SIMULATION OF WRINKLED SURFACES

James F. Blinn
Caltech/JPL

Abstract

Computer generated shaded images have reached an impressive degree of realism with the current state of the art. They are not so realistic, however, that they would fool many people into believing they are real. One problem is that the surfaces tend to look artificial due to their extreme smoothness. What is needed is a means of simulating the surface irregularities that are on real surfaces. In 1973 Ed Catmull introduced the idea of using the parameter values of parametricallydefined surfaces to index into a texture definition function which scales the intensity of the reflected light. By tying the texture pattern to the parameter values, the texture is guaranteed to rotate and move with the object. This is good for showing patterns painted on the surface, but attempts to simulate rough surfaces in this way are unconvincing. This paper presents a method of using a texturing function to perform a small perturbation on the direction of the surface normal before using it in the intensity calculations. This process yields images with realistic looking surface wrinkles without the need to model each wrinkle as a separate surface element. Several samples of images made with this technique are included.

1. INTRODUCTION

Recent work in computer graphics has been devoted to the development of algorithms for making pictures of objects modelled by other than the conventional polygonal facet technique. In particular, several algorithms [4,5,7] have been devised for making images of parametric surface patches. Such surfaces are defined by the values of three bivariate functions:

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

as the parameters vary between 0 and 1. Such algorithms basically consist of techniques for inverting the X and Y functions. That is, given the X and Y of a picture element, the corresponding u and v parameter values are found. This parameter pair is then used to find the Z coordinate of the surface to perform depth comparisons with other objects. The intensity of the resultant picture element is then found by a simulation of the light reflecting off the surface. Functions for performing this computation are described in [3].

The prime component in the calculation of the intensity of a picture element is the direction of the surface normal at that picture element. To calculate the surface normal we first examine the derivatives of the surface definition functions. If the coordinates of a point on the patch is represented by the vector P:

$$P = (X,Y,Z)$$

The partial derivatives of these functions form two new vectors which we will call Pu and Pv.

$$Pu = (Xu,Yu,Zu)$$

$$Pv = (Xv,Yv,Zv)$$

These two vectors define a plane tangent to the surface at that point. Their cross product is thus a vector normal to the surface.

$$N = Pu \times Pv$$

These vectors are illustrated in figure 1. Before using the normal in intensity calculations it must first be scaled to a length of 1.0 by dividing by its length.

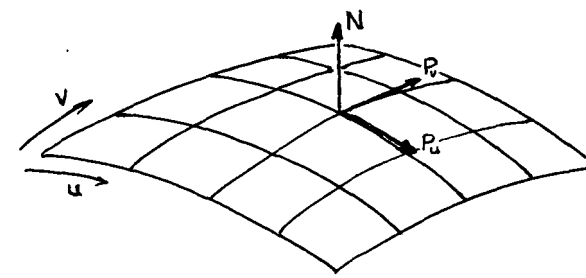


Figure 1 - Definition of Normal Vector

Images of smooth surfaces made directly from the patch description do not have the usual artifacts associated with polygonal facets, they do indeed look smooth. In fact they sometimes look too smooth. To make them look less artificial it is necessary to simulate some of the surface irregularities of real surfaces. Catmull [5] made some progress in this direction with process called texture mapping. Effectively the color of the surface was defined as a fourth bivariate function, C(u,v), and was used to scale the intensity of the generated picture at each point. This technique was good a generating pictures of objects with patterns painted on them. In order to simulate bumpy or wrinkly surfaces one might use, as the defining texture pattern, a digitized photograph of a bumpy or wrinkly

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:

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

This is shown in cross section in figure 2.

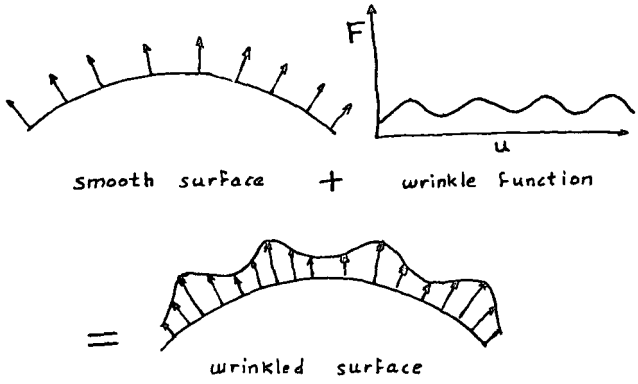


Figure 2 - Mapping Bump Function

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

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

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

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

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

$$Pv' \quad Pv + Fv\,N/|N|$$

The new normal is then

$$N' = (Pu + Fu\,N/NI) \times (Pv + Fv\,N/NI)$$

$$= (Pu \times Pv) + Fu\,(N \times Pv)/|N|$$

$$+ Fv\,(Pu \times \vec{N})/|N| + Fu\,Fv\,(\vec{N} \times \vec{N})/|N|$$

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

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

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

This can be interpreted geometrically by observing that $(N \times Pv)$ and $(N \times Pu)$ 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

