

# Texture

Franklin C. Crow  
Xerox Palo Alto Research Center

Appeared in: (Earnshaw, R. and Rogers, D. Eds.)  
*Computer Graphics Techniques -Theory and Practice*,  
Springer-Verlag, 1989.

## Abstract

Texture applied to a surface adds interest and realism without complicating the underlying surface description. Texture values can be produced by space-filling functions -  $f(x,y,z)$  - or 2-dimensional functions -  $f(x,y)$ . A computed texture value can be used to modify color, reflectance properties, transparency, or other properties of a surface which determine its shading. It may further be used to perturb the shading parameters to produce the appearance of a bumpy surface, or to actually displace the surface, producing "real" bumps.

Photographic images provide a convenient source of two dimensional functions for enhancing realism through texture. This has motivated much study of methods for mapping two-dimensional textures onto surfaces. Similar applications of texture mapping allow reflections of the surrounding environment and representation of complex or multitudinous light sources. Sampling texture functions leads to aliasing problems which may be dealt with in various ways. Finally, different texturing techniques are appropriate for different levels of detail. Making smooth transitions between different forms of texture poses interesting problems.

## 1. Introduction

The progress of realism in computer-synthesized images was greatly aided by the introduction of surface texture in the mid 1970s [Catmull74, Blinn76, Blinn78]. Surface texturing techniques allowed much more interesting images to be made without having to produce more complicated underlying surface descriptions. Since then many texturing techniques have been explored and texture has come to be an expected feature in up-to-date image synthesis systems.

### 1.1 Application of Texture Values

Texture is generally applied by evaluating a texture function at each displayed pixel where a textured surface is represented. The texture value can then be used to modify the surface shading in a number of different ways. The most straightforward use of the texture value is to use it as a surface intensity ranging from zero to one. Where the texture value is one the surface maintains its color. Where it is zero, the color goes to zero. In between the surface color is darkened appropriately. Figure 1 shows an egg with a striped texture applied using this rule.

If three texture values are supplied by the texture function (e.g. a red, green, and blue texture triple) then the texture itself may provide the surface color. This is particularly appropriate when a photograph is used to provide a "natural" texture as in figure 2. Since a large percentage of man-made objects have painted or printed surfaces, such objects are often represented quite well by varying just the color across an otherwise smooth surface. Items made of metal, plastic, or cardboard generally have smooth painted or printed surfaces.

On the other hand, objects made from natural materials, such as wood, exhibit variations in reflectivity across the surface as well as variations in color. Similarly, painted or printed patterns may use materials (paints, inks) with differing reflective properties. Generally, variation in reflectivity coincides with accompanying variations in color. In shading functions capable of rendering highlights, the contribution of the highlight may be modified by a texture value before being combined with the remainder of the shading function to implement variations in reflectivity as seen in figure 3.

Transparent glass and plastic objects often exhibit variations in transparency. These variations can be produced by painting or printing patterns on the surface or by incorporating materials with different properties in the mix from which the object is molded, blown, or extruded. A texture value may be used to vary the transmittance of a surface to achieve such effects. An example may be seen in the glass shown in figure 1.

Many man-made objects exhibit low-relief texture. A surface may be made more interesting or less slippery, for example, by adding some bumpiness or roughness. In other cases, low relief texture can be an artifact of the process, as in a plastered wall, a woven carpet, or paint applied with a roller. This sort of surface is simulated well by techniques developed initially by Blinn [Blinn78]. His method uses texture values to perturb the local orientation of the surface before calculating the shading. The surface is thus made to appear bumpy without actually displacing the surface. An example is shown in Figure 4.

Where higher relief is to be represented, actual displacement of the surface is necessary. This has been accomplished in at least four ways. (1) The data describing the shape can be algorithmically expanded [Csuri79]. (2) The surface can be subdivided using stochastic techniques to displace new vertices [Fournier82, Haruyama84]. (3) The surface can be displaced using texture values during a subdivision algorithm [Hiller87]. (4) The surface can be diced into subpixel sized micropolygons which are then displaced by texture values [Cook84, Cook87].

A texture function can be used to modulate any parameter which can be varied over a surface, even to the presence or absence of the surface. Trees, for example, can be represented by a few very simple polygons with bivalued texture maps determining presence or absence of a leaf. This technique is used by Evans and Sutherland in their CT-6 visual system for training systems.

The examples above span the obvious applications of texture. However as new shading techniques arise, texturing techniques can be applied anew. We can certainly envision texture values modulating characteristics such as hairiness or index of refraction. Only our imaginations limit us.

## 1.2 Choosing Texture Values

A texture function must return a value or set of values within a given range in response to input values from a domain in 2 and sometimes 3 dimensions. For a simple example take the function  $\sin(cy)$ . This could be applied as a "solid" texture function providing objects with regular stripes. On the other hand, it

could be applied as a "surface" texture, putting stripes on each polygon or patch independently.

To apply a function as a solid texture, the surface represented in an image must be intersected with the function and evaluated at each point represented by a pixel in the image. This requires carrying the untransformed surface coordinates along through the entire transformation, clipping, and scan conversion pipeline. Alternatively, an inverse transformation can be applied to recover the original coordinates.

To apply a function as a surface texture, some mapping must be available which takes the surface of an object, or each of its "surface elements" (polygons, patches, etc.) into the domain of the texture function. Normally we refer to this style of texture use as "texture mapping". There are numerous different ways of achieving a mapping between the texture function domain and a surface, a number of which are described below.

Both of these general methods, solid texture and surface texture mapping, require evaluating (sampling) the texture function at discrete points. This gives rise to "aliasing" problems [Crow77, Crow86] whenever the points lie too far apart in the domain of the texture function. Solutions to this problem range from predistorting the texture function to better fit the distribution of evaluation points to elaborate and expensive schemes for approximating a discrete integral over the area represented by a pixel. This problem is visited many times below.

### **1.3 Texture Functions**

The texture function is a fascinating study by itself. Simple analytic functions have some use [Peachey85]. However, algorithmic functions [Perlin85] and especially image-based functions [Catmull74, Blinn78] have proven more effective. Often texture is used for realism. In such cases photographic images of the type of texture desired are much easier to come by than believable analytic or algorithmic approximations.

On the other hand, where solid texture is required or desired, photographic imagery is less satisfactory. An image may be extruded through space to be used as a solid texture [Peachey85], but this hasn't been terribly useful. Recent progress in volumetric imagery used in CAT scan reconstructions suggests the possibility of three dimensional natural imagery as texture. However, limited access to CAT equipment and the huge storage required for volume data of useful resolution has apparently discouraged efforts of this sort.

