SURVEY OF TEXTURE MAPPING

Paul S. Heckbert

Computer Graphics Lab
New York Institute of Technology †

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 which warps a texture onto a surface, and the filtering which is necessary in order to avoid aliasing. An extensive bibliography is included.

KEYWORDS: texture mapping, texture filter, space variant filter, antialiasing.

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 filters 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 which is repeated many times to tile the plane, or more generally, a multidimensional image which 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 either by 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 (x_o,y_o,z_o), and screen space is (x,y).

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

Uses for Texture Mapping

The possible uses for mapped texture are myriad. Some of the parameters which have been texture mapped to date are: surface color (the most common use) [Cat74], specular reflection [Bli76], normal vector perturbation ("bump mapping") [Bli78a], specularity (the glossiness coefficient) [Bli78b], transparency [Smi79], diffuse reflection [Mil84], surface displacement and mixing coefficients [Coo84b].

Illumination Mapping

Mapping specular and diffuse reflection is rather different from mapping other parameters, since these maps are not associated with a particular object in the scene, but to 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 the 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].

Illumination mapping, as these techniques are called, facilitates the simulation of complex lighting environments, since the time required to shade a point is independent of the number of light sources. Other reasons for its recent popularity are: it is one of the few demonstrated techniques for antialiasing highlights [Wil83], it is an inexpensive approximation to ray tracing for mirror reflection, and to radiosity methods [Gor84] for diffuse reflection of objects in the environment. Efficient filtering is especially important for illumination mapping, where variations in surface curvature often necessitate broad areas of the sky to be averaged.

Since specular reflection varies as a function of the viewing direction, it is most conveniently computed on the fly, as in ray tracing. Diffuse reflection of the environment, however, has not yielded to ray tracing even when stochastic methods [Coo84a] are used. The problem is that diffuse reflection scatters light over an entire hemisphere, not a narrow cone, as does specular reflection. Fortunately diffuse reflection is independent of viewing direction, so the incident illumination at each surface point can be precomputed and treated as a texture [Coo84b]. Previous methods have approximated this using polygon subdivision to model hard shadows [Ath78], soft shadows [Nis83], beams of light [Hec84], or indirect illumination [Gor84]. With the development of more efficient algorithms for its computation, incident illumination promises to be a common use for textures in the future.

Even when direct support for illumination mapping is unavailable, tricks can be employed which give a visually acceptable approximation. Rather than calculate the exact ray direction at each pixel, one can compute the reflected or refracted ray direction only at polygon vertices and interpolate it, in the form of u and v texture indices, across the polygon using standard methods. This approximation is similar to that made by beam tracing [Hec84].

† Current address: Pacific Data Images, 1111 Karlstad Dr., Sunnyvale, CA 94089, USA.

MAPPING

The mapping from texture space to screen space is split into two phases. First is the surface parameterization which maps texture space to object space, followed by the standard modeling and viewing transformations which 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: scanning in screen space, scanning in texture space, and two-pass methods.

Traversing the screen in scanline order, sometimes called inverse mapping, is the most common method. For each pixel in screen space the preimage 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.

Traversing the texture in scanline order may seem simpler than scanning the screen since inverting the mapping is unnecessary in this case, but doing this correctly is subtle. Unfortunately, uniform sampling of texture space does not guarantee uniform sampling of screen space except for affine (linear) mappings, so 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 either (a) the texture to screen mapping is difficult to invert, or (b) the texture image must be read sequentially (e.g. 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 applied to the rows of an image and the second 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 random accessed but it has rapid row column access, and a buffer for the intermediate image is available.

Parameterization

In order to map a 2-D texture onto a surface in 3-D, a parameterization of the surface is needed. This comes naturally for surfaces which 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 which are to appear carved out of a solid material.

Parameterizing Planes and Polygons

We will examine mappings for planar polygons in some detail. First we discuss the parameterization and later we discuss 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 have the form $Au+Bv+C$. For polygons with more than three sides, nonlinear functions are needed in general, and one must decide if the flexibility is worth the expense. The alternative is to assume linear parameterizations, and subdivide into triangles where necessary.