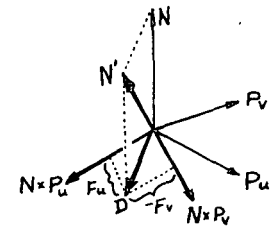


Figure 3 - Perturbed Normal Vector

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

$$NxN' = INI(Fv\ Pu - Fu\ Pv) = IN\ A$$

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

$$|NxD| = |N|\ IDI$$

so

$$DI = |A|$$

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

$$\tan\vartheta = IDI/INI$$

this is illustrated in figure 4.

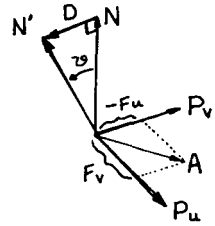


Figure 4 - Rotated Normal Vector

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 lw 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.

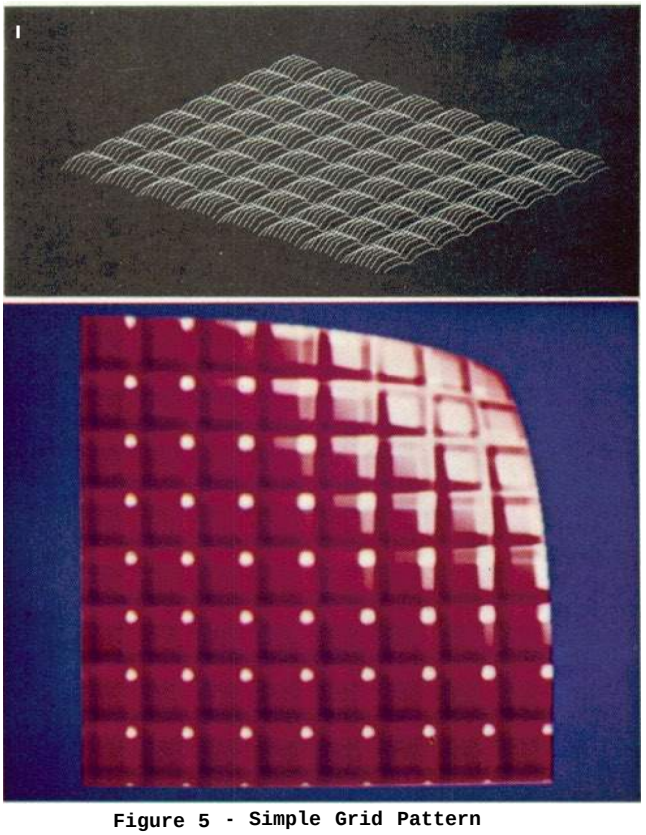


Figure 5 - Simple Grid Pattern

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.

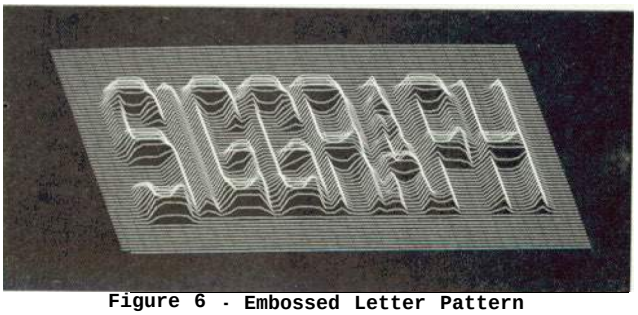


Figure 6 - Embossed Letter Pattern

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.