Algorithmic simulations or approximations of natural processes [Yaeger86] have been effective under some circumstances. Given a large enough library of such functions, one might be able to do as well as with natural imagery. Some examples in later sections should help to illustrate this. However, it is vastly easier to develop a collection of texture functions with a camera than with the intensive creative work required for algorithmic textures. At the moment practicality makes algorithms the dominant source for solid texture and images the dominant source of surface texture.

### **1.4 The Role of Texture**

Texture techniques fit in a gap where the general shape of an object provides insufficient detail but where the exact details of the surface are not of importance. If an egg is viewed from any distance, a matte off-white surface approximates it sufficiently. If viewed more closely, the bumpy nature of the surface requires the use of one of the above texture techniques for an adequate rendition. However, if viewed from microscopic distance, the structure of the shell material becomes important and texture techniques won't be applicable.

Happily, the character of most scenes is such that texture techniques are almost always useful. Although texture is widely used, the realm of texture techniques has by no means been exhaustively explored. We have seen very little activity in time-varying texture, only slightly more in multiple textures, and a great deal of pioneering work remains to be done on integrated use of different textures at different distances from a surface. Texture should remain an area to which research contributions can be made for a long time.

What follows is, for the most part an expansion of the above. The simulation of texture effects without using texture is discussed in the next section. Thereafter, solid and surface texture are treated in more detail. The problem of mapping a texture function to a surface is given an entire section to itself. Finally, shorter sections on surface displacement and levels of detail lead to a brief roundup of areas where much research remains to be done.

## **2. Surface Shading Without Texture**

Surface characteristics may be expressed without the use of a texture function as such. For example the shininess of a surface can be characterized by specular reflectance added to the surface color [Warnock69] and by simulated highlight reflections from the light sources [Phong75]. Different materials reflect light in different ways. Blinn [Blinn77] and then Cook and Torrance [Cook82] described better reflectance models for varying materials.

Effects such as sparkling highlights and glowing filaments have been produced by adding randomly drawn bright colored lines radiating from those pixels representing bright areas. The appearance of neon tubes can be given by shading functions based on the eyepoint position rather than the light source position.

Effects similar to those using texture functions to vary color can be achieved by assigning colors on a vertex-by-vertex basis over shapes defined by polygon meshes. Figure 5 shows apparent texturing produced by algorithmic assignment of appropriate color and transparency values to selected vertices.

The egg and glass in figure 5 are both surfaces of revolution constructed from rectangular polygon meshes. This makes it relatively easy to assign a darker color to regularly spaced vertices of the egg. The spiral pattern on the glass is similarly constructed by skewing a pattern of stripes on the polygon mesh, the vertices falling within the stripes having opaque values.

The banana in figure 2 gets its green extremities and brown ends from vertex colors laboriously assigned by hand using a text editor. However, it is easy to conceive of an interactive vertex coloring system which would make this sort of thing much easier.

If the above sort of color variation is sufficient to achieve the desired effect, then the greater expense of texturing within polygons or patches can be avoided. Furthermore, there is another advantage to vertex coloring in that it adds no additional aliasing problems to those inherent in the rendering system.

## **3. Solid Texture**

Solid texturing requires intersecting a space-filling function with selected visible surfaces in an image. This poses two problems. (1) Three-space coordinates must be recovered at each pixel for the surface represented there. (2) The space filling functions themselves must be defined. Interesting functions don't come all that easily.

The original three-space coordinates of the surface may be recovered by applying the inverse of the transform taking the surface from the space in which it is defined to the image space. Texture space coordinates can then come from

applying any transformations into texture space. However, it is frequently more efficient to carry the texture space coordinates of the surface along, interpolating them just as you would other surface characteristics.

In particular, when the surface is approximated by polygons, color and other surface characteristics are traditionally determined by bilinear interpolation between polygon vertices, as in Gouraud shading [Gouraud71]. Therefore, the incremental cost of obtaining the shading values, and thus the texture space coordinates, is very small for each pixel. However, it should be noted that linearly interpolated texture tends to reveal the underlying polygonal structure that Gouraud shading obscures.

Where the surface is defined by patches or other primitives, the texture space coordinates may be treated similarly. If the image pixels are determined by patch subdivision then the texture space patch is subdivided in lock-step with the corresponding patch defined in the space in which the image is built.

### 3.1 Texture Space Mapping

Normally the texture would be expected to stick with an object as it moves around. To get the texture to stick with the surface, the texture space must be carried, untransformed through to the stage where shading functions are evaluated. However, other effects are possible by changing the space from which the texture coordinates are taken.

If the texture space coordinates are carried through the transform taking definition space coordinates to world space coordinates (the space in which the entire scene is assembled) then the object will appear to move through the texture as the object moves around the scene. This could be useful for creating images of a shape moving through a complicated pattern of light and shadow or in a medium with changing light patterns, such as might be found in an underwater scene.

The chiaroscuro caused by the complex shadows of the leaves of a tree could be easily produced by extruding the image of the tree as seen from the light source through the world space to define a three-space texture function. Objects intersected with this function would then have their shading modified to produce the effect of light and shadow. Of course care would have to be taken to avoid producing the texture on self-shadowed parts of the object.

One can imagine that many other useful mappings exist. Time-varying transforms could be used for many effects. For example an old-style rotating barber pole could be simulated using one of the example functions below. The effect of motion could be achieved by either using a rotation matrix varying in time or by translating along the axis of the pole through the texture space over time. See [Wyvill87] for some other examples.

### 3.2 Example Texture Space Functions

Simple space functions are easy to come up with. The example given in the introductory section,  $f(x,y,z) = \sin(cy)$ , for example, gives the world striations. However, interesting space functions take some imagination. Hopefully, the following simple examples will stimulate some ideas. Note in the following that terrestrial coordinates are used. Looking from the South,  $x$  is East (to the right).  $y$  is North (ahead), and  $z$  is height (up).

