VISUAL REALISM

This article describes a family of mapping techniques that have become firmly established in mainstream computer graphics. Their motivation is to increase the visual interest of rendered objects and their popularity is no doubt due to their flexibility, ease of implementation, low computing requirements, and the inherent difficulty of global illumination methods.

Visual realism is a term that needs careful qualification. For this treatment we define it as add-on techniques that modulate the effect of using a simple local reflection model, such as the Phong model (1). The Phong reflection model is an empirical model that simulates the visual effect of light reflecting from the surface of an object. Adding to it such effects as textures, shadows, or environmental reflections does nothing to make the objects more "real" in the sense that we are attending to more accurate calculations concerning the behavior of light at the surface of an object—just that we are ameliorating the plasticlike effect of using Phong on its own.

The techniques that we describe are approximate, but they are visually effective. For example, in shadow mapping we can calculate only the geometry of the shadow—we cannot find out what the reflected light intensity should be inside an area of a scene that is in shadow. Such calculations are the domain of global illumination methods which attempt to calculate, rather than simulate, light–object interaction and which are often described as methods that "pursue photorealism." Thus photorealism has come to mean modeling light–object interaction with an accuracy that approaches that of a photograph of a scene, whereas visual realism in the context of this treatment is defined as a set of techniques that use efficient "tricks" to make a surface more realistic without going to the inordinate expense of trying to calculate light-object interaction accurately. (And it is the case anyway that global illumination methods are still very much a research area and do not themselves approach complete photorealism.)

First consider texture techniques. As used in computer graphics, "texture" is a somewhat confusing term and generally does not mean controlling the small-scale geometry of the surface of a computer graphic object—the "normal" meaning of the word. Instead the color of a Phong-shaded object is modulated by controlling the three diffuse coefficients in the Phong reflection model. (Color variations in the physical world are not, of course, generally regarded as texture.) Thus as the rendering proceeds at pixel-by-pixel level, we pick up values for the Phong diffuse reflection coefficients, and the diffuse component (the color) of the shading changes as a function of the texture map(s).

This simple pixel-level operation conceals many difficulties, and the geometry of texture mapping is not straightforward. As usual we make simplifications that lead to a visually acceptable solution. There are three origins to the difficulties:

1. We want mostly to use texture mapping with the most popular representation in computer graphics—the polygon mesh representation. This is a geometric representation where the object surface is approximated, and this approximation is defined only at the vertices. In a sense we have no surface, only an approximation to one. So how can we physically derive a texture value at a surface point if the surface does not exist?

2. In the main, we want to use two-dimensional (2-D) texture maps because we have an almost endless source of textures that we can derive from frame grabbing the real world by using 2-D paint software or by generating textures procedurally. Thus the mainstream demand is to map a 2-D texture onto a surface approximated by a polygon mesh.
3. Aliasing problems in texture mapping are usually highly visible. By definition, textures usually manifest some kind of coherence or periodicity. Aliasing breaks this up, and the resulting mess is usually highly visible. This effect occurs as the periodicity in the texture approaches the pixel resolution.

Now consider shadows. Shadows are important in scenes. A scene without shadows looks artificial. They give clues concerning the scene, consolidate spatial relationships between objects, and give information on the position of the light source. To compute shadows completely, we need knowledge of their shapes and the light intensity inside them. An area of the scene in shadow is not completely bereft of light. It simply cannot see direct illumination but receives indirect illumination from another nearby object. Add-on shadow algorithms are all "geometric." By this we mean that they calculate the position and the shape of the shadows. They cannot compute what the light intensity should be inside the shadow area, and this is set arbitrarily. Thus we have the curious procedure of a scene, shaded by using the Phong reflection model, having shadows "pasted in," where the geometry of the shadows is accurately calculated but the light intensity is merely guessed at.

At first it may seem somewhat curious to group texture mapping and shadows into the same topic area, but shadows like texture mapping are commonly handled by using an empirical add-on algorithm. Shadows are pasted into the scene like texture maps. The other parallel with texture maps is that the easiest algorithm to use computes a map for each light source in the scene, known as a shadow map. The map is accessed during rendering, just as a texture map is referenced to find out if a pixel is in shadow or not. Like the Z-buffer algorithm in hidden surface removal, this algorithm is easy to implement and has become a pseudostandard. Also like the Z-buffer algorithm, it trades simplicity against high memory cost.

TEXTURE MAPPING—WHICH ASPECTS OF THE OBJECT TO MODULATE

Now we list the possible ways in which certain properties of a computer graphic model can be modulated with variations under control of a texture map. We have listed these in approximate order of their popularity (which also relates to their ease of use or implementation):

1. Color: As we have already pointed out, this is by far the most common object property that is controlled by a texture map. We simply modulate the diffuse reflection coefficients in the Phong reflection model with the corresponding color from the texture map (2). (We could also change the specular coefficients across the surface of an object so that it appears shiny and matte as a function of the texture map, but this is less common.)
2. Specular "color": This technique, known as environment mapping (3), reflectance mapping, or chrome mapping, is a special case of ray tracing (4) where we use texture map techniques to avoid the expense of ray tracing. The map is designed so that it looks as if the (specular) object is reflecting the environment or background in which it is placed.
3. Normal vector perturbation: This elegant technique applies a perturbation to the surface normal according to the corresponding value in the map. The technique is known as bump mapping and was developed by J. Blinn, a famous pioneer of three-dimensional (3-D) computer graphic techniques (5). The device works because the intensity returned by a Phong shading equation reduces to a function of the surface normal at the point currently being shaded if the appropriate simplifications are made. If the surface normal is perturbed, then the shading changes, and the surface that is rendered looks as if it were textured. Therefore we can use a global or general definition for the texture of a surface that is represented in the database as a polygonal mesh structure.
4. Displacement mapping (6): This mapping method, related to the previous technique, uses a height field to perturb a surface point along the direction of its surface normal. It is not a convenient technique to implement because the map must perturb the geometry of the model rather than modulate parameters in the shading equation.
5. Transparency: A map is used to control the opacity of a transparent object (7). A good example is etched glass whose shiny surface is roughened (to cause opacity) with some decorative pattern.

TWO-DIMENSIONAL TEXTURES AND THREE-DIMENSIONAL OBJECTS

The process of mapping a 2-D texture onto an object and rendering can be viewed (and implemented) as a forward- or inverse-mapping process. First consider forward mapping (Fig. 1). The overall mapping can be described by two transforma-

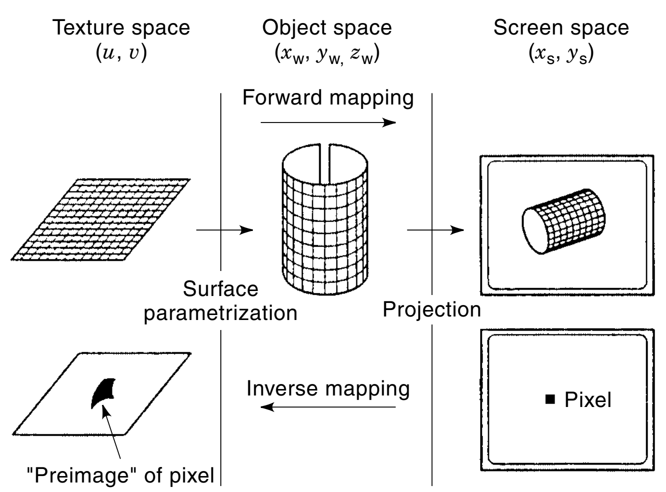

Figure 1. Two ways of viewing the process of 2-D texture mapping.

tions as shown or as a single combined transformation. The first transformation, sometimes known as surface parameterization, takes the 2-D texture pattern and "glues" it on the object. The second transformation is the standard object-to-screen space mapping. Two major difficulties arise in texture mapping: inventing a suitable surface parameterization and antialiasing. The difficulty with the first transformation is caused by the fact that we normally wish to stick a texture pattern on a polygonal mesh object, itself a discontinuous approximation to a real object. Surface parameterizations are not defined for such objects. They have to be invented. This contrasts with quadric and cubic surfaces where parameterizations are readily available. If we use the analogy of wallpaper pasting, How are we going to paste the wallpaper onto the polygonal mesh object? This is a problem to which there is no good solution, but a variety of ad hoc techniques have evolved. In the end, forward mapping is useful only if we have a surface parameterization which means that we virtually treat the texture information as part of the object properties 'collecting' the texture information when we access the geometric information associated with the object.

