Inverse Displacement Mapping

J.W. Patterson*, S.G. Hoggar‡, and J.R. Logie*

Abstract

Inverse displacement mapping is a variant of displacement mapping which does not actually perturb the geometry of the surface being mapped. It is thus a true texture mapping technique which can be applied during rendering without breaking viewing pipeline discipline. The method works by first projecting probing rays into texture space and solving for a ray-texture intersection there. Shadows can also be determined by mapping a probe from the intersection point towards the light source into texture space and seeing if an intersection results. Our implementation uses as much knowledge about the base surface as possible to speed up the ray-surface intersection calculation. We have limited our treatment to spheres, cones, cylinders and planes, and our rendering method to ray casting, in order to contain the scope of this work up to the present. The inverse displacement mapping technique can, however, be applied more widely, for example as part of a full ray-tracer, and also as part of the rendering pipeline for a wider class of smooth surfaces.

This paper was presented at the 9th Annual EUROGRAPHICs (UK) Conference, Sheffield, UK. April 10-12, 1991.

*Department of Computing Science
University of Glasgow
Glasgow G12 *8QQ*, UK

‡Department of Mathematics
University of Glasgow
Glasgow G12 *8QQ*, UK

1. Introduction

In this paper we present a new way of achieving the effects of displacement mapping' in the rendering of three-dimensional objects. Displacement mapping is one of a range of texture mapping techniques[2] which add detail to the appearance of a parameterized three-dimensional surface by using the surface parameters to index a texture map of one or more shading parameter values. Typical entries in a texture map include colour[3,4] roughness[5], and transparency[6].

One texture mapping technique which is especially notable for being able to promote realistic detail is bump mapping[7]. Exceptionally, the texture map for bump mapping contains height value entries, which are not of themselves shading parameter values. The height values are used instead to determine the angle and direction to rotate the normal for the mapped surface and it is the normal which is used in the shading calculation. However, bump mapping is not suitable for textures whose height values are large enough to be distinguishable on the silhouette of the mapped object. In fact one of the terms in the equations describing how the normal is modified is discarded on the assumption that the heights in the bump map are negligibly small compared with the spatial extent of the surface[7,8]. Figure 1 shows an image of a sphere tiled with copies of a spike specified by the small bump map shown in Figure 3 and rendered using Blinn's bump-mapping technique, although in fairness to Blinn we should note that the fore-

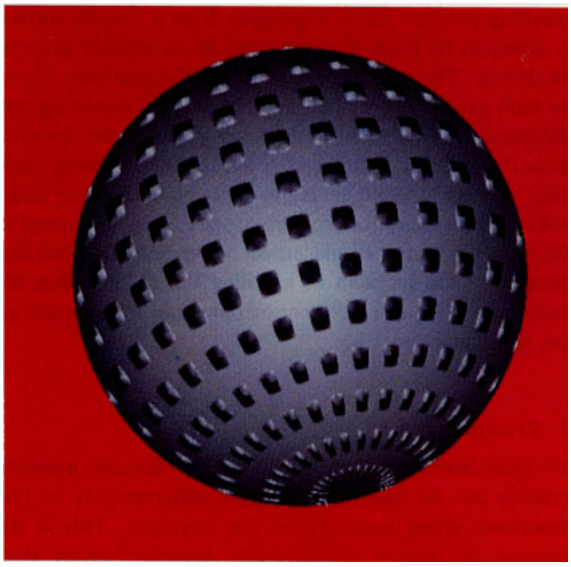

Figure 1. Rump-mapped sphere

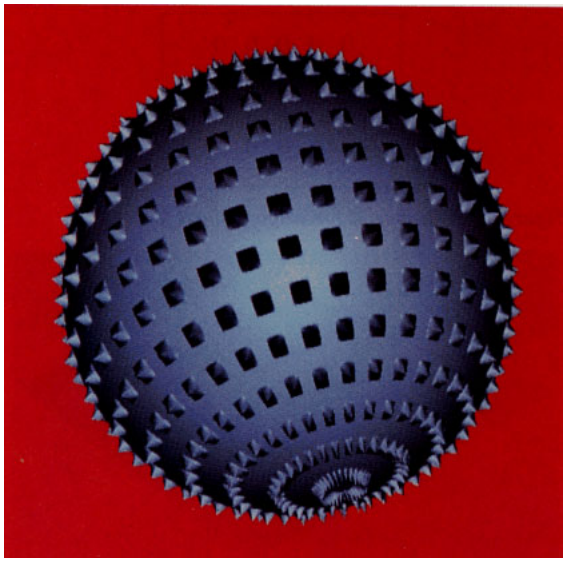

Figure 2. Displacement-mapped sphere

North-Holland
Computer Graphics Forum 10 (1991) 129-139

going assumption about relative dimensions has been clearly (and deliberately) invalidated. The viewer would expect the spikes near the sides of the sphere to break the silhouette yet this does not happen. We should also note that the interior of the sphere's image looks quite convincing nonetheless and it is a tribute to Blinn that he recognized just how far his simplifying assumptions could take him. In particular, he made no attempt to displace the surface or to achieve the identical effect to displacing the surface, which is what this paper is all about.

