SURVEY OF TEXTURE MAPPING

Paul S. Heckbert

PIXAR

ABSTRACT

Texture mapping is one of the most successful new techniques in high quality image synthesis. Its use can enhance the visual richness of raster scan images immensely while entailing only a relatively small increase in computation. The technique has been applied to a number of surface attributes: surface color, surface normal, specularity, transparency, illumination, and surface displacement, to name a few. Although the list is potentially endless, the techniques of texture mapping are essentially the same in all cases. We will survey the fundamentals of texture mapping, which can be split into two topics: the geometric mapping that warps a texture onto a surface, and the filtering that is necessary in order to avoid aliasing. An extensive bibliography is included.

INTRODUCTION

Why Map Texture?

In the quest for more realistic imagery, one of the most frequent criticisms of early synthesized raster images was the extreme smoothness of surfaces - they showed no texture, bumps, scratches, dirt, or fingerprints. Realism demands complexity, or at least the appearance of complexity. Texture mapping is a relatively efficient means to create the appearance of complexity without the tedium of modeling and rendering every 3-D detail of a surface.

The study of texture mapping is valuable because its methods are applicable throughout computer graphics and image processing. Geometric mappings are relevant to the modeling of parametric surfaces in CAD and to general 2-D image distortions for image restoration and artistic uses. The study of texture filtering leads into the development of space variant filters, which are useful for image processing, artistic effects, depth-of-field simulation, and motion blur.

Definitions

We define a *texture* rather loosely: it can be either a texture in the usual sense (e.g. cloth, wood, gravel) - a detailed pattern that is repeated many times to tile the plane, or more generally, a multidimensional image that is mapped to a multidimensional space. The latter definition encompasses non-tiling images such as billboards and paintings.

Texture mapping means the mapping of a function onto a surface in 3-D. The domain of the function can be one, two, or three-dimensional, and it can be represented by either an array or by a mathematical function. For example, a 1-D texture can simulate rock strata; a 2-D texture can represent waves, vegetation [Nor82], or surface bumps [Per84]; a 3-D texture can represent clouds [Gar85], wood [Pea85], or marble [Per85a]. For our purposes textures will usually be 2-D arrays.

The source image (*texture*) is mapped onto a surface in 3-D *object space*, which is then mapped to the destination image (*screen*) by the viewing projection. Texture space is labeled (u, v), object space is labeled (x_o, y_o, z_o), and screen space is labeled (x, y).

We assume the reader is familiar with the terminology of 3-D raster graphics and the issues of antialiasing [Rog85], [Fol82].

---

author's address: PIXAR, P.O. Box 13719, San Rafael, CA, 94913.

This paper appeared in *IEEE Computer Graphics and Applications*, Nov. 1986, pp. 56-67. An earlier version of this paper appeared in *Graphics Interface '86*, May 1986, pp. 207-212.

Uses for Texture Mapping

The possible uses for mapped texture are myriad. Some of the parameters that have been texture mapped to date are, in roughly chronological order:

surface color (the most common use) [Cat74],
specular reflection [Bli76],
normal vector perturbation (‘‘bump mapping’’) [Bli78a],
specularity (the glossiness coefficient) [Bli78b],
transparency [Gar85],
diffuse reflection [Mil84],
shadows, surface displacement, and mixing coefficients [Coo84],
local coordinate system (‘‘frame mapping’’) [Kaj85].

Our focus in this paper is on the computational aspects of texturing: those tasks common to all types of texture mapping. We will not attempt a thorough survey of the optical and semantic implications of texturing, a careful review of which is available [Car85]. One type of texture mapping warrants particular attention, however: illumination mapping.

Illumination mapping is the mapping of specular or diffuse reflection. It is also known as reflection mapping or environment mapping. Mapping illumination is rather different from mapping other parameters, since an illumination map is not associated with a particular object in the scene but with an imaginary infinite radius sphere, cylinder, or cube surrounding the scene [Gre86a]. Whereas standard texture maps are indexed by the surface parameters u and v, a specular reflection map is indexed by the reflected ray direction [Bli76] and a diffuse reflection map is indexed by the surface normal direction [Mil84]. The technique can be generalized for transparency as well, indexing by the refracted ray direction [Kay79]. In the special case that all surfaces have the same reflectance and they are viewed orthographically the total reflected intensity is a function of surface orientation only, so the diffuse and specular maps can be merged into one [Hor81]. Efficient filtering is especially important for illumination mapping, where high surface curvature often necessitates broad areas of the sky to be averaged.

Illumination mapping facilitates the simulation of complex lighting environments, since the time required to shade a point is independent of the number of light sources. There are other reasons for its recent popularity: it is one of the few demonstrated techniques for antialiasing highlights [Wil83], and it is an inexpensive approximation to ray tracing for mirror reflection and to radiosity methods [Gor84] for diffuse reflection of objects in the environment.

MAPPING

The mapping from texture space to screen space is split into two phases, as shown in figure 1. First is the surface parameterization that maps texture space to object space, followed by the standard modeling and viewing transformations that map object space to screen space, typically with a perspective projection [Fol82]. These two mappings are composed to find the overall 2-D texture space to 2-D screen space mapping, and the intermediate 3-D space is often forgotten. This simplification suggests texture mapping’s close ties with image warping and geometric distortion.

