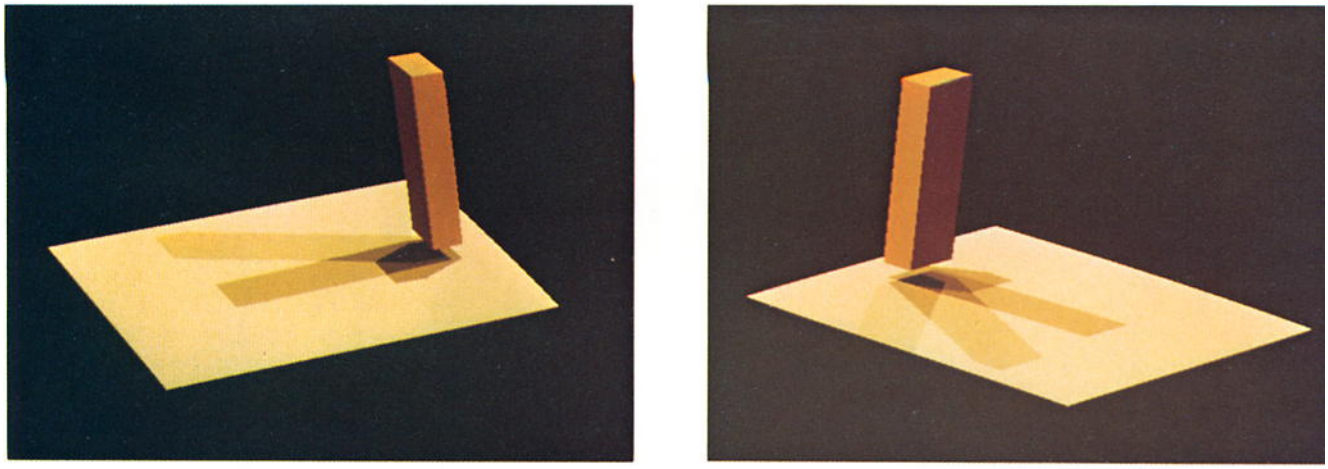


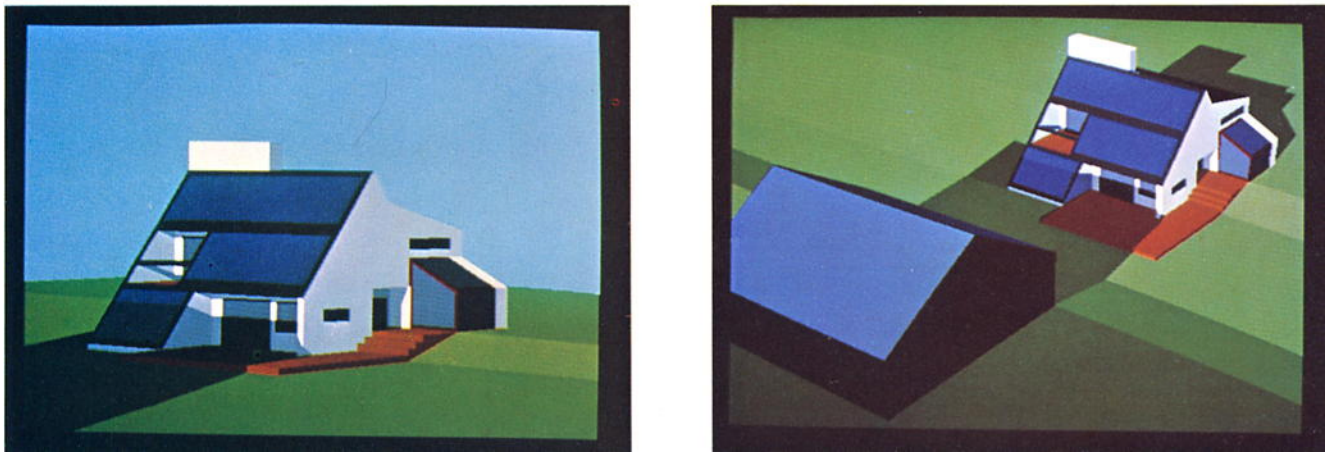

Figure 5. Shadow Creation and Display Process (key).



Figures 6. Shadowed Image Displays with Two Light Sources at Different Locations.



Figures 7. Shadowed Image Displays with Three and Four Distinct Light Sources.



Figures 8. Visual Examination of Simulated Shadowed Site from Two Observer Positions.

surface. Attempts to do this were not very sucessful. The images usually looked like smooth surfaces with photographs of wrinkles glued on. The main reason for this is that the light source direction when making the texture photograph was rarely the same as that used when synthesizing the image. In fact, if the surface (and thus the mapped texture pattern) is curved, the angle of the light source vector with the surface is not even the same at different locations on the patch.