Figure 2 shows the same object as Figure 1, rendered from the same database, but now rendered with the form of displacement mapping to be described in this paper. The rendering method used throughout, in conjunction with the various texture mapping techniques illustrated here, is ray-casting with no attempt at anti-aliasing. When comparing the two figures we see immediately that the spikes in the vicinity of the pole and the silhouette edge of the mapped surface have a quite different appearance in each image. Close examination reveals that even the spikes near the centre do not have quite the same appearance. The reason for this can be seen in Figure 4 which shows a circumstance in which quite different solutions are obtained by the two methods for the equivalent probing ray. To obtain images using Blinn's formula on a surface S, as shown in Figure 4, for which a bump map defines displacements giving a bump M, an incident ray first intersects the surface at A, the bump map defines the height AB, and Blinn's formula determines the rotation of the normal $\mathbf{N}_1$ by θ_1 to $\mathbf{N}_1'$. By contrast, in displacement mapping (or inverse displacement mapping) the intersection of the ray with the surface defined by the same map is at point C and the normal is rotated in the opposite direction by an amount

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

Figure 3a. Tile entries for single-valued spike

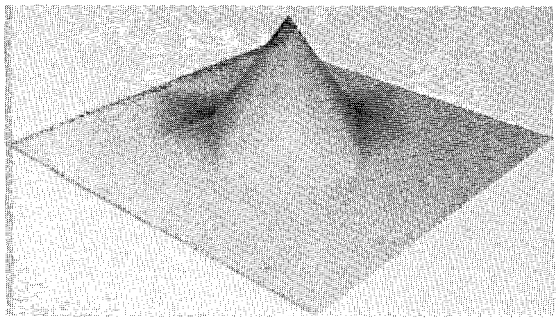

Figure 3b. Shaded image of spike tile

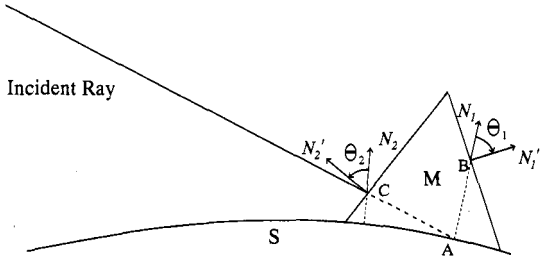

Figure 4. Comparative ray-intersection geometry

θ_2 to give a quite different result for the sample. Supersampling will not materially change this outcome.

Although displacement mapping[1,9] is described using the terminology of texture mapping and is regarded by some as being part of the rendering process, there is some controversy over this[10]. If all texture mapping techniques are described in terms of perturbing something, bump mapping perturbs normals, while displacement mapping perturbs the object's actual geometry. The references to displacement mapping are for the most part quite short on detail, treating displacement mapping as just another texture mapping technique and by implication subject to the usual problems, such as aliasing. However, to have a geometry to perturb, a polygon mesh fine enough to convey the appearance of a smooth surface has first to be built. Such a mesh could for example originate from a patch rendering process. Once built, the vertices of the mesh are then displaced as specified by the texture map along the normals defined there for the surface. Setting aside the question of whether the polygon mesh has to be subdivided further to match the resolution of the texture map, or whether the polygons themselves have to be split further to deal with the ambiguous non-planarities that result, the resulting polygon mesh originating from a smooth surface is going to be huge if constructed in its entirety prior to rendering. The real point here is that the geometry has to be built explicitly then perturbed before any rendering and the question is whether this is properly done in the modeller or in the renderer. Blinn[7], after all, introduced bump mapping in order to avoid having to explicitly model fine detail not only because of the problem of specifying such detail to the modeller, which displacement mapping avoids, but also because of the problem of handling the detail within the modeller, which displacement mapping, as implemented so far. does not avoid.