Scanning Order

There are three general approaches to drawing a texture mapped surface: a scan in screen space, a scan in texture space, and two-pass methods. The three algorithms are outlined below:

```
SCREEN SCANNING:
    for y
        for x
            compute u(x,y) and v(x,y)
            copy TEX[u,v] to SCR[x,y]

TEXTURE SCANNING:
    for v
        for u
            compute x(u,v) and y(u,v)
            copy TEX[u,v] to SCR[x,y]

TWO-PASS:
    for v
        for u
            compute x(u,v)
            copy TEX[u,v] to TEMP[x,v]
    for x
        for v
            compute y(x,v)
            copy TEMP[x,v] to SCR[x,y]
```

where TEX is the texture array, SCR is the screen array, and TEMP is a temporary array. Note that copying pixels involves filtering.

Screen order, sometimes called inverse mapping, is the most common method. For each pixel in screen space, the pre-image of the pixel in texture space is found and this area is filtered. This method is preferable when the screen must be written sequentially (e.g. when output is going to a film recorder), the mapping is readily invertible, and the texture is random access.

Texture order may seem simpler than screen order, since inverting the mapping is unnecessary in this case, but doing texture order correctly requires subtlety. Unfortunately, uniform sampling of texture space does not guarantee uniform sampling of screen space except for affine (linear) mappings. Thus, for non-affine mappings texture subdivision must often be done adaptively; otherwise, holes or overlaps will result in screen space. Scanning the texture is preferable when the texture to screen mapping is difficult to invert, or when the texture image must be read sequentially (for example, from tape) and will not fit in random access memory.

Two-pass methods decompose a 2-D mapping into two 1-D mappings, the first pass applied to the rows of an image and the second pass applied to the columns [Cat80]. These methods work particularly well for affine and perspective mappings, where the warps for each pass are linear or rational linear functions. Because the mapping and filter are 1-D they are amenable to stream processing techniques such as pipelining. Two-pass methods are preferable when the source image cannot be accessed randomly but it has rapid row and column access, and a buffer for the intermediate image is available.

Parameterization

Mapping a 2-D texture onto a surface in 3-D requires a parameterization of the surface. This comes naturally for surfaces that are defined parametrically, such as bicubic patches, but less naturally for other surfaces such as polygons and quadrics, which are usually defined implicitly. The parameterization can be by surface coordinates u and v, as in standard texture mapping, by the direction of a normal vector or light ray, as in illumination mapping, or by spatial coordinates x_o, y_o, and z_o for objects that are to appear carved out of a solid material. Solid textures can be modeled using arbitrary 3-D functions [Per85a] or by sweeping a 2-D function through space [Pea85], [Bie86].

Parameterizing Planes and Polygons

We will focus our attention on the simplest surfaces for texture mapping: planar polygons. First we discuss the parameterization and later we consider the composite mapping.

A triangle is easily parameterized by specifying the texture space coordinates (u, v) at each of its three vertices. This defines an affine mapping between texture space and 3-D object space; each of x_o, y_o, and z_o has the form $Au + Bv + C$. For polygons with more than three sides, nonlinear functions are needed in general, and one must decide whether the flexibility is worth the expense. The alternative is to assume linear parameterizations and subdivide into triangles where necessary.

One nonlinear parameterization that is sometimes used is the bilinear patch:

$$[x_o \;\; y_o \;\; z_o] = [uv \;\; u \;\; v \;\; 1] \begin{bmatrix} A & E & I \\ B & F & J \\ C & G & K \\ D & H & L \end{bmatrix}$$

which maps rectangles to planar or nonplanar quadrilaterals [Hou83]. This parameterization has the strange property of preserving lines and equal spacing along vertical and horizontal texture axes, but preserves neither along diagonals. The use of this parameterization for planar quadrilaterals is not recommended, however, since inverting it requires the solution of quadratic equations.

A better parameterization for planar quadrilaterals is the ‘perspective mapping’ [Hec83]:

$$[x_o w_o \;\; y_o w_o \;\; z_o w_o \;\; w_o] = [u \;\; v \;\; 1] \begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \end{bmatrix}$$

where w_o is the homogeneous coordinate that is divided through to calculate the true object space coordinates [Rob66], [Fol82]. Object coordinates x_o, y_o, and z_o are thus of the form $(Au + Bv + C)/(Ju + Kv + L)$. The perspective mapping preserves lines at all orientations but sacrifices equal spacing. Affine mappings are the subset of perspective matrices for which $J = K = 0$. Note that a perspective mapping might be used for the parameterization whether or not the viewing projection is perspective.

Projecting Polygons

Orthographic Projection

Orthographic projections of linearly-parameterized planar textures have an affine composite mapping. The inverse of this mapping is affine as well, of course. This makes them particularly easy to scan in screen order: the cost is only two adds per pixel, disregarding filtering [Smi80]. It is also possible to perform affine mappings by scanning the texture, producing the screen image in non-scanline order. Most of these methods are quite ingenious.

Braccini and Marino show that by depositing the pixels of a texture scanline along the path of a Bresenham digital line, an image can be rotated or sheared [Bra80]. To fill the holes that sometimes result between adjacent lines, they draw an extra pixel at each kink in the line. This results in some redundancy. They also use Bresenham's algorithm [Fol82] in a totally different way: to resample an array. This is possible because distributing m source pixels to n screen pixels is analogous to drawing a line with slope n/m.

Weiman also uses Bresenham's algorithm for scaling but does not draw diagonally across the screen [Wei80]. Instead he decomposes rotation into four scanline operations: xscale, yscale, xshear, and yshear. He does box filtering by averaging together several phases of the scaled image.

Affine mappings can be performed with multiple pass methods in several ways. Catmull and Smith decompose such mappings into a composition of two shear-and-scale passes: one horizontal and the other vertical [Cat80]. In a simpler variation discovered by Paeth, a rotational mapping is decomposed into three passes of shears: the first horizontal, the second vertical, and the third horizontal [Pae86]. Filtering for this three pass rotate is particularly simple because resampling the scanlines involves no scaling.