## 2. NORMAL VECTOR PERTURBATION

To best generate images of macroscopic surface wrinkles and irregularities we must actually model them as such. Modelling each surface wrinkle as a separate patch would probably be prohibitively expensive. We are saved from this fate by the realization that the effect of wrinkles on the perceived intensity is primarily due to their effect on the direction of the surface normal (and thus the light reflected) rather than their effect on the position of the surface. We can expect, therefore, to get a good effect from having a texturing function which performs a small perturbation on the direction of the surface normal before using it in the intensity formula. This is similar to the technique used by Batson et al. [1] to synthesize aerial picutres of mountain ranges from topographic data.

The normal vector perturbation is defined in terms of a function which gives the displacement of the irregular surface from the ideal smooth one. We will call this function F(u,v). On the wrinkled patch the position of a point is displaced in the direction of the surface normal by an amount equal to the value of F(u,v). The new position vector can then be written as:

$$\vec{P}' = \vec{P} + F\,\vec{N}/|N|$$

This is shown in cross section in figure 2.

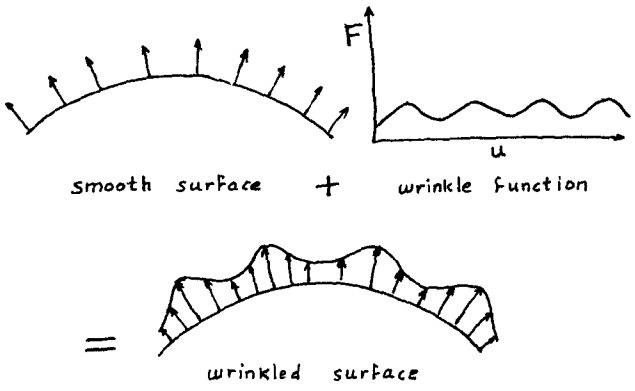



Figure 2 - Mapping Bump Function

The normal vector to this new surface is derived by taking the cross product of its partial derivatives.

$$\vec{N}' = \vec{P}u' \times \vec{P}v'$$

The partial derivatives involved are evaluated by the chain rule. So

$$\begin{aligned}\vec{P}u' &= d/du\ \vec{P}' = d/du(\vec{P} + F\,\vec{N}/|N|)\\ &= \vec{P}u + Fu\,\vec{N}/|N| + F\,(\vec{N}/|N|)u\end{aligned}$$

$$\begin{aligned}\vec{P}v' &= d/dv\ \vec{P}' = d/dv(\vec{P} + F\,\vec{N}/|N|)\\ &= \vec{P}v + Fv\,\vec{N}/|N| + F\,(\vec{N}/|N|)v\end{aligned}$$

The formulation of the normal to the wrinkled surface is now in terms of the original surface definition functions, their derivatives, and the bump function, F, and its derivatives. It is, however, rather complicated. We can simplify matters considerably by invoking the approximation that the value of F is negligably small. This is reasonable for the types of surface irregularities for which this process is intended where the height of the wrinkles in a surface is small compared to the extent of the surface. With this simplification we have

$$\vec{P}u' \approx \vec{P}u + Fu\,\vec{N}/|N|$$

$$\vec{P}v' \approx \vec{P}v + Fv\,\vec{N}/|N|$$

The new normal is then

$$\vec{N}' = (\vec{P}u + Fu\,\vec{N}/|N|) \times (\vec{P}v + Fv\,\vec{N}/|N|)$$

$$= (\vec{P}u \times \vec{P}v) + Fu\,(\vec{N} \times \vec{P}v)/|N|$$

$$+ Fv\,(\vec{P}u \times \vec{N})/|N| + Fu\,Fv\,(\vec{N}\times\vec{N})/|N|^2$$

The first term of this is, by definition, N. The last term is identically zero. The net expression for the perturbed normal vector is then

$$\vec{N}' = \vec{N} + \vec{D}$$

$$\text{where } \vec{D} = (\,Fu\,(\vec{N} \times \vec{P}v) - Fv\,(\vec{N} \times \vec{P}u)\,) \,/\, |N|$$

This can be interpreted geometrically by observing that (N x Pv) and (N x Pu) are two vectors in the tangent plane to the surface. An amount of each of them proportional to the u and v derivatives of F are added to the original, unperturbed normal vector. See figure 3

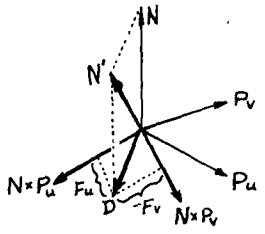



Figure 3 - Perturbed Normal Vector

Another geometric interpretation is that the vector N' comes from rotating the original vector N about some axis in the tangent plane to the surface. This axis vector can be found as the cross product of N and N'.