Most renderers that incorporate texture mapping use algorithms driven from screen space one pixel at a time. Interpolative shading and Z-buffer hidden surface removal imply a pixel-by-pixel ordering for each polygon. This means that we must find a single texture value for each pixel to insert into the interpolative shading scheme. The easiest way to do this is by inverse mapping. We find the "preimage" of the current pixel in the texture domain. Figure 1 shows the general idea of inverse mapping. Because the overall transform is nonlinear, the pixel maps into an area in texture space that generally is a curvilinear quadrilateral. To perform the inverse transformation, we need to take the four pixel corner points, invert the object-to-screen space transformation, and invert the surface parameterization. Another reason for adopting this methodology is that it facilitates antialiasing.

The use of an antialiasing method is mandatory with texture mapping. This is easily seen by considering an object retreating away from a viewer, so that its projection in screen space covers fewer and fewer pixels. As the object size decreases, the preimage of a pixel in texture space increases, covering a larger area. If we simply point sample at the center of the pixel and take the value of $T(u, v)$ at the corresponding point in texture space, then grossly incorrect results follow [Fig. 2(a), (b) and (c)]. These problems are highly visible and move when animated. Consider Fig. 2(b) and (c). Say, for example, that an object projects onto a single pixel and moves so that the preimage translates across the $T(u, v)$. As the object moves, it would switch color from black to white.

In this context then antialiasing means integrating the information over the pixel preimage and using this value in the shading calculation for the current pixel [Fig. 2(d)]. At best we can only approximate this integral because we have no knowledge of the shape of the quadrilateral, only its four corner points.

Polygonal Mesh Texture Mapping: Two-Part Mapping

Two-part texture mapping is a much used technique that overcomes the surface parameterization problem in polygonal mesh objects by using an 'easy' intermediate surface onto which the texture is initially projected. Introduced by Bier and Sloan (8), it is a method that maps 2-D texture onto unconstrained polygonal mesh models. The method is also used to implement environment mapping and is thus a method that unifies texture mapping and environment mapping.

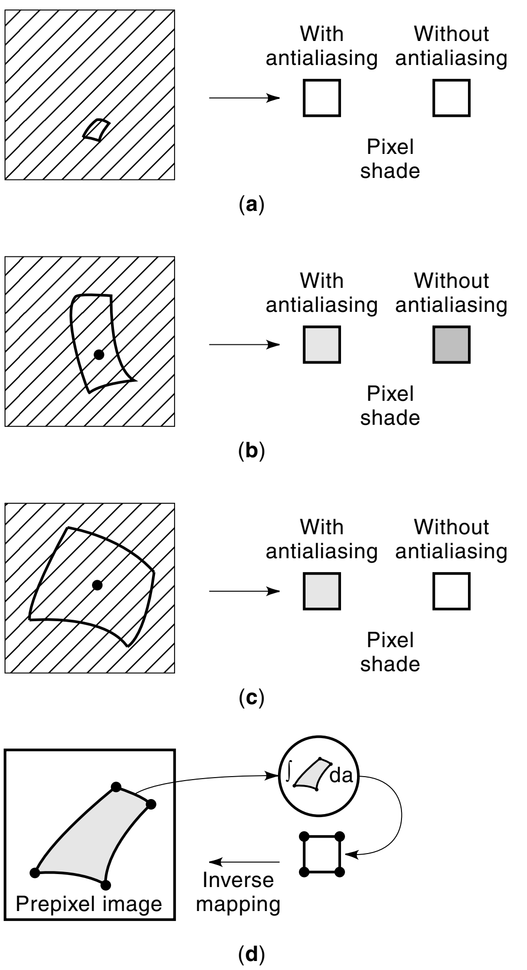

Figure 2. Pixels and preimages in $T(u, v)$ space.

The process is known as two-part mapping because the texture is mapped onto an intermediate surface before being mapped onto the object. The intermediate surface is generally nonplanar, but it possesses an analytic mapping function, and the 2-D texture is mapped onto this surface without difficulty. Then finding the correspondence between the object point and the texture point becomes a 3-D-to-3-D mapping.

The basis of the method is most easily described as a two-stage, forward-mapping process (Fig. 3):

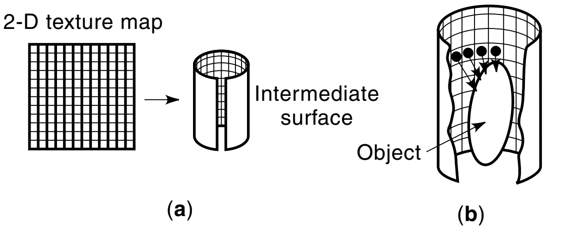

Figure 3. Two-stage mapping as a forward process: (a) S mapping; (b) O mapping.

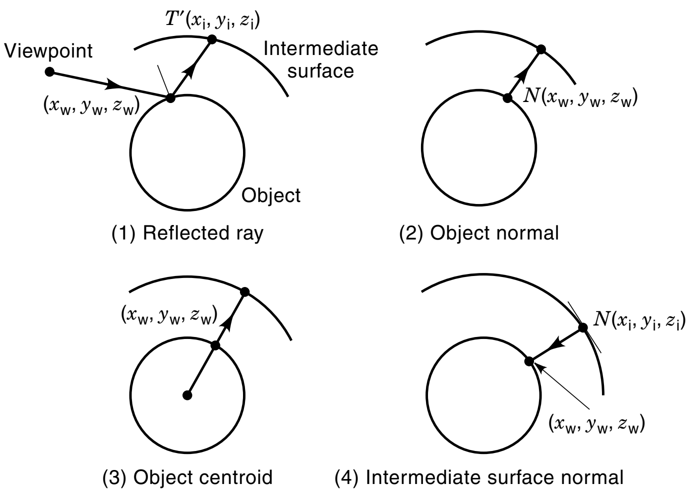

Figure 4. The four possible O mappings that map the intermediate surface texture T' onto the object.

1. The first stage is mapping from 2-D texture space to a simple 3-D intermediate surface, such as a cylinder:

$$T(u, v) \rightarrow T'(x_i, y_i, z_i)$$

   This is known as S mapping.
2. A second stage maps the 3-D texture pattern onto the object surface:

$$T'(x_i, y_i, z_i) \rightarrow O(x_w, y_w, z_w)$$

   This is referred to as O mapping.

These combined operations distort the texture pattern onto the object in a "natural" way, for example, one variation of the method is a "shrink-wrap" mapping, where the planar texture pattern shrinks onto the object in the manner suggested by the eponym.

For S mapping, Bier describes four intermediate surfaces: a plane at any orientation, the curved surface of a cylinder, the faces of a cube, and the surface of a sphere. Although it makes no difference mathematically, it is useful to consider that $T(u, v)$ is mapped onto the interior surfaces of these objects. For example, consider the cylinder. Given a parametric definition of the curved surface of a cylinder as a set of points (θ, h), we transform the point (u, v) onto the cylinder as follows:

$$(\theta, h) \rightarrow (u, v) = [(r/c)(\theta - \theta_0), (1/d)(h - h_0)]$$

where c and d are scaling factors and θ_0 and h_0 position the texture on the cylinder of radius r.