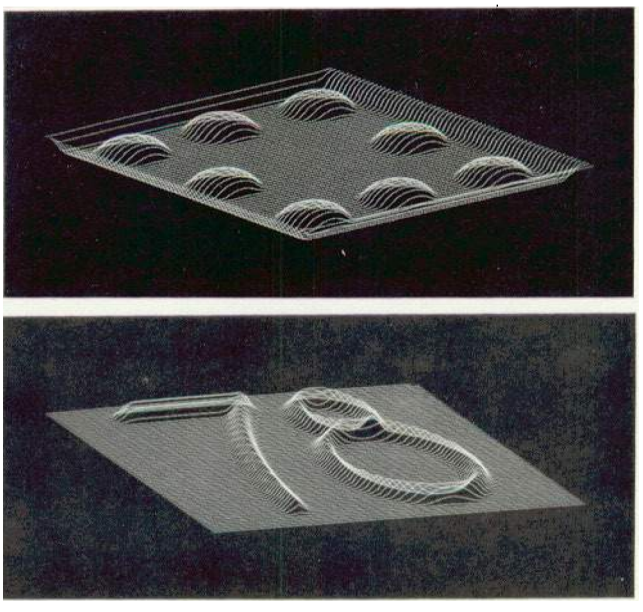
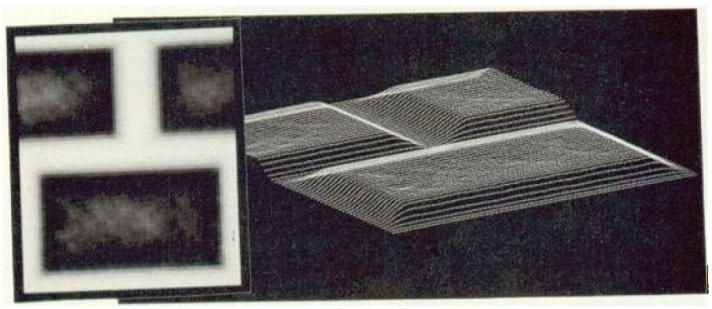
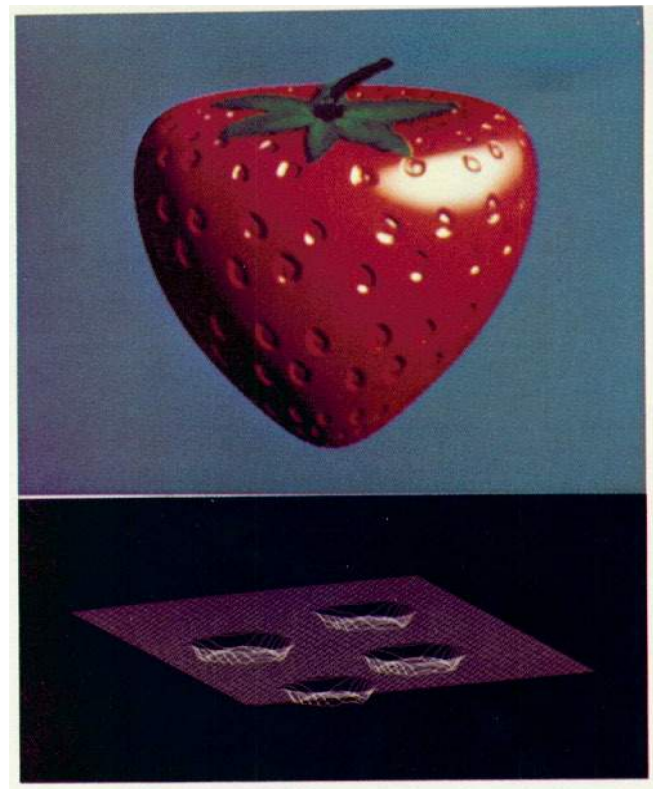