$$\vec{N} \times \vec{N}' = \vec{N} \times (\vec{N}+\vec{D}) = \vec{N} \times \vec{D}$$

$$= \frac{Fu\ (\vec{N} \times (\vec{N} \times \vec{Pv})) - Fv\ (\vec{N} \times (\vec{N} \times \vec{Pu}))}{|N|}$$

Invoking the vector identity Qx(RxS) = R(Q.S) - S(Q.R) and the fact that N.Pu = N.Pv = 0 this axis of rotation reduces to

$$\vec{N}x\vec{N}' = |N|(Fv\ \vec{Pu} - Fu\ \vec{Pv}) \equiv |N|\ \vec{A}$$

This vector, A, is just the perpendicular to the gradient vector of F, (Fu,Fv) when expressed in the tangent plane coordinate system with basis vectors Pu and Pv. Thus the perturbed normal vector will be tipped "downhill" from the slope due to F. Note that, since NxD=|N| A and since N is perpendicular to D then

$$|NxD| = |N|\ |D|$$

so

$$|D| = |A|$$

Next, since the vectors N, D and N' form a right triangle, the effective angle of rotation is

$$\tan\vartheta = |D|/|N|$$

this is illustrated in figure 4.

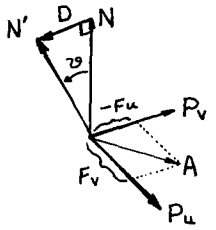



Figure 4 - Rotated Normal Vector

In summary, we can now calculate the perturbed normal vector, N', at any desired u and v parameter value. This vector must still be scaled to a length of 1 by dividing by its length. The result is then passed to the intensity calculation routines in place of the actual normal N.

## 3. TEXTURE FUNCTION DEFINITION

The formulation of the perturbed normal vector is in terms of the position functions X, Y, and Z and the bump displacement function F. To perform calculations we only need a means of evaluating the u and v derivatives of F(u,v) at any required parameter value. In this section we discuss some ways that such functions have been defined, means of evaluating them and show some resultant pictures.

The function F could, of course, be defined analytically as a bivariate polynomial or bivariate Fourier series. In order to generate a function with a sufficient amount of complexity to be interesting, however, an excessive number of coefficients are required. A much simpler way to define complex functions is by a table lookup. Since F has two parameters, this table takes the form of a doubly indexed array of values of F at various fractional parameter values. If the array is 64 by 64 elements and the parameters are between 0 and 1 a simple means of evaluating F (using Fortran style indexing) at u and v would be

```
FUNCTION FVAL(U,V)
    IU = IFIX(64*U)
    IV = IFIX(64*V)
    FVAL = FARRAY(IU+1,IV+1)
```

(We will duscuss the problem of overflow of the indices shortly). This will yield a function made of a checkerboard of constant valued squares 1/64 on a side. A smoother function can be obtained by interpolating values between table entries. The simplest interpolation technique is bilinear interpolation. Such an algorithm would look like

```
FUNCTION FVAL(U,V)
    IU=IFIX(64*U)
    DU=64*U - IU
    IV=IFIX(64*V)
    DV=64*V - IV
    F00 = FARRAY(IU+1,IV+1)
    F10 = FARRAY(IU+2,IV+1)
    F01 = FARRAY(IU+1,IV+2)
    F11 = FARRAY(IU+2,IV+2)
    FU0 = F00 + DU*(F10-F00)
    FU1 = F01 + DU*(F11-F01)
    FVAL= FU0 + DV*(FU1-FU0)
```

This yields a function which is continuous in value but discontinuous in derivative. Since the function F appears in the calculation only in terms of its derivative we should use a higher order interpolation scheme which is continuous in derivative. Otherwise the lines between function samples may show up as creases in the surface. Third order interpolation schemes (e.g. B-splines) are the standard solution to such a situation, but their generality is not really needed here. A cheaper, continuous interpolation scheme for derivatives consists of differencing the (bilinearly interpolated) function along the parametric directions. The increment between which differencing occurs is the distance between function sample values. The function generated by this interpolation scheme has continuity of derivative but not of value. The values of F are not used anyway. Thus

```
E = 1/64.
FU = (FVAL(U+E,V )-FVAL(U-E,V )) / (2*E)
FV = (FVAL(U ,V+E)-FVAL(U ,V-E)) / (2*E)
```

This is the form used in the pictures shown here. It is about as simple as can be obtained and has proven to be quite adequate.