Various possibilities occur for O mapping where the texture values for $O(x_w, y_w, z_w)$ are obtained from $T'(x_i, y_i, z_i)$, and these are best considered from a ray-tracing viewpoint. Following are the four O mappings as shown in Fig. 4:

1. The intersection of the reflected view ray with the intermediate surface, T'. [This is in fact identical to environment mapping (see later section). The only difference between the general process of using this O mapping and environment mapping is that the texture pattern mapped onto the intermediate surface is a surrounding environment like a room interior.]
2. The intersection of the surface normal at (x_w, y_w, z_w) with T'.
3. The intersection of a line through (x_w, y_w, z_w) and the object centroid with T'.
4. The intersection of the line from (x_w, y_w, z_w) to T' whose orientation is given by the surface normal at (x_i, y_i, z_i). If the intermediate surface is simply a plane, then this is equivalent to considering that the texture map is a slide in a slide projector. A bundle of parallel light rays from the slide projector impinges on the object surface. Alternatively it is also equivalent to 3-D texture mapping (see later section) where the field is defined by 'extruding' the 2-D texture map along an axis normal to the plane of the pattern.

Now let us consider this procedure as an inverse-mapping process for the shrink-wrap case. We break the process into three stages (Fig. 5):

1. Inverse map four pixel points to four points (x_w, y_w, z_w) on the surface of the object.

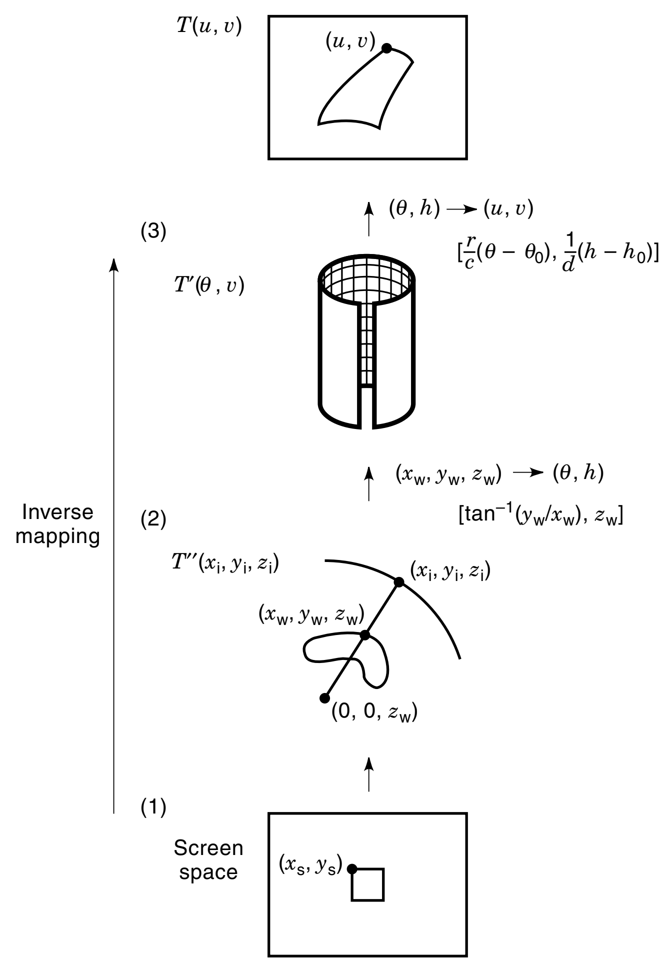

Figure 5. Inverse mapping using the shrink-wrap method.

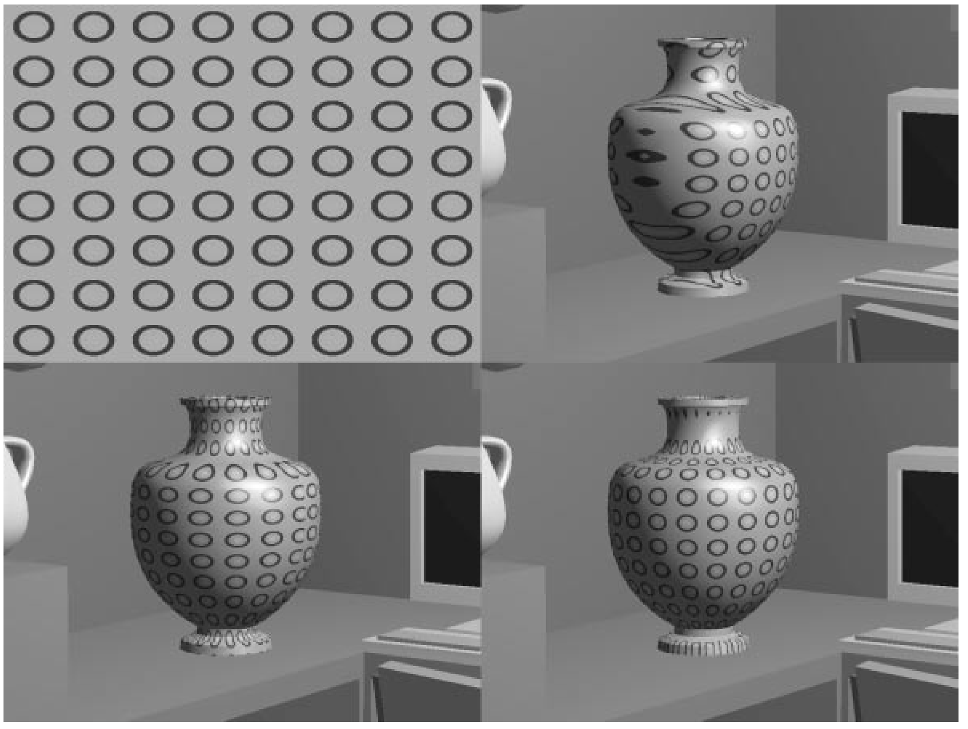

Figure 6. Examples of two-part texture mapping. In clockwise order, starting from the texture map, the intermediate surfaces are a plane, a sphere, and a cylinder.

2. Apply the O mapping to find the point (θ, h) on the surface of the cylinder. In the shrink-wrap case we simply join the object point to the center of the cylinder, and the intersection of this line with the surface of the cylinder gives us (x_i, y_i, z_i):

$$x_w, y_w, z_w \rightarrow (\theta, h) = [\tan^{-1}(y_w/z_w), z_w]$$

3. Apply the S mapping to find the point (u, v) corresponding to (θ, h).

Figure 6 shows examples of mapping the same texture onto an object using different intermediate surfaces. The intermediate objects are a plane (equivalently no object, the texture map is a plane), a cylinder, and a sphere. There are two points that can be made from these illustrations. First an intermediate mapping can be chosen appropriate to the shape of the object. A solid of revolution may be best suited, for example, to a cylinder. Second, although the method does not place any constraints on the shape of the object, the final visual effect may be deemed unsatisfactory. Usually what we mean by texture does not involve subjecting the texture pattern to large geometric distortions. It is because of this that many practical methods are interactive and involve some strategy like predistorting the texture map in 2-D space until it produces a good result when it is struck onto the object.

Two-Dimensional Texture Domain Techniques: Mapping onto Bicubic Parametric Patches

If an object is a quadric or a cubic, then surface parameterization is straightforward. In the previous section we used quadrics as intermediate surfaces exactly for this reason. If the object is a bicubic parametric patch, texture mapping is trivial because a parametric patch, by definition, already possesses (u, v) values everywhere on its surface.

The first use of texture in computer graphics was a method developed by Catmull (2). This technique was applied to bicubic parametric patch models. The algorithm subdivides a surface patch in object space and at the same time executes a corresponding subdivision in texture space. The idea is that the patch subdivision proceeds until it covers a single pixel. When the patch subdivision process terminates, the required texture value(s) for the pixel is obtained from the area enclosed by the current level of subdivision in the texture domain. This is a straightforward technique that is easily implemented as an extension to a bicubic patch renderer. A variation of this method was used by Cook (9) where object surfaces are subdivided into “micropolygons” and are flat shaded with values from a corresponding subdivision in texture space.