The red and green checkered pattern shown on the egg in figure 2 is produced by a real program similar to the following pseudocode. The code implements a function filling space with the three-dimensional analog of a checkerboard. Hint: when implementing this, make sure you understand the behavior of your "MOD" function about zero.

```

red: RGBCOLOR := [0.9, 0.2, 0.2];
green: RGBCOLOR := [0.4, 0.9, 0.2];
xTrue, yTrue, zTrue, isRed: BOOLEAN;
xTrue := (x MOD 2) = 1;    yTrue := (y MOD 2) = 1;    zTrue := (z MOD 2) = 1;
isRed := xTrue XOR yTrue XOR zTrue;
IF isRed THEN RETURN[red] ELSE RETURN[green];

```

The garish colored egg in figure 6 uses the following function providing a simple superposition of 3 sine waves modulating red along the x-axis, green along the y-axis, and blue along the z-axis. Note from figure 6 that while the result is understandable on a CRT, even a well-corrected conversion to color printing can make the correspondence between the image and the texture function hard to see.

```

r, g, b: REAL;
r := (Sin[10.0*x] +1.0) / 2.0;
g := (Sin[10.0*y] +1.0) / 2.0;
b := (Sin[10.0*z] +1.0) / 2.0;
RETURN[ [r,g,b] ];

```

The "barber pole" function is produced by applying a rotation transformation to a sine wave along the x-axis. The angle of rotation is linear in z. This function is used to modulate transparency in the glass in figure 1.

```

r, g, b, angle, intensity: REAL;
angle := 3.1416 * z;
intensity := Sin[40.0 * ( x * cos[angle] + y * sin[angle] ) ];
intensity := (intensity + 1.0) / 2.0;
r := 1.0;
g := intensity;
b := intensity;
RETURN[ [r,g,b] ];

```

More interesting, organic, texture space functions can be built upon randomized space-filling "noise" functions [Perlin85]. Given such a function, elaborate textures such as that on the teapot in figure 7 can be produced with slightly more imaginative functions similar to the latter two above.

The teapot texture is based on a function implemented by Perlin in the following way. A table of pseudo-random numbers is imposed on a three dimensional integer grid. A texture value is determined by interpolating between the 8 bounding grid points of a "voxel" (the cube defined by a set of 8 grid points) in texture space. A cubic function is used to smooth the interpolation with the intention of avoiding high spatial frequencies in the resulting texture function. The result is a space-filling noise function concentrated about a controlled band of frequencies.

The noise function is further modified by repeated application in a 1/f style characteristic of fractal functions [Mandelbrot77]. At each step of an iteration the texture space coordinates are doubled, effectively scaling the

spatial frequencies of the noise function by two. Applying the noise function to the scaled space coordinates, the resulting value is scaled by half the value used in the previous iteration and added to a running sum. The result is a function with decreasing power at increasing frequencies, similar in characteristics to many natural phenomena.

To produce the teapot texture, Perlin's function is evaluated at the texture space coordinates resulting from each pixel representing the teapot surface. The output of the natural phenomena function is included in a weighted sum with the x and y coordinates from texture space. Taking the sine function of the result yields a space filled with distorted waves radiating from the z-axis. Where this function is positive, it is perturbed by a secondary function providing higher frequency distorted waves varying along the z-axis. Where the function is positive, it is perturbed by another secondary function providing somewhat less higher frequency distorted waves varying along the x-axis. Taking this fundamental method, adjusting parameters, and randomly trying this and that yielded the teapot texture in a couple of hours of pure enjoyment.

Interesting as it is, the teapot texture has severe drawbacks. Unlike a natural texture, it does not continue to be interesting as you get closer to it. The detail in the texture is limited by the length of the  $1/f$  iteration in the natural phenomena function. Since the noise function is relatively expensive to evaluate, there is a practical limit on the length of the iteration since it must be done anew for each pixel.

To make a truly interesting texture, the natural phenomena function should be modified to accept an input specifying the texture range spanned by the pixel being colored. Given the range, suitable limits on the iteration can be set to give an appropriate level of detail. Note that the lower limit of the iteration must remain constant to ensure inclusion of the larger terms of the iteration. However, given the usual pattern of calculating adjacent pixels consecutively, the earlier terms of the iteration will change very slowly.

A scheme for caching the earlier terms and reevaluating them only when sufficiently distant from the point where the cached evaluation was computed would bound the iteration from below. Given such a scheme, the computational expense of the natural phenomena function should remain relatively constant over a wide range of viewing distances. It remains, however, for the texture function to do something interesting with the smaller variations in magnitude from the natural phenomena function at higher frequencies.

### **3.3 Sampling and Anti-aliasing**

When evaluating a texture function at discrete intervals then displaying the result on discrete pixels, we are sampling and reconstructing the function which exposes us to all the problems of aliasing. Solid texture is still relatively new and not much has yet been done about the problem.

For limited ranges of viewing distance, choosing a function with appropriate frequency characteristics avoids aliasing. A texture without sharp edges and small features won't cause problems. Also for many images a texture function without strong features, one consisting largely of random noise, will tend to mask aliasing artifacts.

However, the aliasing problem must be addressed at some point. The most effective technique put forth so far is the "natural phenomena" function described above. If the  $1/f$  iteration is chosen properly, based on the pixel extent in texture space, aliasing should not be visible.

No anti-aliasing methods have yet appeared in the literature which will handle sharp-edged functions, such as the cube tessellation function shown on

the egg in figure 2 and described algorithmically above. There are certainly brute force techniques which could be applied. The function could be sampled several times in the vicinity of a center point provided by the texture coordinates. The resulting samples could then be averaged, possibly weighted by proximity to the center point. The average could then be applied as the texture value.

It may be that no more samples would be required for this than are used in the recently developed techniques for jittered sampling applied to scan conversion [Cook86, Mitchell87]. This area is still open for research.

### **3.4 Extensions and Problems**

Recent development on surface reconstruction from computer-aided tomography and other sources of volume imaging data [Herman79, Levoy88, Reynolds88, Drebin88] begs the question of whether we should be using "real" solid textures scanned from real objects. Researchers at the University of North Carolina at Chapel Hill have considered scanning a log in a CAT machine, for example, to get a solid texture for wooden objects.

This idea poses all sorts of interesting problems. Is it possible to use replications of a natural solid texture to fill a large region? Methods discussed below for using replicated 2-D texture images on surfaces can probably be extended.

How much resolution will be necessary to make effective use of such texture? A cube with 128 pixels on a side includes  $27 \times 27 \times 27$  pixels, 2 megapixels. With 512 pixels on a side, 128 megapixels are involved.

How can you anti-alias natural solid texture? Multiple samples can be used as above. However there are a number of more efficient methods discussed in the section on surface texture below. Unfortunately, to varying degrees, they require even more storage.

Perlin suggested the use of time-varying functions for portraying fire and other dynamic phenomena. This poses exciting possibilities which remain largely unexplored.

## **4. Surface Texture**

In contrast to solid texture, where direct evaluation of a texture function has been the usual approach, surface texturing has been dominated by techniques based on storage of the texture in a table. The reasons for this are made clear by noting the difficulty of the methods discussed in the section on solid texture. It is no easier to come up with interesting two-dimensional texture functions than it is for three-dimensional ones. However, it is ever so much easier to produce two-dimensional images by many different means than it is to get true volume imagery. This helps explain the dominance of 2-dimensional texturing techniques in current practice.