Perspective Projection

A naive method for texture mapping in perspective is to linearly interpolate the texture coordinates u and v along the sides of the polygon and across each scanline, much as Gouraud or Phong shading [Rog85] is done. However, linear interpolation will never give the proper effect of nonlinear foreshortening [Smi80], it is not rotationally invariant, and the error is obvious in animation. One solution is to subdivide each polygon into many small polygons. The correct solution, however, is to replace linear interpolation with the true formula, which requires a division at each pixel. In fact, Gouraud and Phong shading in perspective, which are usually implemented with linear interpolation, share the same problem, but the errors are so slight that they're rarely noticed.

Perspective mapping of a planar texture can be expressed using homogeneous matrix notation [Hec83]:

$$[xw \quad yw \quad w] = [u \quad v \quad 1] \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix}$$

This mapping is analogous to the more familiar 3-D perspective transformation using 4x4 homogeneous matrices. The inverse of this mapping (calculated using the adjoint matrix) is of the same form:

$$[uq \quad vq \quad q] = [x \quad y \quad 1] \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} = [x \quad y \quad 1] \begin{bmatrix} EI-FH & FG-DI & DH-EG \\ CH-BI & AI-CG & BG-AH \\ BF-CE & CD-AF & AE-BD \end{bmatrix}$$

The composition of two perspective mappings is also a perspective mapping. Consequently, a plane using a perspective parameterization that is viewed in perspective will have a compound mapping of the perspective form. For screen scanning we compute u and v from x and y as follows:

$$u = \frac{ax+by+c}{gx+hy+i}, \qquad v = \frac{dx+ey+f}{gx+hy+i}$$

Perspective mapping simplifies to the affine form when $G = H = g = h = 0$, which occurs when the surface is parallel to the projection plane.

Aoki and Levine demonstrate texture mapping polygons in perspective using formulas equivalent to the above [Aok78]. Smith proves that the division is necessary in general, and shows how u and v can be calculated incrementally from x and y as a polygon is scanned [Smi80]. As discussed earlier, Catmull and Smith decompose perspective mappings into two passes of shears and scales [Cat80]. Gangnet, Perny, and Coueignoux explore an alternate decomposition that rotates screen and texture space so that the perspective occurs along one of the image axes [Gan82].

Heckbert promotes the homogeneous matrix notation for perspective texture mapping and discusses techniques for scanning in screen space [Hec83]. Since the formulas for u and v above are quotients of linear expressions, they can be computed incrementally at a cost of 3 adds and 2 divides per screen pixel. Heckbert also discusses methods for computing the 3x3 matrices. Since they are homogeneous, all scalar multiples of these matrices are equivalent. If we arbitrarily choose $i = 1$ this leaves 8 degrees of freedom. These 8 values can be computed empirically by solving an 8x8 system of linear equations, which is defined by the texture and screen coordinates of four points in the plane.

Patches

Texture mapping is quite popular for surfaces modeled from patches, probably for two reasons: (a) the parameterization comes for free, (b) the cost of texture mapping is small relative to the cost of patch rendering. Patches are usually rendered using a subdivision algorithm whereby screen and texture space areas are subdivided in parallel [Cat74]. As an alternative technique, Catmull and Smith demonstrate, theoretically at least, that it is possible to perform texture mapping on bilinear, biquadratic, and bicubic patches with two-pass algorithms [Cat80]. Shantz and Fraser, Schowengerdt, and Briggs explore a similar method for the purpose of geometric image distortions [Sha82], [Fra85]. Two-pass algorithms require 1-D space variant texture filters.

FILTERING

After the mapping is computed and the texture is warped, the image must be resampled on the screen grid. This process is called *filtering*.

The cheapest texture filtering method is point sampling, wherein the pixel nearest the desired sample point is used. It works relatively well on unscaled images, but for stretched images the texture pixels are visible as large blocks, and for shrunken images aliasing can cause distracting moire patterns.

Aliasing

Aliasing can result when a signal has unreproducible high frequencies [Cro77], [Whi81]. In texture mapping, it is most noticeable on high contrast, high frequency textures. Rather than accept the aliasing that results from point sampling or avoid those models which exhibit it, we prefer a high quality, robust image synthesis system that does the extra work required to eliminate it. In practice, total eradication of aliasing is often impractical, and we must settle for approximations that merely reduce it to unobjectionable levels.

Two approaches to the aliasing problem are:

a) Point sample at higher resolution
b) Low pass filter before sampling

The first method theoretically implies sampling at a resolution determined by the highest frequencies present in the image. Since a surface viewed obliquely can create arbitrarily high frequencies, this resolution can be extremely high. It is therefore desirable to limit dense supersampling to regions of high frequency and high contrast by adapting the sampling rate to the local intensity variance [Lee85]. Whether adaptive or uniform point sampling is used, stochastic sampling can improve the appearance of images significantly by trading off aliasing for noise [Coo86].

The second method, low pass filtering before sampling, is preferable because it addresses the causes of aliasing rather than its symptoms. To eliminate aliasing our signals must be band-limited (contain no frequencies above the Nyquist limit). When a signal is warped and resampled the following steps must theoretically be performed [Smi83]:

1. reconstruct continuous signal from input samples by convolution
2. warp the abscissa of the signal
3. low pass filter the signal using convolution
4. resample the signal at the output sample points

These methods are well understood for linear warps, where the theory of linear systems lends support [Opp75]; but for nonlinear warps such as perspective, the theory is lacking, and a number of approximate methods have sprung up.