In the above examples, the integer part of the scaled up parameter values were used directly as indices into the F array. In practive, one should protect against array overflow occurring when the parameter happens to be slightly less than 0 or greater than 1. In fact, for the bilinear interpolation case, all parameter values between 63/64 and 1 will attempt to interpolate to a table entry at index 65. The question of what is the function value at parameters outside the range of the table can be answered in a variety of ways. A simple method is to make the function periodic, with the table defining one period.

This is easily accomplished by masking off all but the low 6 bits of the IU and IV values. This also makes it easy to have the table represent a unit cell pattern to be replicated many times per patch. The function values U and V are merely scaled up by the replication count before being passed to FVAL.

Now that we know what to do with the table entries we turn to the question of how to generate them in the first place. Some simple geometric patterns can be generated algorithmically. One such is a gridwork of high and low values. The table entries of the F function for such a grid are shown plotted as a 3D line drawing in figure 5. The result when mapped onto a flat patch with one corner bent back is also shown.

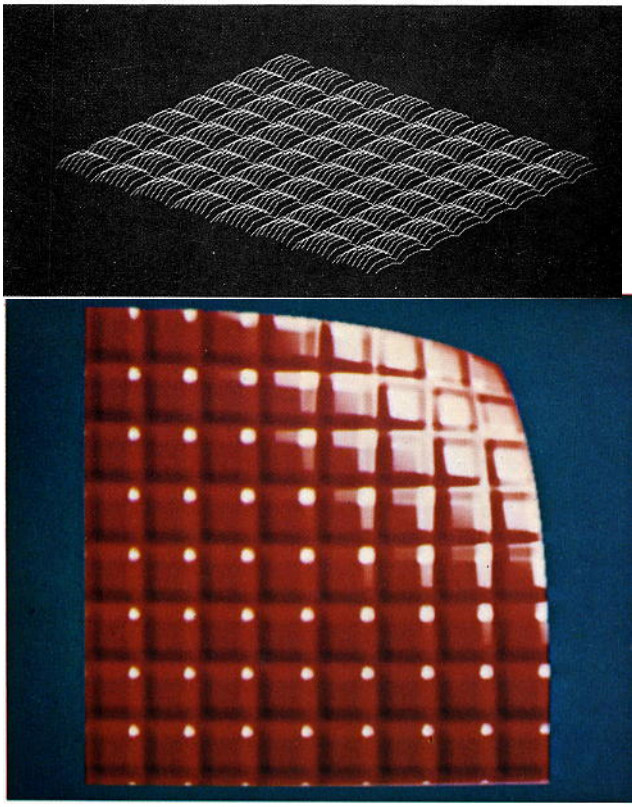


Figure 5 - Simple Grid Pattern

Embossed letters can be generated by using a bit-map character set as used to display text on a raster scan display. Such a texture array appears in figure 6. This pattern was used to make the title on the ribbon on the logo of the cover of these proceedings.

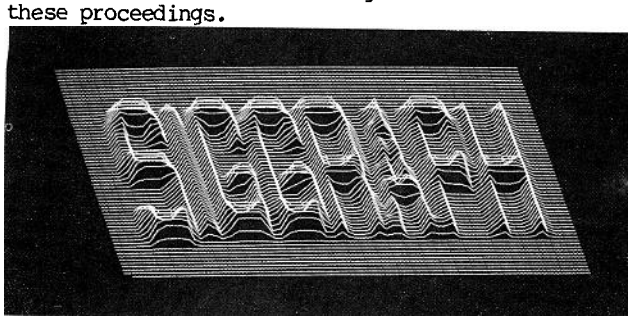


Figure 6 - Embossed Letter Pattern

Another method of generating bump functions derives from image synthesis algorithms which use Z-buffers or depth buffers to perform the hidden surface comparisons [5]. The actual Z values left in the depth buffer after running such an algorithm can be used to define the table entries for a bump function. In figure 7 an image of a sphere was generated using such an algorithm and the resultant Z-buffer replicated several times to generate the rivet-like pattern. This is the pattern mapped onto the cube on the cover logo. Similarly, a 3D character set was used with a Z-buffer algorithm to generate the pattern showing the date also in figure 7. This was used on the ribbon on the cover.