The fact that surface texturing techniques were developed some ten years before solid texture is almost certainly due to the advent of the frame buffer and the subsequent explosion of applications of stored two-dimensional imagery. Catmull [Catmull74] saw the natural correspondence between the addressing of a scanned image and the parametric space used to develop the bicubic patches he was working with. Blinn [Blinn78] noted a few years later how easy it was to produce patterns for textures using the tools he and others had developed for painting and image manipulation on frame buffers.

Meanwhile, Williams [Williams78] and others at the New York Institute of Technology were discovering increasingly interesting and varied ways of using stored images. The few years between the development of the frame buffer and



the wide availability of virtual memory and large resident storage on smaller computers allowed image-based surface texturing to get well-established before competing methods were developed.

A great deal of attention has been paid to the problem of how to map 2-d image data onto surfaces. In contrast to solid data, where the mapping is normally trivial, surface texture can't be evenly distributed over the surface of an object in any generally applicable way. The problem is a reversal of the problem posed by putting the surface of the earth on a rectangular piece of paper which has intrigued cartographers over centuries.

The remainder of this section is devoted to the calculation, preparation, and storage of two-dimensional texture functions. The following section is devoted to the problem of mapping from the texture function to the surface.

#### **4.1 Texture Sources**

Many methods have been used to develop texture. All the techniques described in the previous section on solid texture could easily be adapted to two dimensions. However, there has been little activity of this sort. It takes considerably less mathematical insight to use image data.

Scanners have long been available for converting photographic imagery into stored sample arrays suitable for use as textures, and such imagery has been widely used. However, photographic imagery has its limits. The ideal function for bump mapping, for example would be a function of height rather than intensity or color.

To get a texture map for tree bark, Bloomenthal [Bloomenthal85] obtained an x-ray image of a plaster cast of an actual piece of bark. The result, a thickness function of the bark, was precisely what is needed for bump mapping.

Blinn [Blinn78] often used images produced by hand on an early digital paint system for his texture work. He also used inverse Fourier transforms of digital paintings as texture, a technique he called "Fourier synthesis". The digital paintings were taken as frequency spectra then transformed to spatial images for use as texture. This gave texture images with the desired formlessness and uniform character. Lewis [Lewis84] later described more fully realized, but similar, techniques.

Video frame-grabbers are now so widely available and inexpensive that more examples of animated texture are bound to appear. The Ampex ADO and Quantel Mirage provide a form of texture mapping using video imagery mapped onto simple surfaces as special effects for television. More recently, a system mapping video over much more complex surfaces was described by Oka et al working at a Sony research center [Oka87].

One problem with sampled images as texture is that, once sampled, images are only useful over a small range of resolutions. When expanded, it becomes obvious that there is limited detail, the images become annoyingly fuzzy. When shrunk, there is unuseable detail which makes aliasing a potential problem.

Two-dimensional algorithmic or analytic functions can be designed as shown in the previous section to avoid aliasing artifacts by limiting spatial frequencies to prescribed ranges. Norton, Rockwood and Skomolski [Norton82] showed some examples of functions composed of sine waves and described a "clamping" method for limiting the spread of frequencies to those appropriate for the area covered by a pixel.

Gagalowicz and Ma [Gagalowicz85] describe a texture synthesis method based on analysis of natural textures. They synthesize a texture function based on

statistical analysis of an image. This function captures the characteristics of the texture without retaining any of the prominent features of the image. The resulting function can then be applied without as much concern for position or texture image boundaries. Since the frequency characteristics are known, aliasing can be kept under control.

Burt and Adelson [Burt83] describe an application using multiple filtered versions of an image each representing a limited range of spatial frequencies. Their application was the joining of dissimilar images. However, their basic method has been applied to texture mapping as well.

#### **4.2 Edge matching**

Texture functions with finite extent (e.g., images) are usually applied repeatedly over a surface. It becomes critically important to arrange the function so that it can be seamlessly abutted with copies of itself. The pixels along the right-hand edge must closely match those on the left edge, and so must the top and bottom edges match.

A photograph which appears to be evenly lit will often exhibit gross disparities when opposite edges are compared. Various image processing techniques may be applied to remove subtle gradations in shading. Other techniques, such as that of Burt and Adelson mentioned above can be used to force smooth joins between arbitrarily dissimilar edges. However, if the edges are grossly mismatched the results can be arbitrarily unsatisfactory as repeated texture patterns.

Image edges can be matched straightforwardly using "vignetting". All edges of the image are "faded" (interpolated) to a common shade. If the image is forced to black all around then the edges have to match. However, unless this is done with great subtlety on an image of suitable character, the pattern of repeated texture images will be obvious.

Blinn's digitally painted textures, mentioned above ensured well-matched edges by using a solid-color background. A successful technique used at the New York Institute of Technology employs a paint program with several helpful tools (such as a "brush" which can blend adjacent regions). Using scrolling and panning in the frame buffer, the texture image can be manipulated to put the join between opposite edges in the middle of the screen. The image is then touched up, by hand, to match the edges.

Images with large discrete features will generally make the repetition of texture obvious. It is better to use an image with fairly uniform character across the image. Bloomenthal's tree bark, mentioned above, is a good example. The overall character of the tree bark image is uniform. Burt and Adelson's algorithm serves to join the quite dissimilar edges.

#### **4.3 Sampling and Anti-aliasing**

The dominance of image-based methods for texture has led to extensive efforts at avoiding aliasing problems. The problem, as seen in earlier sections is to limit the spatial frequencies to those matching the area represented at a pixel.

Early anti-aliasing methods described by Blinn and Newell [Blinn76] and then Feibush, Levoy and Cook [Feibush80] calculated a weighted sum of all texture pixels which mapped into the area covered by a pixel. This general method is prohibitively expensive where many texture pixels map into one image pixel. Each of many texture pixels must be retrieved and then added to a weighted sum, a lot of work for each image pixel.

#### **4.4 Multiresolution Texture Images**

Williams [Williams83] noted that by precalculating smaller texture images, wherein each pixel is a weighted sum of a set of pixels from the original image, much of the pixel summing computation could be reused across many pixels. Williams' solution requires that the texture area mapped to each pixel be square, a rather loose approximation which produces fuzzier imagery than more exacting methods

While scan converting a surface, the texture coordinate offset from one pixel to the next is noted and used to select a texture image which has pixels which roughly match the size of the offset. If neighboring pixels map to places in the texture which are eight texture pixels apart then a one-eighth size texture image is appropriate.