Space Invariant Filtering

For affine image warps the filter is *space invariant*; the filter shape remains constant as it moves across the image. The four steps above simplify to:

1. low pass filter the input signal using convolution
2. warp the abscissa of the signal
3. resample the signal at the output sample points

Space invariant filtering is often done using an FFT, a multiply, and an inverse FFT [Opp75]. The cost of this operation is independent of the filter size.

Space Variant Filtering

Nonlinear mappings require *space variant* filters, whose shape varies as they move across the image. Space variant filters are more complex and less well understood than space invariant filters.

In general, a square screen pixel that intersects a curved surface has a curvilinear quadrilateral pre-image in texture space. Most methods approximate the true mapping by the locally tangent perspective or linear mapping, so that the curvilinear pre-image is approximated by a quadrilateral or parallelogram. If pixels are instead regarded as circles, their pre-images are ellipses. This model is often simpler because an ellipse has five degrees of freedom (two for position, two for radii, and one for angle), while a quadrilateral has eight. Figure 2 shows the pre-images of several square screen pixels and illustrates how pixels near a horizon or silhouette require filtering of large, highly elongated areas. One of texture filtering's greatest challenges is finding efficient algorithms for filtering fairly general areas (such as arbitrarily oriented ellipses or quadrilaterals).

The cross sectional shape of the filter is also important. Theoretically we should use the ideal low pass filter, $sinc(x) = \sin(\pi x)/\pi x$, but its infinite width makes it impractical for computation. In practice we must use a finite impulse response (FIR) filter [Opp75]. Commonly used FIR filters are the box, triangle, cubic b-spline, and truncated gaussian (figure 3).

Most algorithms for texture filtering allow filtering requests to be made in arbitrary order. Such a random access capability is important for applications such as reflection mapping or ray tracing, which produce widely scattered requests.

Methods for random access space variant filtering can be categorized as follows:

1. direct convolution
2. prefiltering (pyramid or integrated array)
3. fourier series

Direct Convolution

The most straightforward filtering technique is *direct convolution*, which directly computes a weighted average of texture samples. We now summarize several direct convolution methods.

Catmull, 1974

In his subdivision patch renderer, Catmull computes an unweighted average of the texture pixels corresponding to each screen pixel [Cat74]. He gives few details, but his filter appears to be a quadrilateral with a box cross section.

Blinn and Newell, 1976

Blinn and Newell improve on this with a triangular filter that forms overlapping square pyramids 2 pixels wide in screen space [Bli76]. At each pixel the pyramid is distorted to fit the approximating parallelogram in texture space, and a weighted average is computed.

Feibush, Levoy, and Cook, 1980

The filter used by Feibush, Levoy, and Cook is more elaborate [Fei80].

The following steps are taken at each screen pixel:

(1) Center the filter function (box, cylinder, cone, or gaussian) on the pixel and find its bounding rectangle.

(2) Transform the rectangle to texture space, where it is a quadrilateral. The sides of this quadrilateral are assumed to be straight. Find a bounding rectangle for this quadrilateral.

(3) Map all pixels inside the texture space rectangle to screen space.

(4) Form a weighted average of the mapped texture pixels using a two-dimensional lookup table indexed by each sample's location within the pixel.

Since the filter function is in a lookup table it can be a gaussian or other high quality filter.

Gangnet, Perny, and Coueignoux, 1982

The texture filter proposed by Gangnet, Perny, and Coueignoux is quite similar to the method of Feibush et. al., but they subdivide uniformly in screen space rather than texture space [Gan82].

Pixels are assumed to be circular and overlapping. The pre-image of a screen circle is a texture ellipse whose major axis corresponds to the direction of greatest compression. A square intermediate supersampling grid that is oriented orthogonally to the screen is constructed. The supersampling rate is determined from the longest diagonal of the parallelogram approximating the texture ellipse. Each of the sample points on the intermediate grid is mapped to texture space, and bilinear interpolation is used to reconstruct the texture values at these sample points. The texture values are then weighted by a truncated *sinc* 2 pixels wide in screen space and summed.

The paper contrasts Feibush's ‘‘back transforming’’ method with their own ‘‘direct transforming’’ method, claiming that the latter produces more accurate results because the sampling grid is in screen space rather than in texture space. Other differences are more significant. For example, Gangnet's method requires a bilinear interpolation for each sample point, while Feibush's method does not. Also, Gangnet samples at an unnecessarily high frequency along the minor axis of the texture ellipse. For these two reasons, Feibush's algorithm is probably faster than Gangnet's.

Greene and Heckbert, 1986

The elliptical weighted average filter (EWA) proposed by Heckbert [Gre86b] is similar to Gangnet's method in that it assumes overlapping circular pixels that map to arbitrarily oriented ellipses, and it is like Feibush's because the filter function is stored in a lookup table, but instead of mapping texture pixels to screen space, the filter is mapped to texture space. The filter shape, a circularly symmetric function in screen space, is warped by an elliptic paraboloid function into an ellipse in texture space. The elliptic paraboloid is computed incrementally and used for both ellipse inclusion testing and filter table index. The cost per texture pixel is just a few arithmetic operations, in contrast to Feibush's and Gangnet's methods, which both require mapping each pixel from texture space to screen space, or vice versa.

Comparison of Direct Convolution Methods

All five methods have a cost per screen pixel proportional to the number of texture pixels accessed, and this cost is highest for Feibush and Gangnet. Since the EWA filter is comparable in quality to these other two techniques at much lower cost, it appears to be the fastest algorithm for high quality direct convolution.

Prefiltering

Even with optimization, the methods above are often extremely slow, since a pixel pre-image can be arbitrarily large along silhouettes or at the horizon of a textured plane. Horizon pixels can easily require the averaging of thousands of texture pixels. We would prefer a texture filter whose cost does not grow in proportion to texture area.