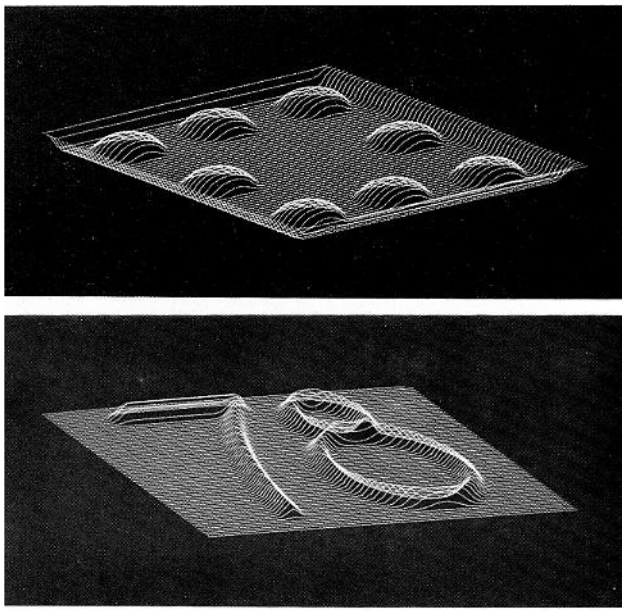
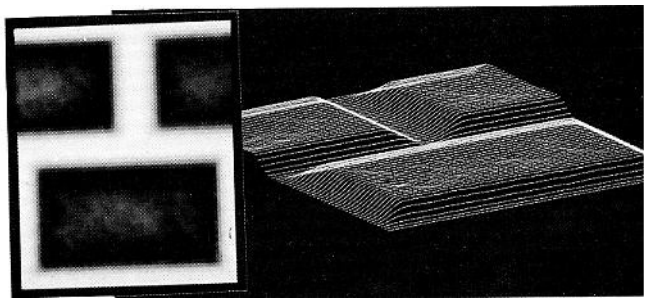
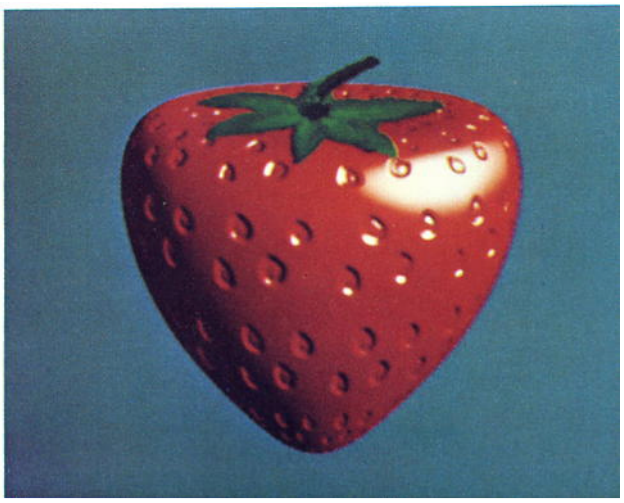