At the time (early 1979), all work at Williams' site was being done on 512x512 pixel frame buffers. Williams noted that an RGB texture image fit neatly into such a frame buffer in the following way: Three 256x256 pixel images are stored in the lower and left quadrants of the frame buffer, leaving the upper right quadrant unfilled. Half-resolution images are then computed and three 128x128 pixel images stored similarly in the upper right quadrant. This leaves the upper right quadrant of the upper right quadrant empty. Recursive application of the process fills the frame buffer (save the upper rightmost pixel) with successively half-sized averaged texture maps.

Texture images are then stored at factors of two in resolution, each pixel in a smaller image being the average of four pixels from the next larger image. At each pixel, the scan conversion process provides a texture coordinate pair and offsets to the next texture coordinate pair. The integer part of the offset, if greater than one, determines which two sizes of texture map to use.

The texture coordinates are scaled appropriately and four texture pixels are retrieved from each map. The texture coordinates are then used to interpolate between the four pixels to best approximate the value which should lie exactly at the coordinates. Having recovered two pixel values (or colors) the fractional part of the texture offset is then used to interpolate between them.

The first set of interpolations serves to smooth out changes between adjacent texture pixels and to handle the case where many image pixels map into one texture pixel. The final interpolation serves to smooth transitions between texture maps at different scales.

The general idea of a nested hierarchy of texture images has appeared in a number of places. The concept was used in early versions of the General Electric visual flight simulator hardware around 1962 [Rougelot69]. Dungan et al [Dungan78] discuss a hierarchy where each nested image is half the size of the next larger one. In these cases the texture was applied only to flat surfaces representing ground planes, etc. Multiresolution images are also used in various applications within the field of image processing.

The importance of this approach is that texture values are computed in constant time. A constant time algorithm can generally be implemented in hardware to run in real time. Furthermore, for nearly all situations, the constant time algorithms run much faster than those which must sum over all texture pixels which map into an image pixel.

#### **4.5 Integrated Texture Images**

Crow [Crow84] and Perlin independently discovered a method generalizing the set of possible prepared texture areas to rectangular areas aligned with the texture image axes. This works particularly well on texture images with primarily rectilinear patterns, such as checkerboards (figure 8).

An image can be integrated by summing its pixels. Texture values which represent the average pixel value over a subimage from the texture can be had by dividing the integral over that area by the area. A table can be built, as follows, from which values representing the integral of axis-aligned rectangles can be easily calculated.

Consider a table where each entry represents the sum of all pixel values lying to the left of and below a given position. Such a table, often called a "summed-area table", is easily computed iteratively from a texture image. Indexing the table in the same way as the texture image, table entry  $(x_1, y_1)$  would represent the sum of all pixels with x-coordinates less than or equal to  $x_1$  and y-coordinates less than or equal to  $y_1$ .

The sum over a rectangular area bounded on the bottom left by  $(x_l, y_b)$  and on the top right by  $(x_r, y_t)$  is then recovered by four accesses to the summed-area table T.

$$S = T(x_r, y_t) - T(x_l, y_t) - T(x_r, y_b) + T(x_l, y_b).$$

See figure 9. S is then divided by  $(x_r - x_l) * (y_t - y_b)$  to get the average pixel value.

As in the multiresolution texture images above, interpolation between table values is used to more accurately place the corners of a texture rectangle and to allow smooth variation between texture values when texture pixels spread over multiple image pixels. Note that this is unnecessary where a large area of texture is involved as the resultant sum will be changed insignificantly.

Glassner [Glassner86] extended the application of summed-area tables by using repeated applications of the texture retrieval algorithm. An image pixel, if considered a square area, forms an arbitrary quadrilateral image in the texture. The difference between this quadrilateral and the axis-aligned rectangle provided by the summed-area table can be reduced by removing smaller and smaller rectangular regions from the initial one. Therefore repeated retrievals from the summed-area table can improve the fit to the pixel image.

Glassner implemented his scheme along with an additional table which helps measure the variance in a texture region. Using the variance to determine how close an approximation is needed, he claims an increase in texture accesses of only 50 percent or so. This is largely due to the fact that only a small minority of pixels in a given image map into large regions of texture. Nonetheless his adaptive technique allows the work to be concentrated where it is needed.

Perlin suggested extending integrated texture to repeated integrations to get weighted sums of texture pixels instead of a simple average. Heckbert [Heckbert86] implemented the idea and was able to show improvements. A second integration allows recovery of a rectangle weighted by a pyramid shape using nine texture accesses instead of four.

The disadvantage of repeated integrations is that they require high precision arithmetic (greater than 32 bits) for the stored tables. However, for good texture functions (small features, evenly distributed) it may be possible to fit smooth, easily evaluated functions to the stored tables. Given the fitted functions, only the differences from the functions would need to be stored. The differences have a chance of staying within 32 bits in length. Of course this incurs the additional overhead of evaluating the fitted functions for each texture retrieval.

Summed-area tables were independently discovered and used in the image processing community. Furthermore the concept shows up in probability as well.

Linear use of summed functions to integrate appears widely. See [Crow78] for one example.

In an excellent recent survey [Heckbert86], Heckbert compares integrated textures, multiresolution textures and a hybrid method which combines multiresolution texture with multiple texture retrievals in a weighted sum. A weighted average of multiple texture pixels was shown to yield the best absolute result. However, it appears that methods using a small (or, hopefully, constant) number of retrievals from a prepared texture function will provide the best quality consistent with reasonable cost. This area continues to inspire good, original research.

## 5. Mapping Texture to Surfaces

Before texture may be applied to a surface it must be determined how the texture fits over the surface. Catmull's [Catmull74] technique used shapes consisting of bicubic patches. The parameters of the bicubic patch formulation provided an obvious index into the texture image. Therefore, each patch contained a replica of the texture image. This approach has been used often, most visibly on the Utah teapot [Blinn76, Crow87].

A generalization of this idea allows an array of polygons or patches to be superposed on a replicated array of texture images. Where an shape is a simple surface of revolution, for example, it is generally composed of a rectangular array of surfaces. The surface may be conceptually split open at a seam and flattened to a rectangular array. A single texture image or multiple copies of one may be mapped to this array. The mapping results in texture coordinates associated with each vertex in the shape.

An advantage of a more global mapping is that the mapping may be scaled translated, rotated, or skewed for additional effect as shown on the striped egg in figure 1. The texture may even be animated. Figure 10 shows a simulated paper web. In an animated sequence the texture slides along the web, giving the appearance of a moving web when actually only the texture mapping transform is being translated, a substantially easier modeling task.