To speed up the process, the texture can be prefiltered so that during rendering only a few samples will be accessed for each screen pixel. The access cost of the filter will thus be constant, unlike direct convolution methods. Two data structures have be used for prefiltering: image pyramids and integrated arrays.

Pyramid data structures are commonly used in image processing and computer vision [Tan75], [Ros84]. Their application to texture mapping was apparently first proposed in Catmull's PhD work.

Several texture filters that employ prefiltering are summarized below. Each method makes its own trade-off between speed and filter quality.

pyramid − Dungan, Stenger, and Sutty, 1978

Dungan, Stenger, and Sutty prefilter their texture ''tiles'' to form a pyramid whose resolutions are powers of two [Dun78]. To filter an elliptical texture area one of the pyramid levels is selected on the basis of the average diameter of the ellipse and that level is point sampled. The memory cost for this type of texture pyramid is $1 + 1/4 + 1/16 + \cdots = 4/3$ times that required for an unfiltered texture. Others have used four dimensional image pyramids which prefilter differentially in u and v to resolutions of the form $2^{\Delta u} \times 2^{\Delta v}$.

pyramid − Heckbert, 1983

Heckbert discusses Williams' trilinear interpolation scheme for pyramids (see below) and its efficient use in perspective texture mapping of polygons [Hec83]. Choosing the pyramid level is equivalent to approximating a texture quadrilateral with a square, as shown in figure 4a. The recommended formula for the diameter d of the square is the maximum of the side lengths of the quadrilateral. Aliasing results if the area filtered is too small, and blurring results if it's too big; one or the other is inevitable (it is customary to err on the blurry, conservative side).

pyramid − Williams, 1983

Williams improves upon Dungan's point sampling by proposing a trilinear interpolation scheme for pyramidal images wherein bilinear interpolation is performed on two levels of the pyramid and linear interpolation is performed between them [Wil83]. The output of this filter is thus a continuous function of position (u, v) and diameter d. His filter has a constant cost of 8 pixel accesses and 7 multiplies per screen pixel. Williams uses a box filter to construct the image pyramid, but gaussian filters can also be used [Bur81]. He also proposes a particular layout for color image pyramids called the ''mipmap''.

EWA on pyramid − Greene and Heckbert, 1986

Attempting to decouple the data structure from the access function, Greene suggests the use of the EWA filter on an image pyramid [Gre86b]. Unlike the other prefiltering techniques such as trilinear interpolation on a pyramid or the summed area table, EWA allows arbitrarily oriented ellipses to be filtered, yielding higher quality.

summed area table − Crow, 1984; Ferrari and Sklansky, 1984

Crow proposes the *summed area table*, an alternative to the pyramidal filtering of earlier methods, which allows orthogonally oriented rectangular areas to be filtered in constant time [Cro84]. The original texture is pre-integrated in the u and v directions and stored in a high-precision summed area table. To filter a rectangular area the table is sampled in four places (much as one evaluates a definite integral by sampling an indefinite integral). To do this without artifacts requires 16 accesses and 14 multiplies in general, but an optimization for large areas cuts the cost to 4 accesses and 2 multiplies. The high-precision table requires 2 to 4 times the memory cost of the original image.

The method was developed independently by Ferrari and Sklansky, who show how convolution with arbitrary rectilinear polygons can easily be performed using a summed area table [Fer84], [Fer85].

The summed area table is generally more costly than the texture pyramid in both memory and time, but it can perform better filtering than the pyramid since it filters rectangular areas, not just squares, as shown in figure 4b. Glassner has shown how the rectangle can be successively refined to approximate a quadrilateral [Gla86]. As an optimization he also suggests that this refinement be done only in areas of high local texture variance. To compute variance over a rectangle he pre-computes variances of 3x3 areas and forms a summed area table of them. A more accurate method for computing rectangle variance would be to precompute summed area tables of the texture values and their squares. Using these two tables variance can be computed as $\sigma^2 = \Sigma x_i^2/n - (\Sigma x_i/n)^2$. Of course there is considerable overhead in time and memory to access a variance table.

repeated integration filtering − Perlin, 1985; Heckbert, 1986; Ferrari et. al., 1986

This elegant generalization of the summed area table was developed independently by Perlin and by Ferrari et. al. [Per85b], [Hec86], [Fer86]. If an image is pre-integrated in u and v n times, an orthogonally oriented elliptical area can be filtered by sampling the array at $(n+1)^2$ points and weighting them appropriately. The effective filter is a box convolved with itself n times (an nth order b-spline) whose size can be selected at each screen pixel. If $n = 0$ the method degenerates to point sampling, if $n = 1$ it is equivalent to the summed area table with its box filter, $n = 2$ uses a triangular filter, and $n = 3$ uses a parabolic filter. As n increases, the filter shape approaches a gaussian, and the memory and time costs increase. The method can be generalized to arbitrary spline filters [Hec86]. One difficulty with the method is that high precision arithmetic (more than 32 bits) is required for $n \geq 2$.