One nonlinear parameterization which 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 that it preserves 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 which is divided through to calculate the true object space coordinates [Rob66], [Fol82]. 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. 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 a linear composite mapping. The inverse of this mapping is linear 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 which 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 [Bre65] in a totally different way: to scale an image. This is possible because distributing m source pixels to n screen pixels is analogous to drawing a line with slope n/m. Braccini and Marino use the simplest filtering: point sampling.

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.

Cohen draws texture scanlines diagonally across the screen like [Bra80], but does not use their scaling trick [Coh84]. He is able, however, to eliminate the holes and redundancy of [Bra80] by carefully nesting the digital lines. Cohen also demonstrates the algorithm's applicability to antialiased line drawing.

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 scan line, much as Gouraud or Phong shading [Fol82] 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 ones. 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 an affine or perspective parameterized plane is:

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

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, as is the composition of two of these mappings. Consequently a plane using a perspective parameterization which is viewed in perspective will have compound mapping which is of the perspective form. The perspective mapping simplifies to the affine form when G and H are zero, 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 which 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 incremental techniques for scanning in screen space [Hec83].

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], [Lan80]. 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]. Fraser, Schowengerdt, and Briggs explore a similar method for the application of geometric image distortions [Fra85].

FILTERING

After the mapping is computed and the texture 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 shrunk 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 which results from point sampling or avoid those models which exhibit it, we prefer a high quality, robust image synthesis system which does the extra work required to eliminate it. In practice, total eradication of aliasing is often impractical and we must settle for approximations which 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 [Cro82] by adapting the sampling rate to the local intensity variance [Lee85], [Dip85]. This is not possible, however, in vectorized algorithms, which must choose a uniform sampling rate a priori and accept any residual aliasing. Whether adaptive or uniform point sampling are 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 out 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, 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 kernel 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 convolutions are often done using an FFT, multiply, and inverse FFT [Opp75]. The cost of this operation is independent of the kernel size.

Direct Convolution

Nonlinear mappings have space-variant filter kernels (in texture space), which require more complex filtering methods. In general, a square screen pixel which intersects a curved surface has a curvilinear quadrilateral preimage in texture space. Most methods approximate the true mapping by the locally tangent perspective or linear mapping, so that the curvilinear preimage is approximated by a quadrilateral or parallelogram. In place of the ideal low pass filter, a *sinc*, a finite impulse response (FIR) approximation is used to form a weighted average of texture samples.

We now summarize several direct convolution texture filters.

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 it appears his filter is a quadrilateral with a box kernel cross section.

Blinn and Newell, 1976

Blinn and Newell improve on this with a triangular kernel which 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 kernel (box, cylinder, cone, or gaussian) on the pixel and find its bounding rectangle.

(2) Transform the rectangle to texture space, where it is warped into 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 kernel is in 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 [Fei80], but they subdivide uniformly in screen space rather than texture space [Gan82].

Pixels are assumed circular and overlapping. The preimage of a screen circle is a texture ellipse whose major axis corresponds to the direction of greatest compression. A square intermediate supersampling grid which 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 [Fei80]'s "back transforming" method with [Gan82]'s "direct transforming" method, claiming that the latter produces more accurate results because the sampling grid is in screen space rather than texture space. Other differences are more significant. For example, [Gan82] requires a bilinear interpolation for each sample point, while [Fei80] does not. Also, [Gan82] samples at an unnecessarily high frequency along the minor axis of the texture ellipse. For these two reasons, [Fei80] is probably faster than [Gan82] (an estimate denied in [Gan84]).

Greene and Heckbert, 1986

The elliptical weighted average filter (EWA) proposed by Heckbert [Gre86b] is similar to [Gan82] in that it assumes overlapping circular pixels which map to arbitrarily oriented ellipses, and like [Fei80] because the kernel is stored in lookup table, but instead of mapping texture pixels to screen space, the kernel is mapped to texture space, as in [Bli76]. The kernel, 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 kernel table index. The cost per texture pixel is just a few arithmetic operations, in contrast to [Fei80] and [Gan82], which both require mapping each texture 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 [Fei80] and [Gan82]. Since [Gre86b] has quality comparable to [Fei80] and [Gan82] at much lower cost, it appears to be the fastest algorithm for high quality direct convolution.