THREE-DIMENSIONAL TEXTURE DOMAIN TECHNIQUES

A method that neatly circumvents the 2-D-to-3-D mapping problem is to employ a 3-D texture domain (10,11). We can image that a texture value exists everywhere in the object domain or definition. Ignoring object scale problems (the texture ‘size’ does not vary as the size of the object changes), then we can say that, given a point (x_w, y_w, z_w) on the surface of an object, its texture is given by the identity mapping $T(x_w, y_w, z_w)$. This is like the process of sculpting or carving an object out of a solid block of material. The color of the object is determined by the intersection of its surface with the predefined 3-D texture field.

A fairly obvious requirement of this technique is that the 3-D texture field is obtained by procedural generation. Storing a complete 3-D field would be prohibitively expensive in memory requirements. Thus the coordinates (x_w, y_w, z_w) are used to index a procedure that defines the 3-D texture field for that point.

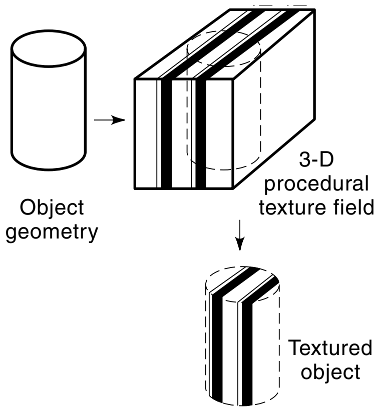

Figure 7. 3-D texture mapping in object space.

A significant advantage of eliminating the mapping problem is that objects of arbitrary complexity can receive a texture on their surface in a 'coherent' fashion. No discontinuities occur when the texture appears on the object.

Figure 7 shows the overall idea of the technique which is used mostly in conjunction with a 3-D noise function to generate the definition. This approach is well established now in 3-D computer graphics because it works well visually. It is particularly successful at simulating such phenomena as turbulence and has been used to model, for example, objects of marble. A 3-D noise function is built by assigning random integers to a 3-D array. Then this 3-D block of random numbers is accessed by a 3-D real number, and interpolation among the nearest integers returns a 3-D real noise value. This is used to perturb the color associated with the point on the surface of the object by using the point to access the noise function. Consider simulating a dark seam in a marble object. We could set up a block of marble as a "sandwich" of light and dark material. Then we have two fields accessed by a surface point: the light dark definition which determines the initial color of the point and then the noise function which perturbs this color. Figure 8 is an example of an object that has been textured by using this process.

The big problem with 3-D texture mapping is that it is difficult to create procedural definitions and because of this the method lacks the flexibility and generality of 2-D texture mapping.

BUMP MAPPING

Bump mapping, a technique developed by Blinn in 1978 (5), is an elegant device that enables a surface to appear as if it were wrinkled or dimpled without the need to model these depressions geometrically. Instead, the surface normal is angularly perturbed according to information given in a 2-D bump map and this "tricks" a local reflection model, wherein intensity is a function mainly of the surface normal, into producing (apparent) local geometric variations on a smooth surface. The only problem with bump mapping is that because the pits or depressions do not exist in the model, a silhouette edge that appears to pass through a depression does not produce the expected cross section. In other words the silhouette edge follows the original geometry of the model.

It is an important technique because it appears to texture a surface in the normal sense of the word rather than modulating the color of a flat surface. Figure 9 shows an example of this technique.

Texturing the surface in the rendering phase without perturbing the geometry, bypasses serious modeling problems that would otherwise occur. If the object is polygonal, the mesh would have to be fine enough to receive the perturbations from the texture map, a serious imposition on the original modeling phase, particularly if the texture is to be an option.

Figure 8. 3-D texturing using a perturbed "sandwich" of light and dark material to give a marble effect.

Figure 9. An example of bump mapping.

In bump mapping we need to perturb the normal vector at a point on the surface so that when a local reflection model is applied and the surface is shaded, it looks as if the surface geometry has been perturbed by the bump map which is a 2-D height field. Refer to Fig. 10 which shows an overview of the process.

For simplicity, if we assume that $O(u, v)$ is a parameterized function representing the position vectors of points O on the surface of an object, then the normal to the surface at a point is given by

$$N = O_{\mathrm{u}} \times O_{\mathrm{v}}$$

where O_{u} and O_{v} are the partial derivatives of the surface at point O in the tangent plane.

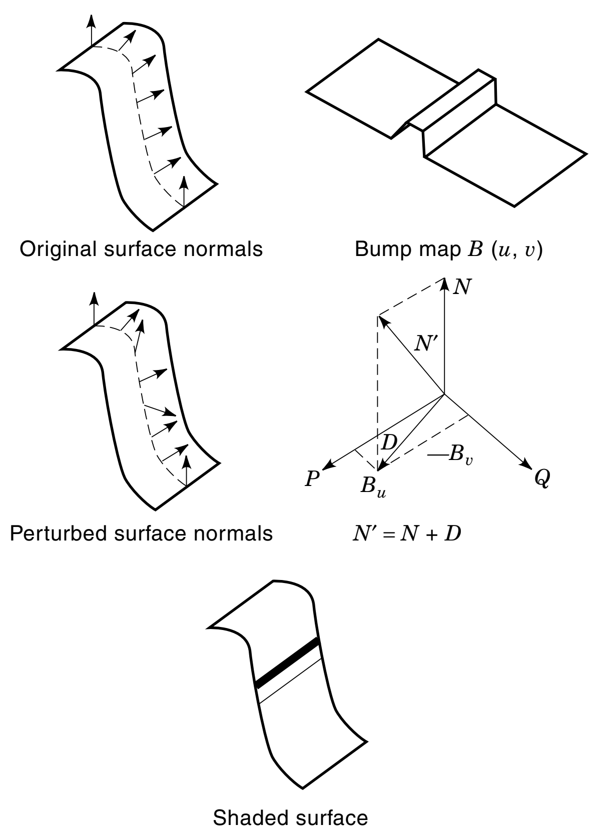

Figure 10. Bump-mapping geometry.

We define two other vectors that lie in the tangent plane:

$$P = N \times O_{\mathrm{v}}$$

and

$$Q = N \times O_{\mathrm{u}}$$

D is a vector added to N to perturb its direction to N':

$$N' = N + D$$

The vectors P, Q, and N form a coordinate system. D is derived from P, Q, and B, a bump map. The bump map is a height field and the idea is that D should transfer the height variations in the bump map into orientation perturbations in N, so that when the surface is shaded, the variations in N produce the effect specified in the bump map. In other words the height variations in the bump map are transformed into orientation perturbations in the surface normal which makes the surface look as if it has been displaced by the height variations in the bump map. It can be shown that D is given by

$$D = B_{\mathrm{u}} P - B_{\mathrm{v}} Q$$

where B_{u} and B_{v} are the partial derivatives of the bump map $B(u, v)$. Thus we define a bump map as a displacement function or height field but use its derivatives at the point (u, v) to calculate D.