Comparison of Prefiltering Methods

The following table summarizes the prefiltering methods we have discussed:

REFERENCE	FILTER	SHAPE	DOF	TIME	MEMORY
point sampling	impulse	point	2	1,0	1
point sampled pyramid	box	square	3	1,0	1.33
trilinear interp. on pyramid	box	square	3	8,7	1.33
4-D pyramid (scaling u and v)	box	rectangle	4	16,15	4
EWA on pyramid	any	ellipse	5	unlimited?	1.33
summed area table	box	rectangle	4	16,14; or 4,2	2-4
repeated integration (order 2)	triangle	rectangle	4	36,31; or 9,4	2-6
direct convolution	any	any	5+	unlimited	1

The high quality direct convolution methods of Feibush and Gangnet have been included at the bottom for comparison. In this table, FILTER means cross section, while SHAPE is 2-D filter shape. The number of degrees of freedom (DOF) of the 2-D filter shape provides an approximate ranking of filter quality; the more degrees of freedom are available the greater is the filter shape control. The numbers under TIME are the number of texture pixel accesses and the number of multiplies per screen pixel. MEMORY is the ratio of memory required relative to an unfiltered texture.

We see that the integrated array techniques of Crow and Perlin have rather high memory costs relative to the pyramid methods, but allow rectangular or orthogonally oriented elliptical areas to be filtered. Traditionally pyramid techniques have lower memory cost but allow only squares to be filtered.

Since prefiltering usually entails a setup expense proportional to the square of the texture resolution, its cost is proportional to that of direct convolution − if the texture is only used once. But if the texture is used many times, as part of a periodic pattern, or appearing on several objects or in several frames of animation, the setup cost can be amortized over each use.

Figure 5 illustrates several of these texture filters on a checkerboard in perspective, which is an excellent test of a texture filter, since it has a range of frequencies at high contrast. Note that trilinear interpolation on a pyramid blurs excessively parallel to the horizon but that integrated arrays do not. Second order repeated integration does just as well as EWA for vertically oriented texture ellipses (on the left of the images), but there is a striking difference between gaussian EWA and the other prefiltering methods for ellipses at 45 degrees (on the right of the images).

Fourier Series

An alternative to texture space filtering is to transform the texture to frequency space and low pass filter its spectrum. This is most convenient when the texture is represented by a Fourier series rather than a texture array. Norton, Rockwood, and Skolmoski explore this approach for flight simulator applications and propose a simple technique for clamping high frequency terms [Nor82]. Gardner employs 3-D Fourier series as a transparency texture function, with which he generates surprisingly convincing pictures of trees and clouds [Gar85].

Perlin's ''Image Synthesizer'' uses band limited pseudo-random functions as texture primitives [Per85a]. Creating textures in this way eases transitions from macroscopic to microscopic views of a surface; in the macroscopic range the surface characteristics are built into the scattering statistics of the illumination model, in the intermediate range they are modeled using bump mapping, and in the microscopic range the surface is explicit geometry [Per84], [Kaj85]. Each term in the texture series can make the transition independently at a scale appropriate to its frequency range.

Filtering Recommendations

The best filtering algorithm for a given task depends on the texture representation in use. When filtering a texture array, the minimal texture filter should be trilinear interpolation on a pyramid. This blurs excessively, but it eliminates most aliasing and costs little more than point sampling. Next in quality and cost are integrated arrays of order 1 (box filter) or order 2 (triangle filter). These allow elongated vertical or horizontal areas to be filtered much more accurately (resulting in less blurring at horizons and silhouettes) but blur is still excessive for elongated areas at an angle (such as an eccentric ellipse at 45 degrees). For near-perfect quality at a cost far below that of direct convolution, the EWA or similar filter on a pyramid is recommended. Its cost is not strictly bounded like most prefiltering methods, but is proportional to ellipse eccentricity. The cost of direct convolution, meanwhile, is proportional to ellipse area. Any of these methods could probably benefit from adaptive refinement according to local texture variance [Gla86].

In the case of arbitrary texture functions, which can be much harder to integrate than texture arrays, adaptive stochastic sampling methods are called for [Lee85].

Future research on texture filters will continue to improve their quality by providing greater filter shape control while retaining low time and memory costs. One would like to find a constant-cost method for high quality filtering of arbitrarily oriented elliptical.

SYSTEM SUPPORT FOR TEXTURE MAPPING

So far we have emphasized those tasks common to all types of texture mapping. We now summarize some of the special provisions that a modeling and rendering system must make in order to support different varieties of texture mapping. We assume that the rendering program is scanning in screen space.

The primary requirements of standard texture mapping are texture space coordinates (u, v) for each screen pixel plus the partial derivatives of u and v with respect to screen x and y for good antialiasing. If the analytic formulas for these partials are not available, they can be approximated by differencing the u and v of neighboring pixels.

Bump mapping requires additional information at each pixel: two vectors tangent to the surface pointing in the u and v directions. For facet shaded polygons these tangents are constant across the polygon, but for Phong shaded polygons they vary. In order to ensure artifact-free bump mapping on Phong shaded polygons, these tangents must be continuous across polygon seams. One way to guarantee this is to compute tangents at all polygon vertices during model preparation and interpolate them across the polygon. The normal vector can be computed as the cross product of the tangents.

Proper antialiasing of illumination mapping requires some measure of surface curvature in order to calculate the solid angle of sky to filter. This is usually provided in the form of the partials of the normal vector with respect to screen space. When direct rendering support for illumination mapping is unavailable, however, tricks can be employed that give a visually acceptable approximation. Rather than calculate the exact ray direction at each pixel, one can compute the reflected or refracted ray direction at polygon vertices only and interpolate it, in the form of u and v texture indices, across the polygon using standard methods.

Although texture maps are usually much more compact than brute force 3-D modeling of surface details, they can be bulky, especially when they represent a high resolution image as opposed to a low resolution texture pattern that is replicated numerous times. Keeping several of these in random access memory is often a burden on the rendering program. This problem is especially acute for rendering algorithms that generate the image in scanline order rather than object order, since a given scanline could access hundreds of texture maps. Further work is needed on memory management for texture map access.