Given texture coordinates at the vertices, the coordinates at pixels can be found by interpolation, linear in the case of polygons, up to bicubic for appropriate patches. The interpolation process can easily provide texture coordinate offsets between adjacent pixels for use by the anti-aliasing algorithms described in the last section.

Caution must be exercised when using linear interpolation of texture values. Perspective distortion must be taken into account if interpolating over more than a small number of pixels. Perspective effects will cause the texture to appear to sag and to move strangely during animation.

These effects can be avoided by using adaptive subdivision techniques which guarantee that distortion due to linear interpolations will stay below a given threshold. However, large areas can be interpolated using a more complex formula including a perspective divide at each pixel. See Heckbert [Heckbert86] for a thorough treatment of this problem.

Where the polygon or patch mesh is irregular or the shape consists of an assembly of simpler pieces, different mapping techniques must be employed. A popular technique uses the normal vector to the surface to determine which texture value to retrieve. This technique was pioneered by Blinn and Newell [Blinn76] for "environment mapping".

Environment mapping simulates an enveloping environment for the shape by using a texture image representing an unwrapped spherical surface, similar to

some map projections used to represent the surface of the Earth. A reflected vector calculated from the eyepoint direction and normal vector is converted to spherical coordinates. The resulting azimuth and elevation are used to address the texture image.

Environment maps have advantages for representing light sources or lighting in general. The lighting environment can be made into an image and used as an environment map. The advantage lies in being able to have an arbitrarily complex lighting scheme at constant cost. Furthermore, using the anti-aliasing schemes described above, light source reflections (highlights) are automatically anti-aliased, a difficult task for direct highlight calculation methods.

The same general mapping technique can be used to apply texture to a surface. Instead of using a reflected vector the normal vector is used directly to pick off a texture value. Bier [Bier86] has done a thorough investigation of this approach. He points out that undulations in the surface cause wide excursions in the direction of the surface normal which can, in turn, cause the texture to vary excessively (even invert) over the surface.

Bier suggests using a ray from the centroid of the object through its surface to pick off a texture value. This improves the situation but fails for objects with projections or concavities which cause a ray from the centroid to pierce the surface more than once.

Barr [Barr84] proposes a decal-like paradigm in which an area of texture is placed in proximity to the surface and then mapped to it using nearness and the direction of the surface normal to pick off texture values. This opens a promising area of research which hasn't been followed up very well as yet. However, the problem of stitching together decals to completely cover a surface looks no easier than that of mapping replicated texture images.

Ma and Gagalowicz [Ma85] show a mapping based on a starting point and propagation of a local coordinate system along geodesic curves on the surface. This shows some promise as long as the surface can be broken into locally simple pieces. They slice their objects by sets of parallel planes to accomplish this. However, it appears that complex areas such as branch points in a surface still cannot be handled automatically.

Complex shapes composed of multiple pieces, having concavities and/or projections, or just tortuously convoluted don't lend themselves to any of the mapping schemes described thus far. Solid texturing avoids the mapping problem but poses difficulties in describing natural surface textures. Finding ways of stitching together patches of texture to cover arbitrary surfaces remains a fertile area for further research.

## **6. Surface Displacement and Modification**

The solid and surface texturing schemes described in the previous sections are fine for variations in surface color and transparency characteristics, and even low relief. However surface texture in reality is a richer domain involving a continuum of relief ranging from bumpy surfaces like stucco to extremely complex phenomena such as hair and clothing.

A class of higher-relief textures can be developed based on local displacement of the surface according to a texture function. This technique extends what can be accomplished with texture function techniques. For example, it is possible with this technique to view lumpy surfaces closely enough to see the self-occlusions caused by one lump hiding another. Surfaces with large displacements, such that the shape of the profile is clearly modified are also possible.

"Displacement mapping", as it has come to be called, extends the technique of "bump mapping" to actual surface displacements. This, of course must be accomplished through a subdivision process which dices the polygon or patch into small enough pieces to express the surface undulations. As the surface is subdivided the texture function is used to displace the resulting vertices.

Pixar [Cook87] dices patches into subpixel sized "micropolygons" which are then displaced by the texture function. This process can run very efficiently where a texture image is mapped directly to the patch being diced. In this case the appropriate texture value can be read directly from the coordinates used for dicing the patch. Simple scaling giving multiple texture images per patch or multiple patches per texture image maintains this simplicity.

The Pixar algorithm is predicated on the idea that patches will be diced into micropolygons. If large surface displacements occur over small areas, the micropolygons could be stretched to significantly larger sizes, possibly causing problems with their display algorithm. At the time of their paper, they claimed not to have tried this.

Hiller [Hiller87] built a similar displacement mapping system on top of the author's rendering system at Xerox PARC. In this implementation, polygons are adaptively subdivided until it can be determined that further subdivisions would not displace the edges of the subpolygons enough to make a difference in the resulting image. The adaptive termination of the algorithm allows for arbitrary excursions in the resulting surface. It remains to be seen how useful that will be.

There are limits to what can be expressed with displacement mapping. For example, a displacement map only offers a height field. Therefore a surface can't fold back on itself or produce cave-like hollows. Other techniques must be used to model hairy or spiny surfaces or add high relief with loops and hollows in the surface.

To get surfaces with more relief than provided by displacement maps, additional structures must be layered over the polygons or patches. An early example of this was described by Csuri et al [Csuri79]. Here a surface of revolution served as a base for a large number of long thin polygons which radiated from the surface. While very limited in scope, the method made some interesting images of wire-brush-like objects.

More recently, Miller [Miller88] has produced very convincing images of hairy surfaces by drawing carefully colored line segments from specified points on the polygons of an underlying surface. He has also produced scaly surfaces by attaching polygons rather than lines to the underlying surface.

The attachment points of the hairs/scales are determined either by a regular grid for man-made effects, such as a brush, or by a dithered grid for more natural surfaces. Given the underlying surface and the attachment points, all the techniques developed for particle systems [Reeves83] can be applied to produce textured surfaces.

Using an array of attachment points on a surface, of course allows the attachment of anything that can be rendered. One can envision a texture of attached doorknobs or decorations using full-dimensional figures as on a Grecian frieze. One advantage of treating such things as texture is that the hidden surface problem is eased since rendering order can be controlled by the order of surface elements on the underlying surface as long as the attached shapes don't interlock.

## **7. Unifying Texture over Levels of Detail**

A longstanding problem in computer graphics lies in controlling the level of detail in an image or animated sequence where objects may be seen from vastly different distances. Flight simulator databases often contain multiple descriptions of objects, sparse descriptions for distant views and detailed ones for closeups. Texture techniques offer a range of detail which could be used similarly.