ENVIRONMENT MAPPING

Environment mapping (3,12) is the process of reflecting a surrounding environment in a shiny object. Environment mapping was originally introduced as an inexpensive alternative to ray tracing. The idea is that a shiny object reflects its surroundings or environment and if this is prestored or rendered as a map, the texture mapping can be used when the object is rendered to give this effect. Thus the reflections are achieved by texture mapping rather than the expensive alternative of ray tracing. It is distinguished from "normal" texture mapping in that the pattern seen on an object is a function of the view vector V. A particular detail in the environment moves across the object as V changes. The idea,

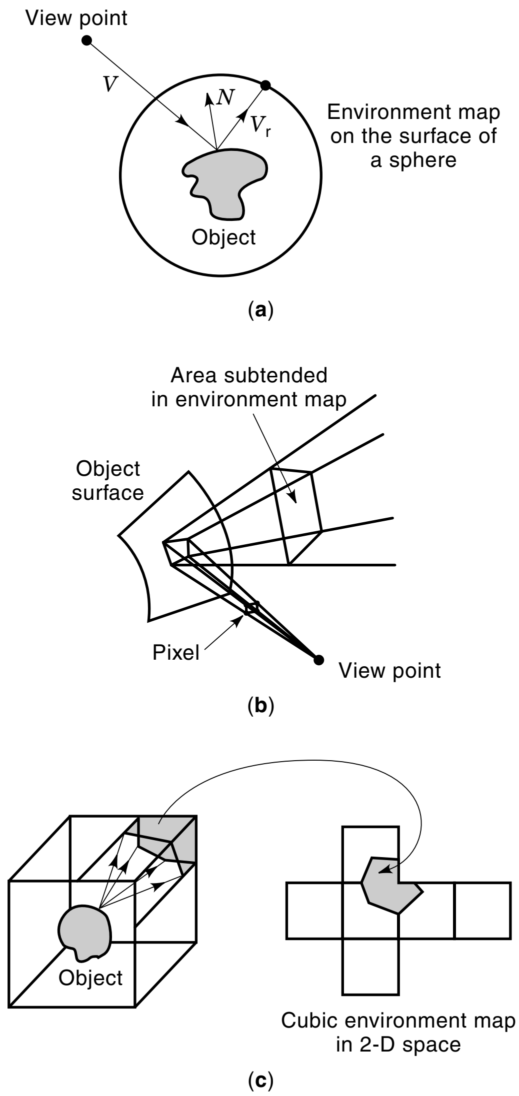

Figure 11. Environment mapping—principle and practice. (a) Environment mapping in principle. (b) Inverse mapping produces a reflection beam. (c) Cubic maps are used in practice.

depicted in principle in Fig. 11, shows a cross section of a spherical map surrounding an object. (Note that this is a reproduction of part of Fig. 4 which deals with two-part texture mapping. Environment mapping is a special case of two-part texture mapping.) Reflecting a view ray V from the surface of an object produces an index into the map which is then used as a normal texture map.

Originally introduced in the 1980s, it quickly became a popular technique. The most popular manifestation of environment mapping uses a box or cube as an intermediate surface. The maps are constructed by taking six photographs of (say) a room interior or by rendering the map with a computer graphics renderer using six mutually perpendicular viewing directions. Cubic environment maps are easier to construct than spherical maps which also suffer from distortion at the poles.

Photographic environment maps offer the potential to be used in productions in which a computer graphics object can be matted into a real environment. The object, usually animated, has the real environment reflected in its surface as it is rendered and moves about the room. The resulting effect makes the rendered object look as if it were part of the environment from which the map has been constructed. This device has been much used in TV commercials where an object, usually the advertised product, is animated in a photographed real environment.

Recently, environment mapping has found a new lease on life as an image-based rendering technique. Here, a person, the virtual viewer, replaces the object, and that part of the map intercepted by the viewer's field of vision is presented to the viewer as a 2-D projection.

Consider Fig. 11 again. In practice we have to consider four rays through the pixel point that define a reflection 'cone' with a quadrilateral cross section. Then the region that subtends the environment map is filtered to give a single shading attribute for the pixel. In other words, the technique is identical to normal inverse-mapping texture mapping except that the area intercepted by a pixel may spread over one, two or three maps. Environment mapping is, geometrically, an approximate technique and an object that is environment mapped does not exhibit the same reflected images as a ray-traced object placed in the same environment. The geometric errors are a function of the size of the object in the environment. An example of environment mapping is shown in Fig. 12.

INTERACTIVE TECHNIQUES IN TEXTURE MAPPING

One of the main problems in designing a conventional 2-D texture map is visualizing the result on the rendered object. Say an artist or a designer is creating a texture map by painting directly in the 2-D u, v space of the map. We know that the distortion of the map, when it is "stuck" on the object, is a function of the shape of the object and the mapping method used. To design a texture interactively, the artist needs to see the final rendered object and have some intuition of the mapping mechanism to predict the effect of changes made to the texture map.

Now we describe two interactive techniques. In the first the designer paints in u, v or texture space. The second attempts to make designers think that they are painting directly on the object in 3-D world space.

The first technique (4) is extremely simple and evolved to texture animals/objects that exhibit a plane of symmetry. It is simply an interactive version of two-part texture mapping with a plane as the intermediate object. The overall idea is shown in Fig. 13. The animal model is enclosed in a bounding box. Then the texture map $T(u, v)$ is "stuck" on the two faces of the box by using the 'minimax' coordinates of the box, and points in $T(u, v)$ are projected onto the object by using a parallel projection with projectors normal to the plane of symmetry.

The second technique (13) is to allow the artist to interact directly with the rendered version on the screen. The artist applies the texture by using an interactive device simulating a brush, and the effect on the screen is as if the painter were applying paint directly to the 3-D object. It is easy to see the advantages of such a method by looking first at how it differs from a normal 2-D paint program which basically enables a user to color selected pixels on the screen.

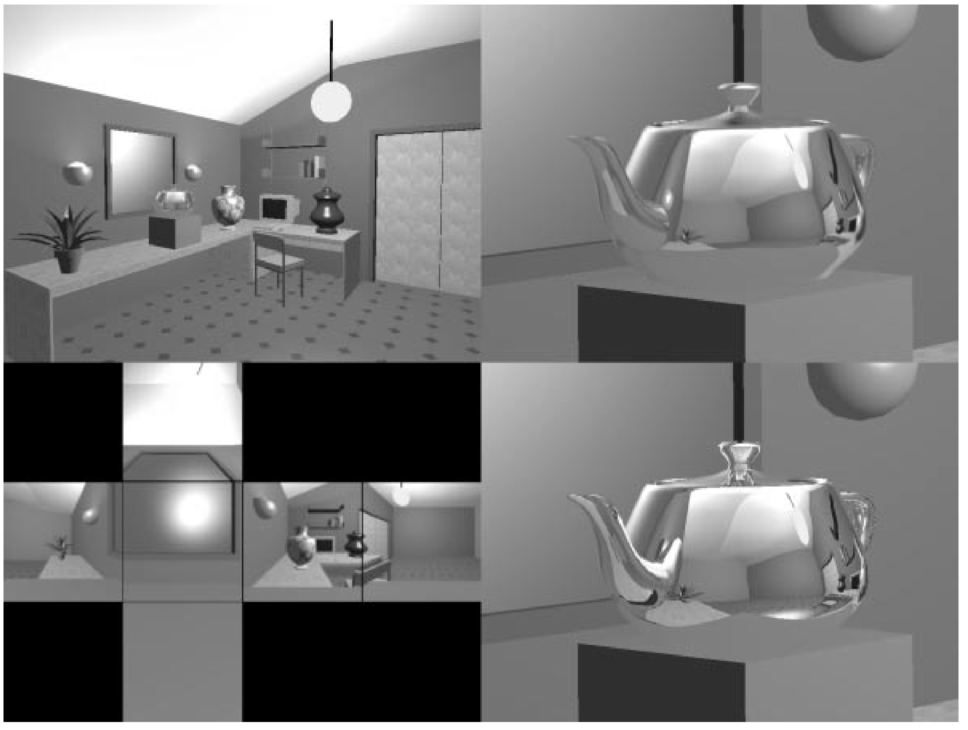

Figure 12. Environment mapping: each of the environment maps (individual faces of the flattened cube) has a resolution of 128 × 128 pixels. The top right image is a close-up of the environment-mapped teapot. In contrast with the ray traced teapot below it, the technique produces geometrically incorrect reflections of the environment, and no self-reflections occur.