SUMMARY

Texture mapping has become a widely used technique because of its generality and efficiency. It has even made its way into everyday broadcast TV, thanks to new real-time video texture mapping hardware such as the Ampex *ADO* and Quantel *Mirage*. Rendering systems of the near future will allow any conceivable surface parameter to be texture mapped. Despite the recent explosion of diverse applications for texture mapping, a common set of fundamental concepts and algorithms is emerging. We have surveyed a number of these fundamentals: alternative techniques for parameterization, scanning, texture representation, direct convolution, and prefiltering. Further work is needed on quantifying texture filter quality and collecting theoretical and practical comparisons of various methods.

ACKNOWLEDGEMENTS

My introduction to textures came while mapping Aspen's facades for Walter Bender's *Quick and Dirty Animation System (QADAS)* at the Massachusetts Institute of Technology's Architecture Machine Group; we learned the hard way that point sampling and linear u, v aren't good enough. I did most of the research that evolved into this paper while at New York Institute of Technology's Computer Graphics Lab, where Pat Hanrahan and Mike Chou contributed many ideas and Ned Greene, Jules Bloomenthal, and Lance Williams supplied the cheeseburger incentives. Thanks also to Rob Cook at Pixar for his encouragement, and to all the good folks at Pacific Data Images.

REFERENCES

Recommended reading: Rogers' book [Rog85] is an excellent introduction to image synthesis. For a good introduction to texture mapping see Blinn and Newell's paper [Bli76]. Smith [Smi83] gives a helpful theoretical/intuitive introduction to digital image filtering, and Feibush et. al. [Fei80] go into texture filtering in detail. Two pass methods are described by Catmull and Smith [Cat80]. The best references on bump mapping and illumination mapping are by Blinn [Bli78a] and Miller & Hoffman [Mil84], respectively. Systems aspects of texture mapping are discussed by Cook [Coo84] and Perlin [Per85a].

[Aok78] Masayoshi Aoki, Martin D. Levine, "Computer Generation of Realistic Pictures", *Computers and Graphics*, Vol. 3, 1978, pp. 149-161.

[Bie86] Eric A. Bier, Kenneth R. Sloan, Jr., "Two-Part Texture Mappings", *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, Sept. 1986, pp. 40-53.

[Bli76] James F. Blinn, Martin E. Newell, "Texture and Reflection in Computer Generated Images", *CACM*, Vol. 19, No. 10, Oct. 1976, pp. 542-547.

[Bli78a] James F. Blinn, "Simulation of Wrinkled Surfaces", *Computer Graphics*, (SIGGRAPH '78 Proceedings), Vol. 12, No. 3, Aug. 1978, pp. 286-292.

[Bli78b] James F. Blinn, "Computer Display of Curved Surfaces", PhD thesis, CS Dept., U. of Utah, 1978.

[Bra80] Carlo Braccini, Giuseppe Marino, "Fast Geometrical Manipulations of Digital Images", *Computer Graphics and Image Processing*, Vol. 13, 1980, pp. 127-141.

[Bur81] Peter J. Burt, "Fast Filter Transforms for Image Processing", *Computer Graphics and Image Processing*, Vol. 16, 1981, pp. 20-51.

[Car85] Richard J. Carey, Donald P. Greenberg, "Textures for Realistic Image Synthesis", *Computers and Graphics*, Vol. 9, No. 2, 1985, pp. 125-138.

[Cat74] Ed Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD thesis, Dept. of CS, U. of Utah, Dec. 1974.

[Cat80] Ed Catmull, Alvy Ray Smith, "3-D Transformations of Images in Scanline Order", *Computer Graphics*, (SIGGRAPH '80 Proceedings), Vol. 14, No. 3, July 1980, pp. 279-285.

[Coo84] Robert L. Cook, "Shade Trees", *Computer Graphics*, (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 223-231.

[Coo86] Robert L. Cook, "Antialiasing by Stochastic Sampling", to appear, *ACM Transactions on Graphics*, 1986.

[Cro77] Franklin C. Crow, "The Aliasing Problem in Computer-Generated Shaded Images", *CACM*, Vol. 20, Nov. 1977, pp. 799-805.

[Cro84] Franklin C. Crow, "Summed-Area Tables for Texture Mapping", *Computer Graphics*, (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 207-212.

[Dun78] William Dungan, Jr., Anthony Stenger, George Sutty, "Texture Tile Considerations for Raster Graphics", *Computer Graphics*, (SIGGRAPH '78 Proceedings), Vol. 12, No. 3, Aug. 1978, pp. 130-134.

[Fei80] Eliot A. Feibush, Marc Levoy, Robert L. Cook, "Synthetic Texturing Using Digital Filters", *Computer Graphics*, (SIGGRAPH '80 Proceedings), Vol. 14, No. 3, July 1980, pp. 294-301.

[Fer84] Leonard A. Ferrari, Jack Sklansky, "A Fast Recursive Algorithm for Binary-Valued Two-Dimensional Filters", *Computer Vision, Graphics, and Image Processing*, Vol. 26, No. 3, June 1984, pp. 292-302.

[Fer85] Leonard A. Ferrari, Jack Sklansky, "A Note on Duhamel Integrals and Running Average Filters", *Computer Vision, Graphics, and Image Processing*, Vol. 29, Mar. 1985, pp. 358-360.

[Fer86] Leonard A. Ferrari, P. V. Sankar, Jack Sklansky, Sidney Leeman, "Efficient Two-Dimensional Filters Using B-Spline Approximations", to appear, *Computer Vision, Graphics, and Image Processing*.