Texturing by color variation, by bump mapping and by surface displacement span a hierarchy of levels of detail which can be useful in representing the same surface at different sizes in an image. However, to gracefully segue from one description to the next will require that all these techniques be controlled by the same texture function or derivatives of the same function.

A continuous effective zoom into a surface starting with a single color at a very great distance then working through color mapping, bump mapping and displacement mapping as the surface is approached has yet to be demonstrated.

Hiller [Hiller87] has described using the same texture for bump mapping and displacement mapping. This works quite reasonably since both techniques require a height field. However, as seen in figure 4, the same function used for intensity mapping and bump mapping is less successful. The height map of the bump texture determines shading which changes with the relative position of the light source. This sort of variation isn't possible with conventional intensity mapping. It remains to be seen whether some sort of derived intensity map can be used effectively in a texture hierarchy for levels of detail.

There are other models for surface development which take care of multiple levels of detail quite naturally. For example, fractal and other stochastic models develop detail adaptively. Note, however, that even these surfaces pose some problems. As a surface is subdivided, there is no guarantee that the shade over a small section of surface after subdivision will be the same as that before subdivision. Visible artifacts are most certainly possible. Nonetheless, as long as reasonably unobtrusive changes take place, adaptively subdivided surfaces can be zoomed quite successfully.

A technique frequently seen in movies about Dr. Jekyll and Mr. Hyde can be used to segue between very different representations. Here both representations are used over a fairly wide range with a mix of the two appearing in the image. This is similar to the linear interpolation between texture maps at different scales used in the multiresolution texture maps discussed above. Unfortunately, the effectiveness of this technique is directly related to the similarity of the interpolated images.

There are some important unsolved problems remaining in this area. It is not at all clear how one could gracefully segue from any kind of ordinary texture map into a hairy surface based on attachment points, for example. Kajiya [Kajiya85] and Miller [Miller 88] both discuss anisotropic reflection models which could serve to represent certain kinds of hairiness from a distance. But how do you get from there to a particular representation of individual hairs?

## **8. Areas Needing Further Work**

It is pretty clear from the above that a number of issues concerning texture remain unresolved at this time. The following list will serve as a summary.

### **8.1 Rich texture definitions**

Solid texture functions give us the possibility for truly interesting surfaces which continue to show detail at many magnifications, as do the



recently popular Mandelbrot and Julia sets. As yet this hasn't been demonstrated, except for stochastic terrain models.

### **8.2 Natural solid texture sources**

Methods for reconstructing surfaces from volume data make it clear that natural objects could be scanned and stored as solid textures. Ever expanding memory capacity should make such textures possible if not practical. This just awaits somebody doing it.

### **8.3 Better anti-aliasing**

Techniques for preparing texture maps for use at different resolutions and methods of refining the areas mapped to a pixel can be improved as can frequency bounds on the detail returned by algorithmic texture functions.

### **8.4 Mapping to arbitrary surfaces**

Mapping texture to surfaces with branches, projections, and hollows remains difficult.

### **8.5 Time varying texture**

Some very rudimentary probes at time varying texture were described above. However, some truly interesting dynamic images should be possible through texture techniques. There should be some interesting effects possible by running simultaneous animations, the output of one serving as a texture map to the other.

### **8.6 Fuzziness**

It ought to be possible to represent fuzzy surfaces in a way which captures what one sees from a distance at which the individual hairs can't be distinguished.

Texture offers much added richness to our imagery at a bounded cost. It has proven very useful over the past decade. It now remains to integrate it better with more and less detailed descriptions, to generalize it to a broader range of effects, and to find ways to speed up its execution without incurring deleterious defects in the resulting images.

## **References**

- [Barr84] Alan H. Barr, "Decal Projections", Siggraph '84, Course #15, Mathematics of Computer Graphics, July 1984.
- [Bier86] Eric A. Bier and Kenneth R. Sloan Jr., "Two-Part Texture Mappings", IEEE Computer Graphics and Applications, 6, 9, September 1986, pp. 40-53.
- [Blinn76] James F. Blinn and Martin E. Newell, "Texture and Reflection in Computer Generated Images", Communications of the ACM, 19, 10, October 1976, pp. 542-547.
- [Blinn77] James F. Blinn, "Models of Light Reflection for Computer Synthesized Pictures", Proc. Siggraph '77, Computer Graphics, 11, 2, July 1977, pp. 192-198.
- [Blinn78] James F. Blinn, "Simulation of Wrinkled Surfaces", Proc. Siggraph '78, Computer Graphics, 12, 3, August 1978, pp. 286-292.
- [Bloomenthal85] Jules Bloomenthal, "Modeling the Mighty Maple", Proc. Siggraph '85, Computer Graphics, 19, 3, July 1985, pp. 305-311.

- [Burt83] Peter J. Burt and Edward H. Adelson, "A Multiresolution Spline With Application to Image Mosaics", ACM Transactions on Graphics, 2, 4, October 1983, pp. 217-236.
- [Catmull74] Edwin E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces", UTEC-CSc-74-133, Department of Computer Science, University of Utah, Salt Lake City, Utah, December 1974.
- [Cook82] Robert L. Cook and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics", ACM Transactions on Graphics, 1, 1, January 1982, pp. 7-24.
- [Cook84] Robert L. Cook, "Shade Trees", Proc. Siggraph '84, Computer Graphics, 18, 3, July 1984, pp. 223-231.
- [Cook86] Robert L. Cook, "Stochastic Sampling in Computer Graphics", ACM Transactions on Graphics, 5, 1, January 1986, pp. 51-72.
- [Cook87] Robert L. Cook, Loren Carpenter, and Edwin Catmull, "The Reyes Image Rendering Architecture", Proc. Siggraph '87, Computer Graphics, 21, 4, July 1987, pp. 95-102.
- [Crow77] Franklin C. Crow, "The Aliasing Problem in Computer-Generated Shaded Images", Communications of the ACM, 20, 11, November 1977, pp. 799-805.
- [Crow78] Franklin C. Crow, "The Use of Grayscale for Improved Raster Display of Vectors and Characters", Proc. Siggraph '78, Computer Graphics, 12, 2, July 1978, pp. 1-5.
- [Crow84] Franklin C. Crow, "Summed Area Tables for Texture Mapping", Proc. Siggraph '84, Computer Graphics, 18, 3, July 1984, pp. 207-212.
- [Crow86], "Advanced Image Synthesis - Anti-Aliasing", In: (Enderle, G., Grave, M. and Lillehagen, F. Eds.) Advances in Computer Graphics I, Springer-Verlag, 1986, pp. 419-440
- [Crow87] Franklin C. Crow, "The Origins of the Teapot", IEEE Computer Graphics and Applications, 7, 1, January 1987, pp. 8-19
- [Csuri79] Charles Csuri et al, "Towards an Interactive High Visual Complexity Animation System", Proc. Siggraph '79, Computer Graphics, 13, 2, August 1979, pp. 289-299.
- [Drebin88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan, "Volume Rendering", Proc. Siggraph '88, Computer Graphics, 22, 3, August 1988.
- [Dungan78] William Dungan Jr. , Anthony Stenger, and George Suttly, "Texture Tile Considerations for Raster Graphics", Proc. Siggraph '78, Computer Graphics, 12, 3, August 1978, pp. 130-134.
- [Feibush80b] Eliot A. Feibush, Marc Levoy, and Robert L. Cook, "Synthetic Texturing Using Digital Filters", Proc. Siggraph '80, Computer Graphics, 14, 3, July 1980, pp. 294-301.
- [Fournier82] Alain Fournier, Don Fussell and Loren Carpenter, "Computer Rendering of Stochastic Models", Communications of the ACM, 25, 6, June 1982, pp. 371-384.
- [Gagalowicz85] Andre Gagalowicz and Song De Ma, "Model Driven Synthesis of Natural Textures for 3-D Scenes", Proc. Eurographics '85, September 1985, pp. 91-108.
- [Glassner86] Andrew Glassner, "Adaptive Precision in Texture Mapping", Proc. Siggraph '86, Computer Graphics, 20, 4, August 1986, pp. 297-306.