Prefiltering the Texture

Even with optimization, the methods above are often extremely slow, since a pixel preimage can be arbitrarily large along silhouettes or at the horizon. We would prefer a texture filter whose cost does not grow proportionately 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 can be used for prefiltering: image pyramids and integrated arrays.

Pyramidal 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 [Smi79].

We now summarize several texture filters which employ prefiltering.

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 based on the average diameter of the ellipse and the 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; only 33% more expensive.

Smith, 1979

Smith describes the "mipmap", which is a particular layout for color image pyramids invented by Williams [Smi79]. Smith points out that the square filter area inherent in pyramids is inaccurate if the pixel preimage is elongated.

Heckbert, 1983

Heckbert describes 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. 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 of these two is inevitable.

Williams, 1983

Williams improves upon [Dun78] 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].

Gangnet and Ghazanfarpour, 1984

In Gangnet and Ghazanfarpour's survey a variation on the image pyramid is proposed which allows unequal filtering in u and v (they call it "asymmetrical" filtering, but is more properly termed "anisotropic"). The image is prefiltered to resolutions of the form $2^{\Delta u} \times 2^{\Delta v}$, so this pyramid is four dimensional: u, v, Δu and Δv. Its memory requirements are four times that of an unfiltered image, three times that of an isotropic pyramid, and the time cost is 16 texture pixel accesses and 15 multiplies per screen pixel.

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.

Crow, 1984

Crow proposes the *summed area table*, an alternative to the pyramidal filtering of [Dun78] and [Wil83], which allows orthogonally oriented rectangular areas to be filtered in constant time [Cro84]. The original texture is preintegrated 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 there is an optimization for large areas which cuts the cost to 4 accesses and 2 multiplies. The high-precision table requires 4 times the memory cost of the original image. 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. It clearly outperforms the four-dimensional pyramid in [Gan84].

Perlin, 1985

Perlin's *selective image filter* is an elegant generalization of [Cro84], developed independently [Per85b]. If an image is preintegrated 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 kernel is a box convolved with itself n times 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 kernel, $n=2$ uses a triangular kernel, and $n=3$ uses a parabolic kernel. With increasing n the kernel approaches a gaussian, and the memory and time costs increase.

Comparison of Prefiltering Methods

The following table summarizes the prefiltering methods we have discussed:

REF.	KERNEL	SHAPE	DOF	TIME	MEMORY
pt samp	impulse	point	2	1,0	1
Dun78	box	square	3	1,0	1.33
Wil83	box	square	3	8,7	1.33
Gan84	box	rectangle	4	16,15	4
Gre86b	any	ellipse	5	?	1.33
Cro84	box	rectangle	4	16,14 or 4,2	4
Per85b	triangle	ellipse	4	36,31 or 9,4	6

The pair of numbers under 'time' is the number of texture pixel accesses and the number of multiplies per screen pixel. The DOF (degrees of freedom) of the filter provides an approximate ranking of filter quality; the more degrees of freedom are available the greater is the kernel shape control.

We see that the integrated array techniques [Cro84] and [Per85b] 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 of the same order as 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.

Filtering in Frequency Space

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]. 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 and scanning order in use. When filtering a texture array in a screen order rendering system, the EWA filter [Gre86b], summed area table [Cro84], or selective image filter [Per85b] are recommended because of their good shape control and high speed. Since the above algorithms are still under development and the EWA filter has yet to be tested on a pyramid, it is too early to make definitive judgements. When the texture is a fourier series, filtering is simply a matter of clamping or truncating the high frequency terms [Nor82]. In the case of arbitrary texture functions, which can be much harder to integrate than texture arrays, adaptive stochastic sampling methods are called for [Dip85]. Two-pass algorithms require 1-D space variant texture filters.