Say we have a sphere (circle in screen space). With a normal paint program, if we selected, say, the color green and painted the sphere, then unless we explicitly altered the color, the sphere's projection would be filled with the selected uniform green color. However, the idea of using a paint interaction in object space is that as the green paint is applied, its color changes according to the application of the Phong shading equation. If the paint is shiny, a specular highlight appears. Extending the idea to texture mapping means that the artist can paint the texture on the object directly and the program, reversing the normal texture mapping procedure, can derive the texture map from the object. Once the process is complete, new views of the object are rendered and texture mapped in the normal way.

This approach requires a technique that identifies the corresponding point on the object surface from the screen pixel being pointed to. In the method described by Hanrahan and Haeberli (13), an auxiliary frame buffer, known as an item buffer, is used. Accessing this buffer with the coordinates of the screen cursor gives a pointer to the position on the object surface and the corresponding (u, v) coordinate values for the texture map. Clearly, we need an object representation where the surface is everywhere parameterized, and Hanrahan and Haeberli divide the object surface into a large number of micropolygons. The overall idea is illustrated in Fig. 14.

ADDING SHADOWS IN RENDERING

As we mentioned in the introduction, shadows are properly part of the global illumination problem and in 'geometric' shadow algorithms, we simply calculate the shape of a shadow. We have no way of knowing what the light intensity inside a shadow should be. This restriction has long been tolerated in mainstream rendering. Presumably, the rationale is that it is better to have a shadow pasted into the scene, as if it were a texture map, rather than having no shadow at all. Thus in the following section we deal with this aspect of rendering and leave the more considered discussion of shadows as part of a discussion of the global illumination problem. It is important to bear in mind that shadow algorithms of this type consider the geometry of the shadow, whereas in (most) global illumination approaches the shadow areas are not considered a phenomenon separate from the normal distribution of light in an environment. They are simply part of the simulation and emerge from the algorithm as an area exhibiting reflected light no different from any other area.

Shadow Mapping

Possibly the simplest approach to shadow computation, one that is easily integrated into a Z-buffer-based renderer, is the shadow Z-buffer developed by Williams in 1978 (14). This technique requires a separate shadow Z-buffer for each light source.

The algorithm is a two-step process. A scene is "rendered," and depth information is stored in the shadow Z-buffer using the light source as a viewpoint. No intensities are calculated. This computes a "depth image" from the light source of the polygons visible to the light source.

The second step is to render the scene using a normal Z-buffer algorithm. This process is enhanced as follows: if a point is visible, a coordinate transformation is used to map (x, y, z), the coordinates of the point in 3-D screen space (from the viewpoint) to (x_1, y_1, z_1), the coordinates of the point in screen space from the light point as a coordinate origin. The coordinates (x_1, y_1) are used to index the shadow Z-buffer and the corresponding depth value is compared with z_1. If z_1 is greater than the value stored in the shadow Z-buffer for that point, then a surface is nearer to the light source than the

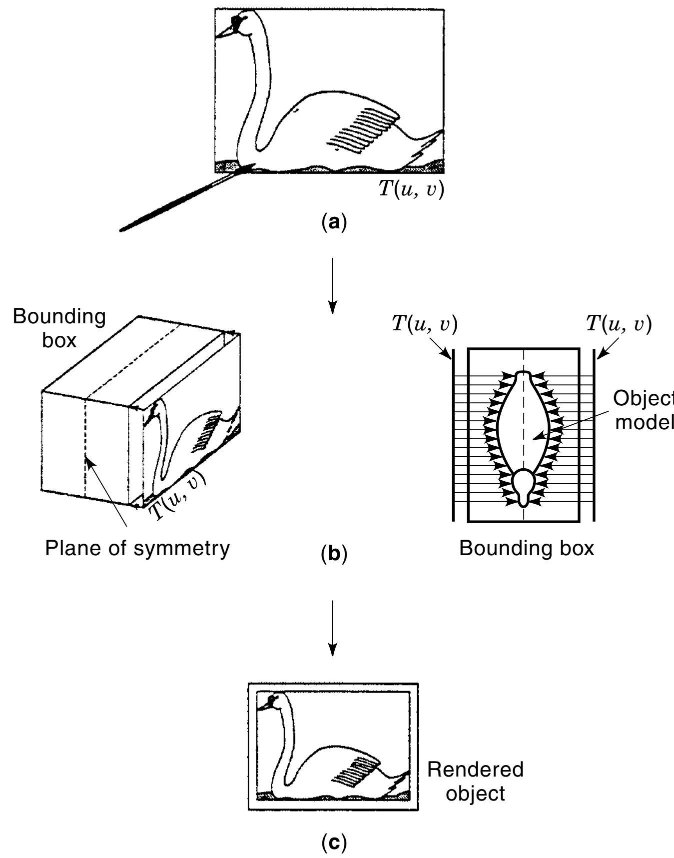

Figure 13. Interactive texture mapping—painting in $T(u, v)$ space. (a) Texture is painted using an interactive paint program. (b) Using the object's bounding box, the texture map points are projected onto the object. Each projector is parallel and normal to the bounding-box face. (c) The object is rendered, the "distortion" visualized, and the artist repeats the cycle if necessary.

point under consideration and the point is in shadow. Thus a shadow "intensity" is used, otherwise the point is rendered as normal.

An example of shadows calculated in this way and the corresponding shadow map are shown in Fig. 15. Apart from extending the high memory requirements of the Z-buffer hidden surface removal algorithm, the algorithm also extends its inefficiency. Shadow calculations are performed for surfaces that may be subsequently "overwritten," just as shading calculations are.

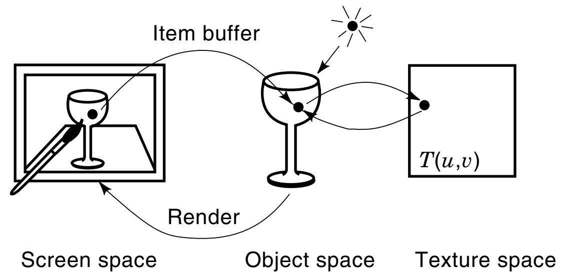

Figure 14. Iterative texture mapping—painting in object space.

MAPPING TECHNIQUES AND COMPLEXITY

Although the foregoing mapping techniques have served the computer graphics community well for two decades, recent demands for lower cost have arisen from applications for which standard rendering techniques are too expensive. The demand for interactivity in immersive virtual reality (VR) and 3-D computer games are two examples of applications where the complexity of the scene means that an alternative rendering method must be used to meet the frame generation time (which is, say, $\frac{1}{25}$ s for an interactive 3-D game).

Photographic Texture Mapping and Low-Resolution Geometry

There is nothing to stop photographs of real scenes from being used as texture maps and such a device is used in a variety of approaches all of which attempt to deal with the complexity/processing cost trade-off. A good example is to be found in the approach of Debevec et al. (15). A simple way in which this device is employed is to 'compensate' for low polygonal resolution. For example, in game applications a character can have the geometry of the head represented by a very small number of polygons if a photographic texture map is used. The detail in the facial texture compensates for the inadequate geometry. The coarseness of the polygonal resolution becomes less noticeable to the viewer.

In this sense the use of the photographic texture map subtly differs from using a photograph of an actual 2-D texture, such as, say, wood grain as used in traditional texture mapping. In this case, although the texture map is 2-D, we are using the projected 3-D information in it to imitate 3-D reality when we map it onto an object.

In this context we want to use photographic texture maps to represent or simulate fairly large-scale geometric variations rather than small-scale surface variations. It is a curious mix of two and three dimensions. We stick photographic texture maps representing geometric variations onto 3-D objects in the scene.