- [Gouraud71] Henri Gouraud, "Computer Display of Curved Surfaces", IEEE Transactions on Computers, c-20, 6, June 1971, pp. 623-629.
- [Haruyama84] Shinichiro Haruyama and Brian A. Barsky, "Using Stochastic Modeling for Texture Generation", IEEE Computer Graphics and Applications, 4, 3, March 1984, pp. 7-19.
- [Heckbert86] Paul S. Heckbert, "Survey of Texture Mapping", IEEE Computer Graphics and Applications, 6, 11, November 1986, pp. 56-67.
- [Herman79] Gabor T. Herman and H. K. Liu, "Three-Dimensional Display of Human Organs from Computer Tomograms", Computer Graphics and Image Processing, January 1979.
- [Hiller87] Martha Hiller, "Three Dimensional Texture Trees", Master's thesis, Mass. Institute of Technology, Cambridge, Mass., June 1987.
- [Levoy88] Mark Levoy, "Display of Surfaces from Volume Data", IEEE Computer Graphics and Applications, 8, 3, May 1988, pp. 29-37.
- [Lewis84] John-Peter Lewis, "Texture Synthesis for Digital Painting", Proc. Siggraph '84, Computer Graphics, 18, 3, July 1984, pp. 245-292.
- [Ma85] Song De Ma and Andre Gagalowicz, "Determination of Local Coordinate Systems for Texture Synthesis on 3-D Surfaces", Proc. Eurographics '85, September 1985, pp. 109-118.
- [Mandelbrot77] Benoit B. Mandelbrot, Fractals - Form, Chance, and Dimension, W.H. Freeman, San Francisco, 1977.
- [Mitchell87] Don P. Mitchell, "Generating Antialiased Images at Low Sampling Densities", Proc. Siggraph '87, Computer Graphics, 21, 4, July 1987, pp. 65-72.
- [Norton82] Alan Norton, Alyn P. Rockwood, and Philip T. Skomolski, "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space", Proc. Siggraph '82, Computer Graphics, 16, 3, July 1982, pp. 1-8.
- [Oka87] Masaaki Oka et al, "Real-Time Manipulation of Texture-Mapped Surfaces", Proc. Siggraph '87, Computer Graphics, 21, 4, July 1987, pp. 181-188.
- [Peachey85] Darwyn R. Peachey, "Solid Texturing of Complex Surfaces", Proc. Siggraph '85, Computer Graphics, 19, 3, July 1985, pp. 279-286.
- [Perlin85] Ken Perlin, "An Image Synthesizer", Proc. Siggraph '85, Computer Graphics, 19, 3, July 1985, pp. 287-296.
- [Phong75] Bui-Tuong Phong, "Illumination for Computer Generated Images", Communications of the ACM, 18, 6, June 1975, pp. 311-317.
- [Reynolds88] R. A. Reynolds et al, "Realistic Presentation of Three-Dimensional Medical Datasets", Proc. Graphics Interface '88, June 1988.
- [Reeves83] William T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions On Graphics, 2, 2 April 1983, pp. 91-108.
- [Rougelot69] Rodney S. Rougelot, "The General Electric Computer Color TV Display", In: (M. Faiman and J. Nievergelt Eds.) Pertinent Concepts in Computer Graphics, University of Illinois Press, Urbana, 1969, pp. 261-281.
- [Warnock69] John E. Warnock, "A Hidden Surface Algorithm for Computer Generated Half-Tone Pictures", Dept. of Computer Science, University of Utah, TR 4-15. June 1969, (available through NTIS #AD-753-671).

- [Williams78] Lance Williams, "Casting Curved Shadows on Curved Surfaces", Proc. Siggraph '78, Computer Graphics, 12, 3, August 1978, pp. 270-274.
- [Williams83] Lance Williams, "Pyramidal Parametrics", Proc. Siggraph '83, Computer Graphics, 17, 3, July 1983, pp. 1-12.
- [Wyvill87] Geoff Wyvill, Bryan Wyvill, and Craig McPheeters, "Solid Texturing of Soft Objects", IEEE Computer Graphics and Applications, 7, 12, December 1987, pp. 20-26.
- [Yaeger86] Larry Yaeger, Craig Upson, and Robert Myers, "Combining Physical and Visual Simulation - Creation of the Planet Jupiter for the Film '2010'", Proc. Siggraph '86, Computer Graphics, 20, 4, August 1986, pp. 85-93.

## Figures

- Figure 1: Texture as intensity (egg) and transparency (glass).
- Figure 2: Natural images, vertex colors, and space functions as texture.
- Figure 3: Variations in reflectivity, the brown texture is matte.
- Figure 4: A texture used both for intensity mapping and bump mapping.
- Figure 5: Apparent surface texture from vertex coloring.
- Figure 6: Garish, but simple, space filling texture function.
- Figure 7: Teapot with pseudonatural solid texture.
- Figure 8: Checkerboard texture, (from top) aliased, multiresolution, summed area.
- Figure 9: Recovery of texture values from a summed-area table.
- Figure 10: A simulation of a printer web.