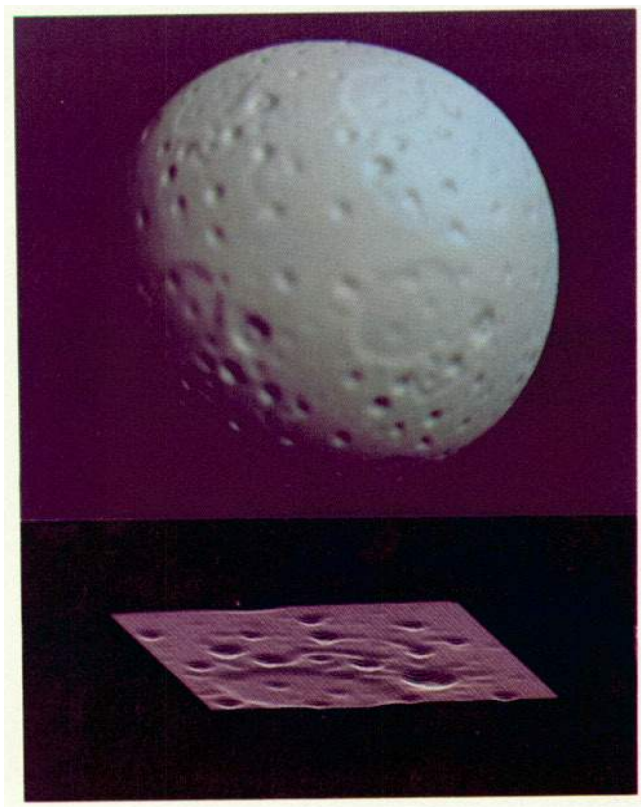
Figure 7 - Z-Buffer Patterns

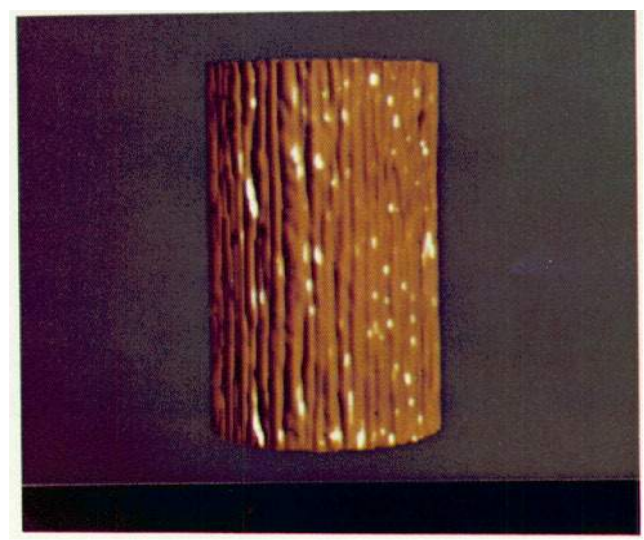
The most general method of generating bump functions relies on video frame buffer technology and its standard tool, the painting program. Briefly, a frame buffer is a large digital memory with one word per picture element of an image. A video signal is continually synthesized fran this memory so that 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.









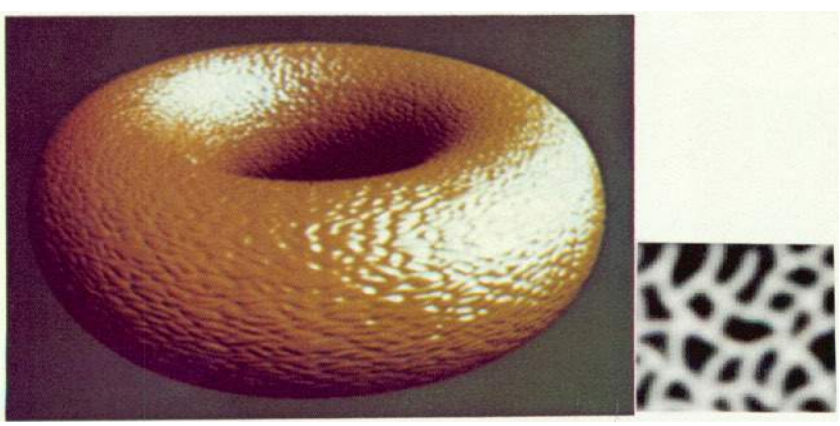


Figure A- Hand Drawn Bump Funtions

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|, 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.

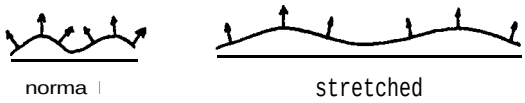


Figure 9 - stretched Bump Texture

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

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

so

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

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

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

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

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

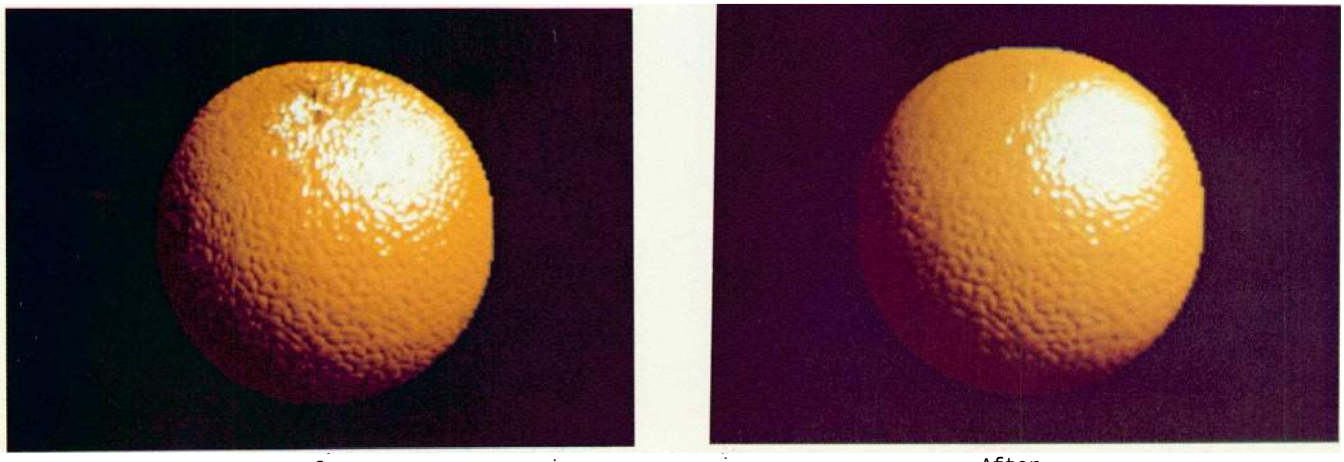
Here the value of a is purely a function of F. Use of the same function for arbitrary patches corresponds to a perturbation of