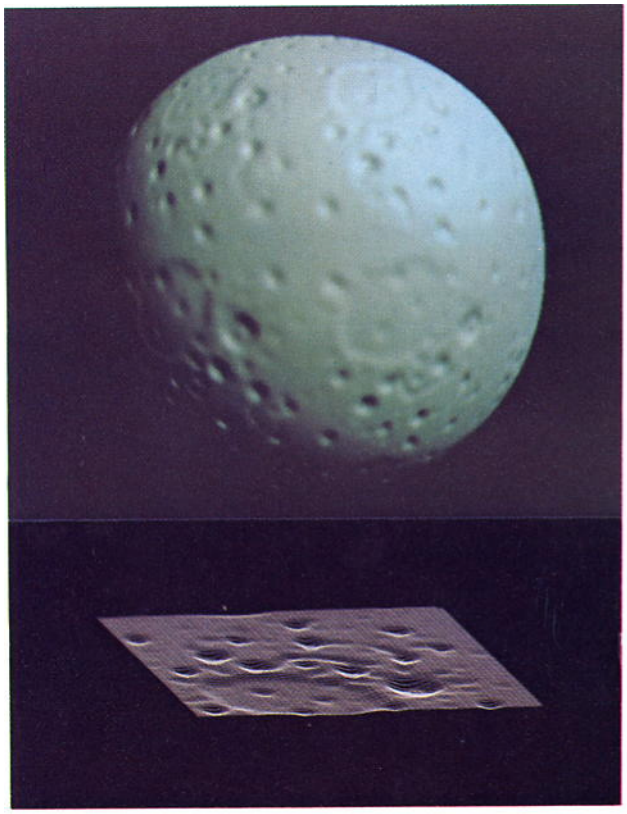
Figure 7 - Z-Buffer Patterns

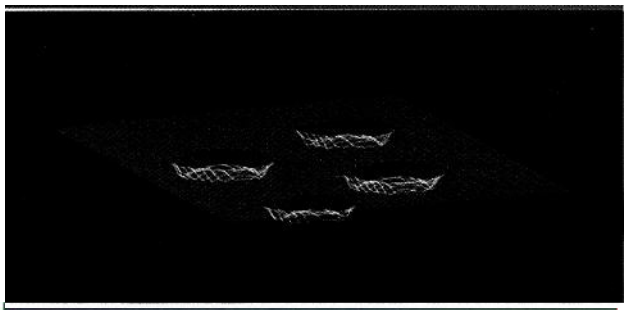
The most general method of generating bump functions relies on video frame buffer technology and its standard tool, the painting program. Briefly, a frame buffer is a large digital memory with one word per picture element of an image. A video signal is continually synthesized from this memory so that the screen displays an image of what is in memory. A painting program utilizes a digitizing tablet to control the alteration of the values in the memory to achieve the effect of painting on the screen. By utilizing a region of the frame buffer as the defining table of the F function, a user can actually paint in the function values. The interpretation of the image will be such that black areas produce small values of F and white areas produce large values. Since only the derivatives of F are used in the normal vector perturbation, any area of constant intensity will look smooth on the final image. However, places where the image becomes darker will appear as dents and places where it becomes brighter will appear as bumps. (This correspondance will be reversed if the base patch is rotated to view the back side). The generation of interesting patterns which fit together end-to-end to form a continuous join between patches then becomes primarily an artistic effort on the part of the drawer. Figure 8 shows some

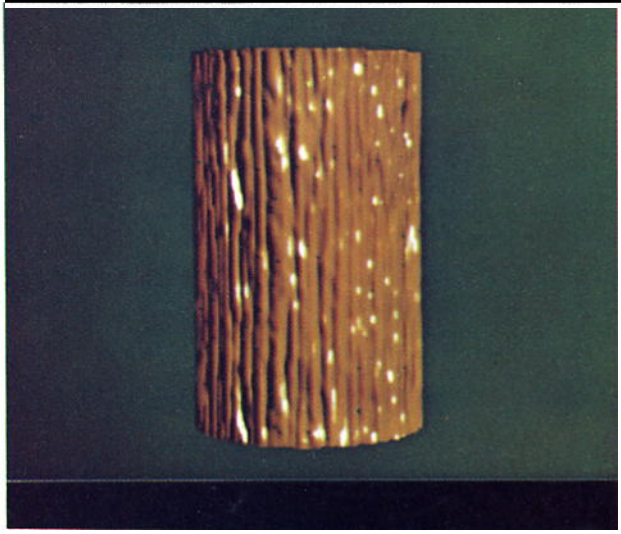
sample results that can be achieved with this technique. The first pattern, a hand drawn unit cell of bricks was mapped onto the sphere on the cover.

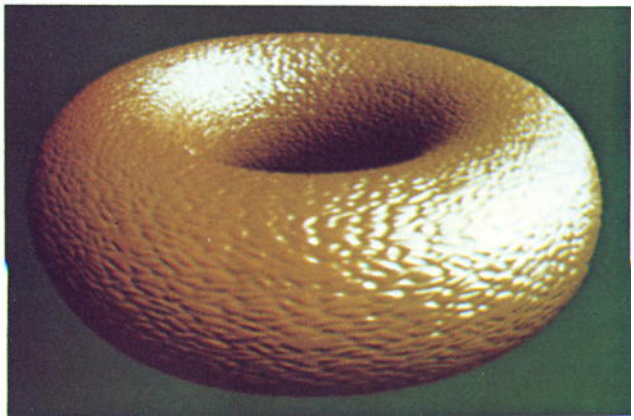












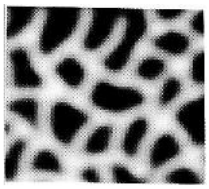


Figure 8 - Hand Drawn Bump Functions

## 4. DEPENDANCE ON SCALE

One feature of the perturbation calculation is that the perturbation amount is not invariant with the scale at which the object is drawn. If the X, Y, and Z surface definiton functions are scaled up by 2 then the normal vector length, |N|, is scaled up by a factor of 4 while the perturbation amount, |D|, is only scaled by 2. This effect is due to the fact that the object is being scaled but the displacement function F is not. (Scale changes due to the object moving nearer or farther from the viewer in perspective space do not affect the size of the wrinkles, only scale shanges applied directly to the object.) The net effect of this is that if an object is scaled up, the wrinkles flatten out. This is illustrated in figure 9.