Another problem with complex (existing) environments is the labor involved in modeling the scene. Consider the device of using photography to assist in modeling. Currently available commercial photo-modeling software concentrates on extracting pure geometry by using a high degree of manual intervention. They use a precalibrated camera, knowledge of the position of the camera for each shot, and a sufficient number of shots to capture the structure of the building, say, that is being modeled. Extracting the edges from the shots of the building enables constructing a wire-frame model. This is usually done semiautomatically with an operator that corresponds edges in the different projections. The obvious potential advantage is that photo-modeling offers the possibility of automatically extracting the rich visual detail of the scene and the geometry. The point here is that all the detail need not be captured geometrically. It may be sufficient to represent the facade of a building by a single plane leaving the detailed geometric excursions of windows and ornamentation to be taken care of by the photo-texture.

Using photo-modeling to capture detail has some problems. One is that the information we obtain may contain light-source and view-dependent phenomena, such as shadows and specular reflections. These must be removed before the imagery is used to generate the simulated environment from any

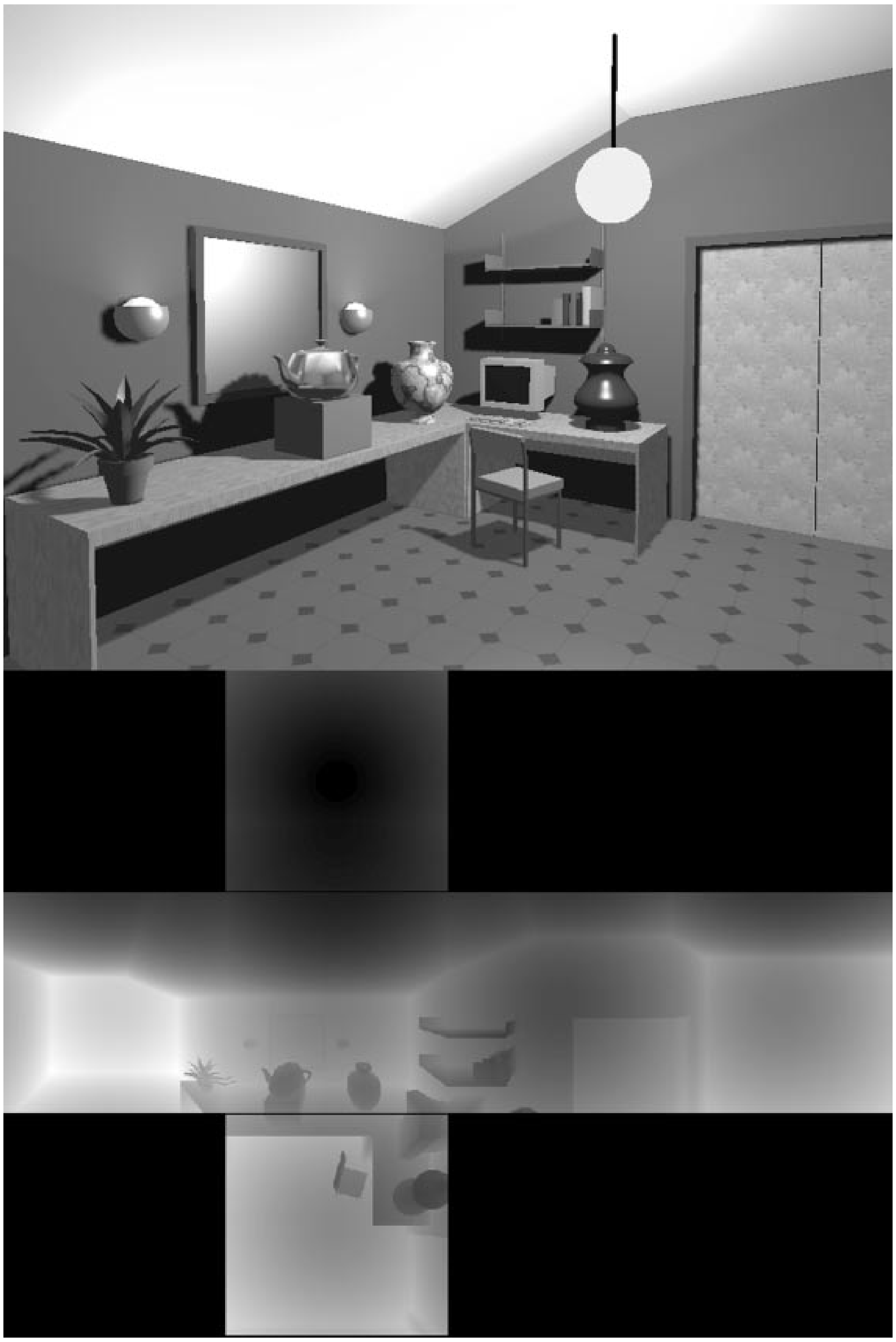

Figure 15. A scene rendered by using shadow mapping together with the shadow map for the main (spherical) light source.

viewpoint. Another problem of significance is that we may need to warp detail in a photograph to fit the geometric model. This may involve expanding a very small area of an image. Consider, for example, a photograph taken from the ground of a high building with a detailed facade. Important detail information near the top of the building may be mapped into a small area because of the projective distortion.

Photographic and Prerendered Environment Mapping

We have already mentioned a "traditional" use of photographic environment maps in animation where a computer graphic object is combined with a photographic environment map. Now consider replacing an object with a virtual viewer. For example we could position a user at a point from which a six-view (cubic) environment map has been constructed (either photographically or synthetically). If we use the approximation that the user's eyes are always positioned exactly at the environment map's viewpoint, then we can compose any projection dependent on the view direction that is demanded by users who change their direction of gaze by sampling the appropriate environment maps. Thus, for a stationary viewer positioned at the environment map viewpoint, we have achieved our goal of a view-independent solution. We have decoupled the viewing direction from the rendering pipeline.