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

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 121, 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. Sane 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 1975111.

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

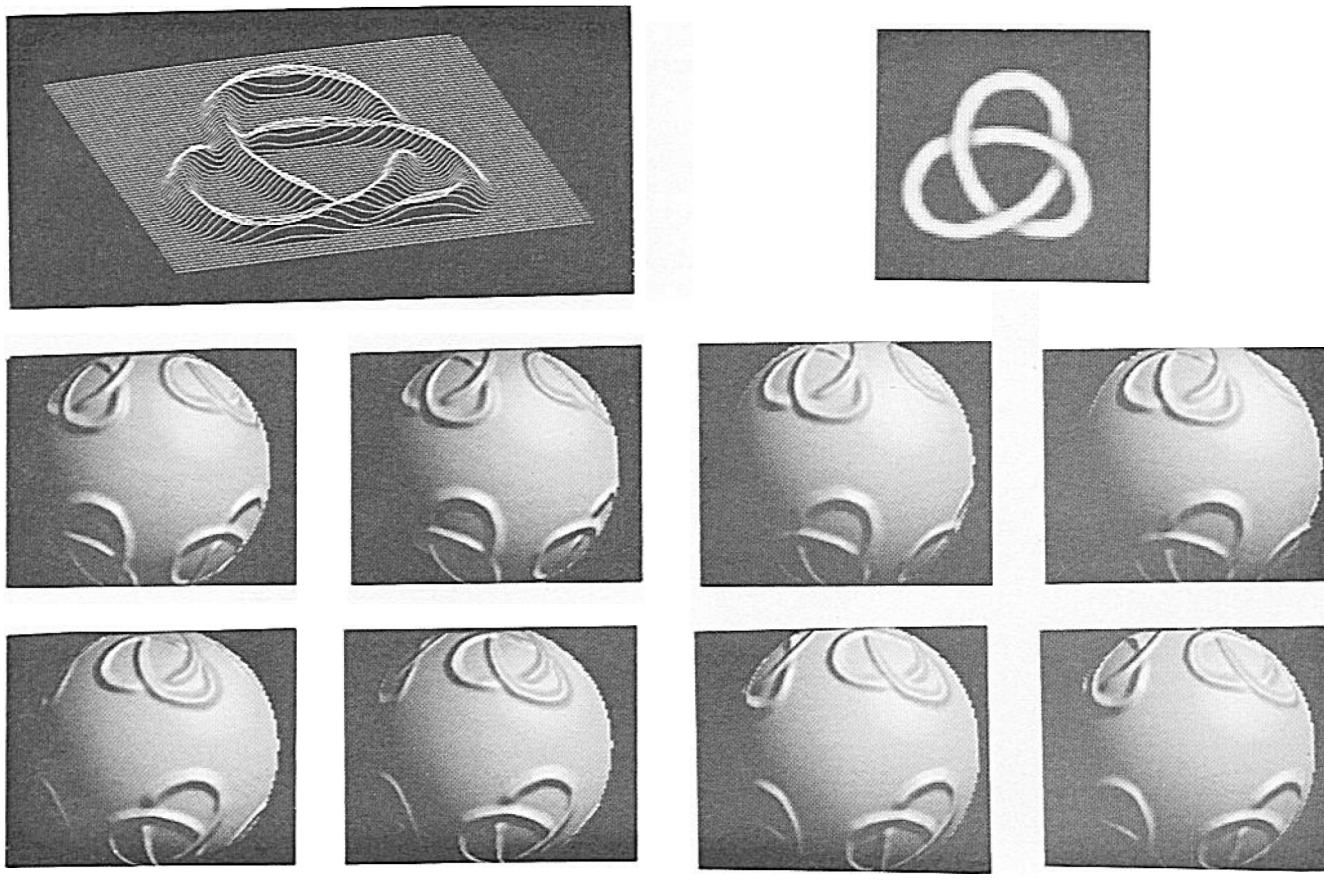


Figure 11 - Rotating Textured Sphere