Figure 9 - Stretched Bump Texture

This effect might be desirable for some applications but undesirable for others. A scale invariant perturbation, D', must scale at the same rate as N. An obvious choice for this is

$$D' = a\ D\ |N|/|D|$$

so

$$|D'| = a\ |N|$$

where a is independant of scales in P. The value of a is then the tangent of the effective rotation angle.

$$\tan\vartheta' = |D'|/|N| = a$$

This can be defined in various ways. One simple choice is a generalization from the simple, flat unit square patch

$$X(u,v) = u$$
$$Y(u,v) = v$$
$$Z(u,v) = 0$$

For this patch the original normal vector perturbation gives

$$N = (0,0,1)$$
$$D = (-Fu,-Fv,0)$$
$$\tan\vartheta = \mathrm{sqrt}(Fu^2+Fv^2)$$

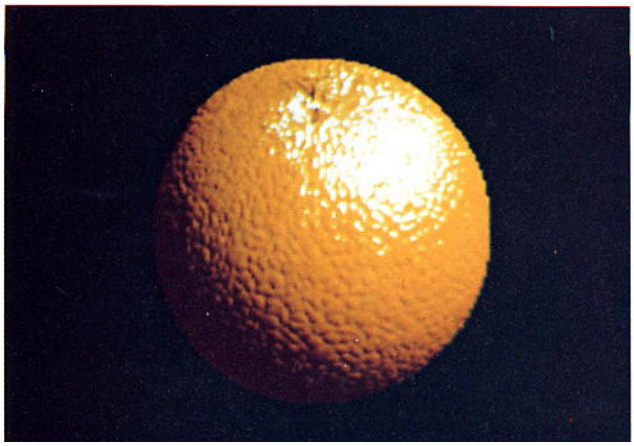
Here the value of a is purely a function of F. Use of the same function for arbitrary patches corresponds to a perturbation of

$$a = \mathrm{sqrt}(Fu^2+Fv^2)$$
$$D' = a\ D\ |N|/|D|$$
$$N'' = N + D'$$

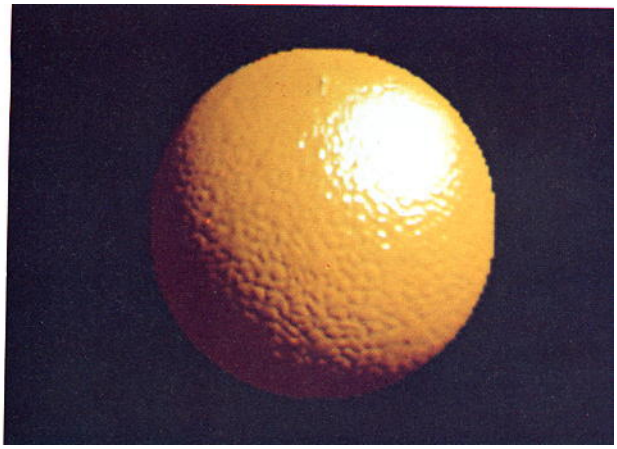
The texture defining function F is now no longer being used as an actual displacement added to the position of the surface. It just serves to provide (in the form if its derivatives) a means of defining the rotation axis and angle as functions of u and v.

## 5. ALIASING

In an earlier paper [2], the author described the effect of aliasing on images made with color texture mapping. The same problems can arise with this new form. That is, undesirable artifacts can enter the image in regions where the texture pattern maps into a small screen region. The solution applied to color textures was to average the texture pattern over the region corresponding to each picture element in the final image. The bump texture definition function, however, does not have a linear relationship to the intensity of the final image. If the bump texture is averaged the effect will be to smooth out the bumps rather than average the intensities. The correct solution to this problem would be to compute the intensities at some high sub-pixel resolution and average them. Simply filtering the bump function can, however, reduce the more offensive artifacts of aliasing. Figure 10 shows the result of such an operation.



Before



After

Figure 10 - Filtering Bump Texture

## 6.RESULTS

Surfaces appearing in images made with this technique look quite convincingly wrinkled. An especially nice effect is the interaction of the bumps with calculated highlights. We must realize, however, that the wrinkles are purely illusory. They only come from some playing with the parameters used in intensity calculations. They do not, for example, alter the smooth silhouette edges of the object. A useful test of any image generation algorithm is to see how well the objects look as they move in animation sequences. Some sample frames from such an animation sequence appear in figure 11. The illusion of wrinkles continues to be convincing and the smoothness of the silhouette edges is not overly bothersome.

Some simple timing measurements indicate that bump mapping takes about 4 times as long as Phong shading and about 2 times as long as color texture mapping. The pictures in this paper took from 3 to 7 minutes each to produce.

The author would like to thank Lance Williams and the New York Institute of Technology Computer Graphics Laboratory for providing some of the artwork and assistance in preparing the logo on the cover made with the techniques described in this paper.

## REFERENCES

[1] Batson, R. M., Edwards, E. and Eliason, E. M. "Computer Generated Shaded Relief Images", Jour, Research U.S. Geol. Survey, Vol. 3, No. 4, July-Aug 1975, p. 401-408.