Future research on texture filters will continue to improve their quality by providing greater kernel shape control while retaining low time and memory costs. One would like to find a constant-cost prefiltering method which filters arbitrarily oriented elliptical areas using a gaussian kernel.

CONCLUSIONS

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 which a modeling and rendering system must make in order to support different varieties of texture mapping.

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 (assuming that the rendering program is scanning in screen space).

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 [Fol82] 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 [Max86]. 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.

Although they are usually much more compact than brute force 3-D modeling of surface details, texture maps can be bulky, especially when they represent a high resolution image as opposed to a low resolution texture pattern which 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 which generate the image in scanline order rather than object order, since a given scan line could access hundreds of texture maps. Further work is needed on memory management for texture map access.

General

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.

ACKNOWLEDGEMENTS

I got my introduction to textures while mapping Aspen's facades as part of Walter Bender's *Quick and Dirty Animation System (QADAS)* at the MIT Architecture Machine Group. Pat Hanrahan and Mike Chou contributed enthusiasm and ideas at NYIT's Computer Graphics Lab; Ned Greene provided the cheeseburgers. Thanks also to all the good folks at PDI.

REFERENCES

Recommended reading: for a good introduction to texture mapping see [Bli76]. [Smi83] gives a helpful theoretical/intuitive introduction to digital image filtering, and [Fei80] goes into texture filtering in detail. The best references on bump mapping and illumination mapping are [Bli78a] and [Mil84], respectively. Systems aspects of texture mapping are discussed in [Coo84b] and [Per85a].

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

[Ath78] Peter R. Atherton, Kevin Weiler, Donald P. Greenberg, "Polygon Shadow Generation", *Computer Graphics*, (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 275-281.

[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, pp. 127-141, 1980.

[Bre65] J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter", *IBM Systems Journal*, vol. 4, no. 1, July 1965, pp. 25-30.

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

[Cat74] Edwin E. 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.

[Coh84] Ephraim Cohen, "Raster Image Rotation and Anti-Aliased Line Drawing", unpublished, NYIT Computer Graphics Lab, 1984.

[Coo84a] Robert L. Cook, Thomas Porter, Loren Carpenter, "Distributed Ray Tracing", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 137-145.

[Coo84b] 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.

[Cro82] Franklin C. Crow, "Computational Issues in Rendering Anti-Aliased Detail", *Proc. COMPCON '82*, Spring 1982, pp. 238-244.

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

[Dip85] Mark A. Z. Dippe, Erling Henry Wold, "Antialiasing Through Stochastic Sampling", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 69-78.

[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.

[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 which appeared in *Computer Graphics*, Vol. 16, No. 1, May 1982).

[Gan84] Michel Gangnet, Djamchid Ghazanfarpour, "Techniques for Perspective Mapping of Planar Textures", *Colloque Image de Biarritz, CESTA*, May 1984, pp. 29-35 (in French).

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

[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, "Applications of World Projections", *Graphics Interface '86*, May 1986.

[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*, to appear, 1986.

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

[Hec84] Paul S. Heckbert, Pat Hanrahan, "Beam Tracing Polygonal Objects", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 119-127.

[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.

[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.

[Lan80] Jeffrey M. Lane, Loren C. Carpenter, Turner Whitted, James F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces", *CACM*, vol. 23, no. 1, Jan. 1980, pp. 23-34.

[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.

[Max86] Nelson Max, personal communication, 1986.

[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.

[Nis83] Tomoyuki Nishita, Eihachiro Nakamae, "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources", *COMPSAC '83, Proc. IEEE 7th Intl. Comp. Soft. and Applications Conf.*, Nov. 1983, pp. 237-242.

[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.

[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*, Leesberg, VA, July 1982, Springer, Berlin, 1984.

[Smi79] Alvy Ray Smith, *TEXAS (Preliminary Report)*, Technical Memo 10, NYIT Computer Graphics Lab, July 1979.

[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.