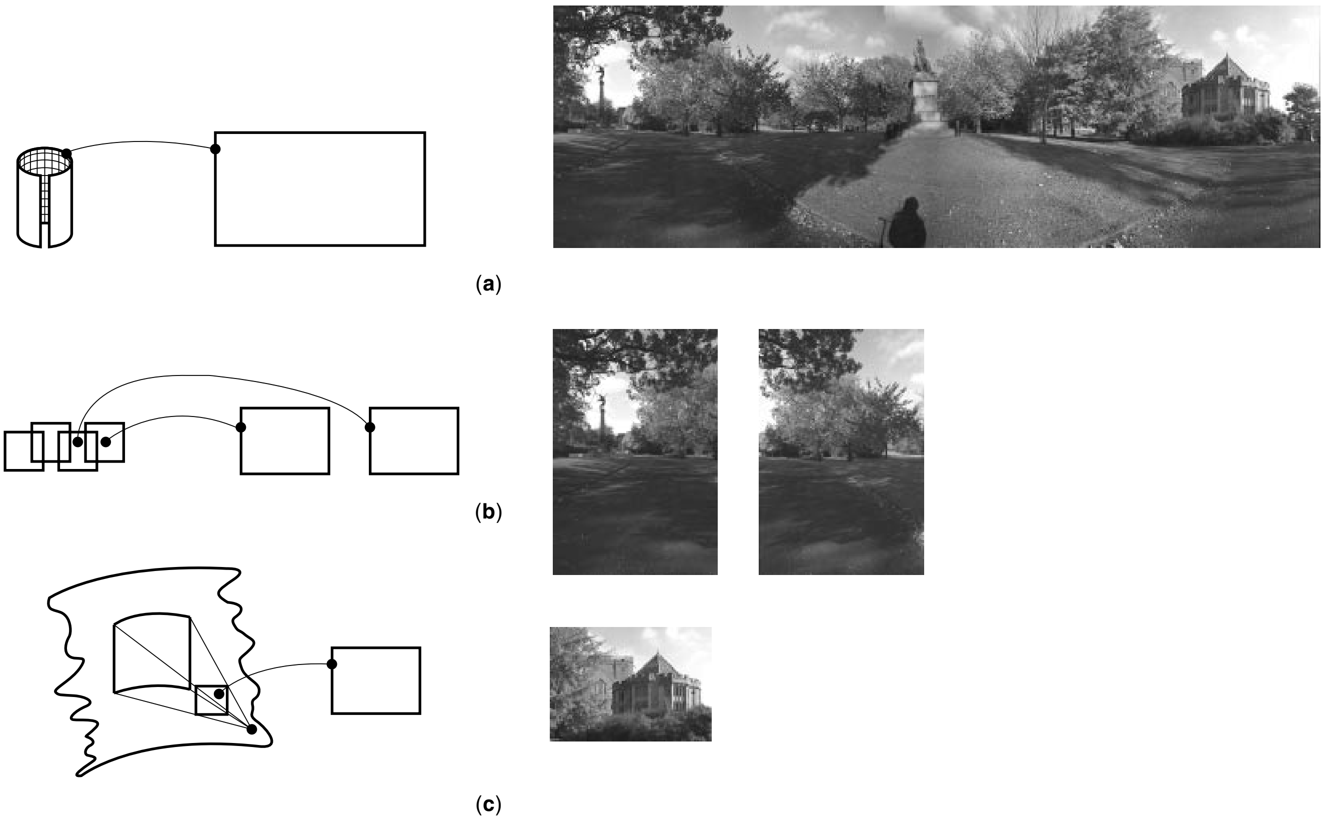

Figure 16. The QuickTime VR process for cylindrical panoramas: (a) a cylindrical environment map (a panorama) is made (b) by “stitching” normal photographs taken by moving a camera through 360°; (c) a virtual viewer positioned at the center of the cylinder looks at a section of the cylinder which is unwarped for the image plane.

Now composing a new view consists of sampling environment maps, and the scene complexity problem has been bound by the resolution of the precomputed or photographed maps. We can (to some extent) remove the constraint of the single-position viewer by having a number of environment maps rendered or photographed from different viewpoints and ‘hopping’ between them.

The best current example of this is Apple Computer’s QuickTime VR that uses a cylindrical environment map (16). QuickTime VR operates with panoramas collected from fixed viewpoints enabling the user to look around 360° and up and down to a certain extent (see Fig. 16). Walkthroughs need to be implemented by hopping, and their accuracy depends on the number of panoramas collected to represent an environment.

There is nothing to stop the environment maps from being prerendered, rather than photographic, or mixing prerendered backgrounds with computer graphics objects, as done in computer games.

FURTHER READING

Texture mapping is not covered too well outside of research publications. An exception to this is the book by Ebert et al. (17). The two-part mapping idea, together with examples of different combinations of S and O mappings, is described in the paper by Bier and Sloan (8).

The 3-D texture idea was reported simultaneously by Peachey (10) and Perlin (11). The work contains impressive illustrations that demonstrate the visual efficacy of this technique. The paper by Blinn on bump mapping (5) contains a full mathematical treatment of his elegant technique together with a discussion of some of its difficulties.

Antialiasing is mandatory in texture mapping because, by definition, texture maps normally exhibit some form of periodicity. This can “break up” disturbingly when the period approaches a pixel extent. The classic antialiasing method is mip-mapping described in a paper by Williams (18).

Hanrahan and Haeberli (13) developed the 3-D paint approach that we have described. Their paper also contains many possible extensions to this technique that we have not mentioned, such as geometry painting which is using the brush to make small scale changes in the geometry of the surface.

Full details of antialiasing with the shadow Z-buffer approach are given in the paper by Reeves (19).

Our article addresses well-established, much implemented techniques. Other research has broadened these approaches and has included fur modeling as a form of texture mapping (20–22), texture models inspired by biochemical processes (23,24), interactive texturing on implicit surfaces (25), and using textures for modeling dirty environments and simulating wear and aging (26,27).

BIBLIOGRAPHY

1. P. Bui-Tuong, Illumination for computer-generated pictures, *Comm. ACM,* **18** (6): 311–317, 1975.
2. E. Catmull, Subdivision algorithm for the display of curved surfaces, Ph.D. Thesis, Univ. Utah, Salt Lake City, 1974.
3. J. Blinn and M. Newell, Texture and reflection in computer generated images, *Comm. ACM,* **19** (10): 542–546, 1976.
4. A. H. Watt and M. Watt, *Advanced Animation and Rendering Techniques: Theory and Practice,* Reading, MA: Addison-Wesley, 1992.
5. J. F. Blinn, Simulation of wrinkled surfaces, *Proc. SIGGRAPH '78,* 1978, pp. 286–292.
6. R. Cook, Shade trees, *Proc. SIGGRAPH '84,* 1984, pp. 223–231.
7. D. S. Kay and D. Greenberg, Transparency for computer synthesized images, *Proc. SIGGRAPH '79,* 1979, pp. 158–164.
8. E. A. Bier and K. R. Sloan, Two-part texture mapping, *IEEE Comput. Graphics Appl.,* **6** (9): 40–53, 1986.
9. R. L. Cook, L. Carpenter, and E. Catmull, The Reyes image rendering architecture, *Proc. SIGGRAPH '87,* 1987, pp. 95–102.
10. D. R. Peachey, Solid texturing of complex surfaces, *Proc. SIGGRAPH '85,* 1985, pp. 279–286.
11. K. Perlin, An image synthesizer, *Proc. SIGGRAPH '85,* 1985, pp. 287–296.
12. N. Greene, Environment mapping and other applications of world projections, *IEEE Comput. Graphics Appl.,* **6** (11): 108–114, 1986.
13. P. Hanrahan and P. Haeberli, Direct WYSIWYG painting and texturing on 3-D shapes, *Proc. SIGGRAPH '90,* 1990, pp. 215–223.
14. L. Williams, Casting curved shadows on curved surfaces, *Proc. SIGGRAPH '78,* 1978, pp. 270–274.
15. P. E. Debevec, C. J. Taylor, and J. Malik, Modelling and rendering architecture from photographs: A hybrid geometry and image based approach, *Proc. SIGGRAPH '96,* 1996, pp. 11–20.
16. S. E. Chen, Quicktime VR—An image based approach to virtual environment navigation, *Proc. SIGGRAPH '95,* 1995, pp. 29–38.
17. D. S. Ebert et al., *Texturing and Modeling, A Procedural Approach,* New York: Academic Press, 1994.
18. L. Williams, Pyramidal parametrics, *Proc. SIGGRAPH '83,* 1983, pp. 1–11.
19. W. Reeves, D. Salesin, and R. Cook, Rendering antialiased shadows with depth maps, *Proc. SIGGRAPH '87,* 1987, pp. 283–291.
20. J. T. Kajiya and T. L. Kay, Rendering fur with three dimensional textures, *Proc. SIGGRAPH '89,* 1989, pp. 271–277.
21. K. Perlin and E. M. Hoffert, Hypertexture, *Proc. SIGGRAPH '89,* 1989, pp. 253–262.
22. D. B. Goldman, Fake fur rendering, *Proc. SIGGRAPH '97,* 1997, pp. 127–134.
23. G. Turk, Generating textures for arbitrary surfaces using reaction-diffusion, *Proc. SIGGRAPH '91,* 1991, pp. 289–298.
24. A. Witkin and M. Kass, Reaction-diffusion textures, *Proc. SIGGRAPH '91,* 1991, pp. 299–308.
25. H. K. Pedersen, A framework for interactive texturing on curved surfaces, *Proc. SIGGRAPH '96,* 1996, pp. 295–302.
26. W. Becket and N. I. Badler, Imperfection for realistic image synthesis, *J. Visualization Comput. Animation,* **1**: 26–32, 1990.
27. J. Dorsey and P. Hanrahan, Modeling and rendering metallic patinas, *Proc. SIGGRAPH '96,* 1996, pp. 387–396.

ALAN WATT
STEVE MADDOCK
University of Sheffield

VLIW PROCESSORS. See PARALLEL PROCESSING, SUPERSCALAR AND VLIW PROCESSORS.