[Fol82] James D. Foley, Andries van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

[Fra85] Donald Fraser, Robert A. Schowengerdt, Ian Briggs, "Rectification of Multichannel Images in Mass Storage Using Image Transposition", *Computer Vision, Graphics, and Image Processing*, Vol. 29, No. 1, Jan. 1985, pp. 23-36.

[Gan82] Michel Gangnet, Didier Perny, Philippe Coueignoux, "Perspective Mapping of Planar Textures", *Eurographics '82*, 1982, pp. 57-71, (slightly superior to the version that appeared in *Computer Graphics*, Vol. 16, No. 1, May 1982).

[Gar85] Geoffrey Y. Gardner, "Visual Simulation of Clouds", *Computer Graphics*, (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 297-303.

[Gla86] Andrew S. Glassner, "Adaptive Precision in Texture Mapping", *Computer Graphics*, (SIGGRAPH '86 Proceedings), Vol. 20, No. 4, Aug. 1986, pp. 297-306.

[Gor84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics*, (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 213-222.

[Gre86a]
Ned Greene, "Environment Mapping and Other Applications of World Projections", *Graphics Interface '86*, May 1986, pp. 108-114.

[Gre86b]
Ned Greene, Paul S. Heckbert, "Creating Raster Omnimax Images from Multiple Perspective Views Using The Elliptical Weighted Average Filter", *IEEE Computer Graphics and Applications*, Vol. 6, No. 6, June, 1986, pp. 21-27.

[Hec83] Paul S. Heckbert, "Texture Mapping Polygons in Perspective", Technical Memo No. 13, NYIT Computer Graphics Lab, April 1983.

[Hec86] Paul S. Heckbert, "Filtering by Repeated Integration", *Computer Graphics*, (SIGGRAPH '86 Proceedings), Vol. 20, No. 4, Aug. 1986, pp. 317-321.

[Hor81] Berthold K. P. Horn, "Hill Shading and the Reflectance Map", *Proc. IEEE*, Vol. 69, No. 1, Jan. 1981, pp. 14-47.

[Hou83] J. C. Hourcade, A. Nicolas, "Inverse Perspective Mapping in Scanline Order onto Non-Planar Quadrilaterals", *Eurographics '83*, 1983, pp. 309-319.

[Kaj85] James T. Kajiya, "Anisotropic Reflection Models", *Computer Graphics*, (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 15-21.

[Kay79] Douglas S. Kay, Donald P. Greenberg, "Transparency for Computer Synthesized Images", *Computer Graphics*, (SIGGRAPH '79 Proceedings), Vol. 13, No. 2, Aug. 1979, pp. 158-164.

[Lee85] Mark E. Lee, Richard A. Redner, Samuel P. Uselton, ''Statistically Optimized Sampling for Distributed Ray Tracing'', *Computer Graphics*, (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 61-67.

[Mil84] Gene S. Miller, C. Robert Hoffman, ''Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments'', *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*, July 1984.

[Nor82] Alan Norton, Alyn P. Rockwood, Philip T. Skolmoski, ''Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space'', *Computer Graphics*, (SIGGRAPH '82 Proceedings), Vol. 16, No. 3, July 1982, pp. 1-8.

[Opp75] Alan V. Oppenheim, Ronald W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[Pae86] Alan W. Paeth, ''A Fast Algorithm for General Raster Rotation'', *Graphics Interface '86*, May 1986, pp. 77-81.

[Pea85] Darwyn R. Peachey, ''Solid Texturing of Complex Surfaces'', *Computer Graphics*, (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 279-286.

[Per84] Ken Perlin, ''A Unified Texture/Reflectance Model'', *SIGGRAPH '84 Advanced Image Synthesis seminar notes*, July 1984.

[Per85a] Ken Perlin, ''An Image Synthesizer'', *Computer Graphics*, (SIGGRAPH '85 Proceedings), Vol. 19, No. 3, July 1985, pp. 287-296.

[Per85b] Ken Perlin, ''Course Notes'', *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, July 1985.

[Rob66] Lawrence G. Roberts, *Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs*, MS-1045, Lincoln Lab, Lexington, MA, July 1966.

[Rog85] David F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.

[Ros84] A. Rosenfeld, *Multiresolution Image Processing and Analysis*, Springer, Berlin, 1984.

[Sha82] Michael Shantz, ''Two Pass Warp Algorithm for Hardware Implementation'', *Proc. SPIE, Processing and Display of Three Dimensional Data*, Vol. 367, 1982, pp. 160-164.

[Smi80] Alvy Ray Smith, ''Incremental Rendering of Textures in Perspective'', *SIGGRAPH '80 Animation Graphics seminar notes*, July 1980.

[Smi83] Alvy Ray Smith, ''Digital Filtering Tutorial for Computer Graphics'', parts 1 and 2, *SIGGRAPH '83 Introduction to Computer Animation seminar notes*, July 1983, pp. 244-261, 262-272.

[Tan75] S. L. Tanimoto, Theo Pavlidis, ''A Hierarchical Data Structure for Picture Processing'', *Computer Graphics and Image Processing*, Vol. 4, No. 2, June 1975, pp. 104-119.

[Wei80] Carl F. R. Weiman, ''Continuous Anti-Aliased Rotation and Zoom of Raster Images'', *Computer Graphics*, (SIGGRAPH '80 Proceedings), Vol. 14, No. 3, July 1980, pp. 286-293.

[Whi81] Turner Whitted, ''The Causes of Aliasing in Computer Generated Images'', *SIGGRAPH '81 Advanced Image Synthesis seminar notes*, Aug. 1981.

[Wil83] Lance Williams, ''Pyramidal Parametrics'', *Computer Graphics*, (SIGGRAPH '83 Proceedings), Vol. 17, No. 3, July 1983, pp. 1-11.