[2] Blinn, J. F., and Newell, M. E., "Texture and Reflection in Computer Generated Images", CACM 19, 10, Oct 1976, pp 542-547.

[3] Blinn, J. F., "Models of Light Reflection for Computer Synthesized Pictures", Proc. 4th Conference on Computer Graphics and Interactive Techniques, 1977.

[4] Blinn, J. F., "A Scan Line Algorithm for Displaying Parametrically Defined Surfaces", Proc. 5th Conference on Computer Graphics and Interactive Techniques, 1978.

[5] Catmull, E. E., "Computer Display of Curved Surfaces", Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures, Los Angeles (May 1975)11.

[6] Whitted, J. T., "A Scan Line Algorithm for Computer Display of Curved Surfaces", Proc. 5th Conference on Computer Graphics ond Interactive Techniques, 1978.

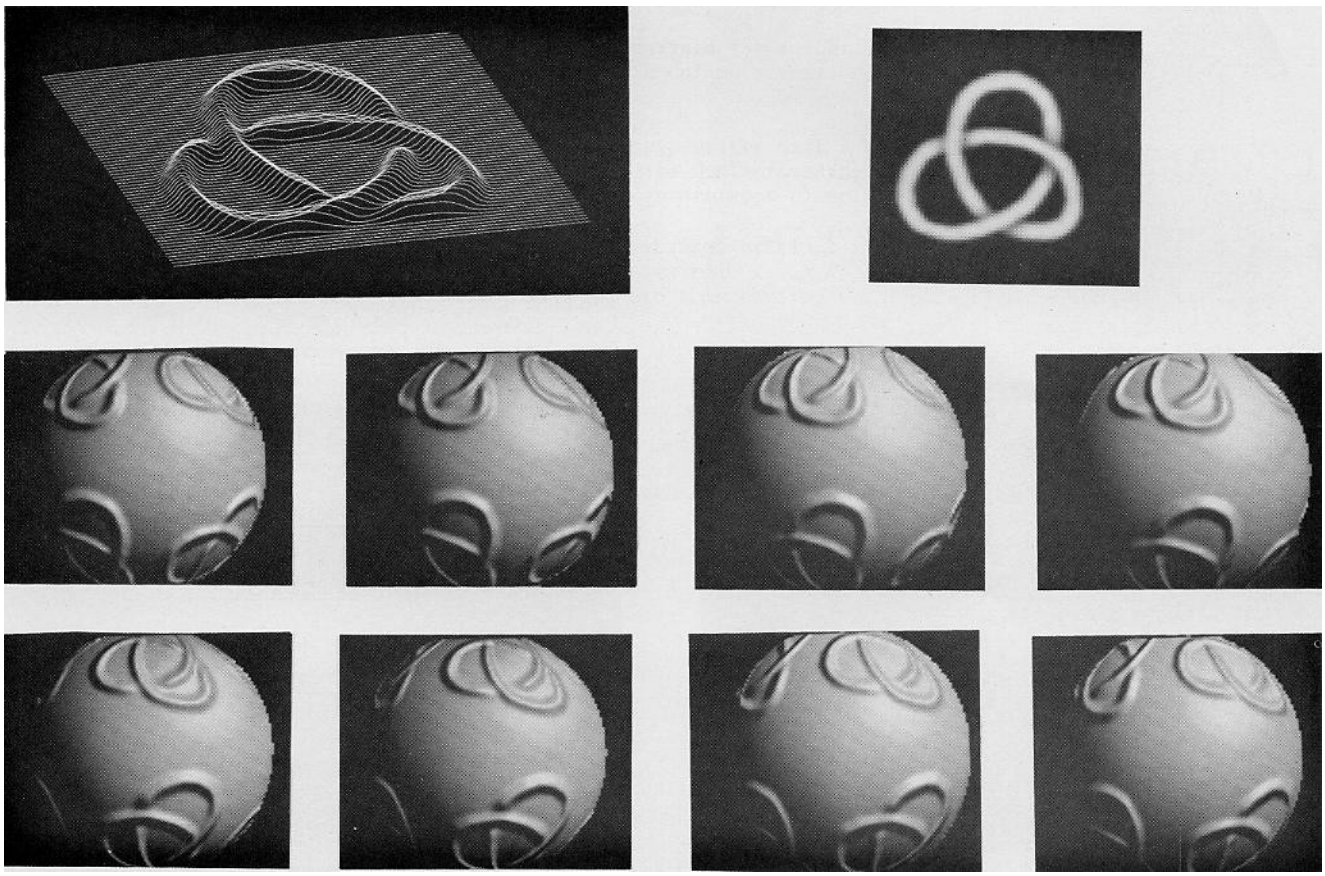


Figure 11 - Rotating Textured Sphere