2. Principles

We show here that in some cases the displacement mapping process can be handled in a quite different way to that described, albeit sketchily, in the literature. This is the method of *inverse displacement mapping* which does everything in 'the opposite way' to the forms of displace-

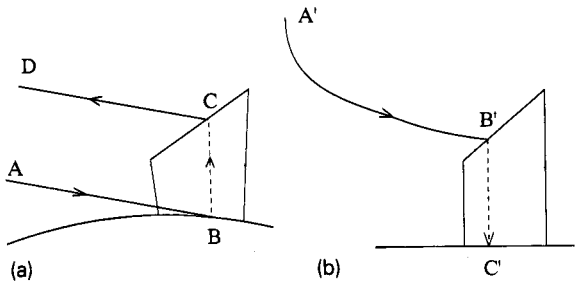

Figure 5. Object-space geometry for direct displacement mapping

ment mapping we have described so far. Essentially direct displacement mapping requires that texture height samples define patches in object space which are then projected into image space, as shown in Figure 5a, while inverse displacement mapping projects the rays used to collect image-space samples into texture space and determines the point of intersection as shown in Figure 5b. In Figure 5a it can be seen that the intersection point B of probe AB is actually used to determine the image value for some other point in image space somewhere along CD, while in Figure 5b it can be seen that the projected probe A'B' intersects the texture at B ' determining at C' the surface parameters needed to complete the calculation of the shade colours sought by the original probe.

As is done for (direct) displacement mapping we start with the vector equation for a smooth base surface $S(u,v)=0$ and a rectangular texture map of height values $T(u, v)$, both parameterized by a pair of coordinates u, v in the range $-1 \le u, v < 1$. The appearance of the surface resulting from applying the texture map to **S** is determined by probing within the projection of the region between the bounding volumes $\mathbf{S'}_{h_{max}}(u,v) = 0$ and $\mathbf{S'}_{h_{min}}(u,v) = 0$ onto the screen, where

$$\mathbf{S'}_h(u,v) = \mathbf{S}(u,v) + \hat{\nabla}\mathbf{S}(u,v) \,.\, h, \qquad (1)$$

$$h_{max} = \max_{u,v}(T(u,v)),$$

$$\text{and } h_{min} = \min_{u,v}(T(u,v))$$

with rays originating from an eye-point ***e*** and passing through a screen-point **s** paramaterized by ***t*** as

$$\mathbf{x} = \mathbf{e} + (\mathbf{e} - \mathbf{s}) \,.\, t \qquad (2)$$

Here we indicate vectors in bold and the normalization of vector **x** as $\hat{\mathbf{x}} = \mathbf{x}/|\mathbf{x}|$. The values of **S** for equation ***(2)*** lie within the projection of the region between $\mathbf{S'}_{h_{max}}()$ and $\mathbf{S'}_{h_{min}}()$.

The next step is to project the ray into texture space and thus determine whether or not it intersects the surface defined by the height samples there, and where. The resulting u, v values for the first intersection are then used to establish first the unit normal at parametric point u, v on the surface $S(u, v)$, which is $\mathbf{N} = \hat{\nabla}\mathbf{S}(u,v)$, and then the modification to the normal according to the Blinn[7] formula:

$$\mathbf{N'} = \mathbf{N} + a \,.\, \hat{\mathbf{D}} \qquad (3)$$

$$\text{where } \mathbf{D} = \left[\frac{\Delta T}{\Delta u}, -\frac{\Delta T}{\Delta v}\right] \cdot \begin{bmatrix} \mathbf{N}\times\mathbf{m} \\ \mathbf{N}\times\mathbf{l} \end{bmatrix}$$

$$\text{and } a = \begin{vmatrix} \frac{\Delta T}{\Delta u} & -\frac{\Delta T}{\Delta v} \\ \frac{\Delta T}{\Delta v} & \frac{\Delta T}{\Delta u} \end{vmatrix}^{\frac{1}{2}}$$

The vectors $\mathbf{l}, \mathbf{m}$ are the tangent components in the u and v directions, that is $\frac{\partial \mathbf{S}}{\partial u}, \frac{\partial \mathbf{S}}{\partial v}$. The derivatives of T in either parametric direction is approximated with differences $\frac{AT}{\Delta u}$ etc. Max[11] suggests a variation in this model, when calculating the shadows bump maps should produce if they were actual displacements, but, as will be seen, we derive shadows in a way which is unaffected by the normal perturbation formula so we have not used this variation here.

We can calculate shadows by firing a ray towards the light source (or sources) taking ***e*** as the point of intersection of the original ray with the texture in object space and **s** as a point along the vector towards the light source. This ray is again projected into texture space and the same procedure as before is applied to see whether it intersects any other part of the texture before passing out through the outer bounding volume. If so, the point of origin is in shadow for that light source, and, if not, the normal **N'** participates in the calculation for the illumination provided by that light source. An example of a sphere with displacement-mapped spikes with shadows calculated for a single light source can be seen in Figure 6a. The spike tile used is visualized in Figure 6b.

We contend that this form of displacement mapping is a true texture-mapping process because it can be carried out during rendering in a straightforward way. There is no need to build a model which is then displaced locally and rendered, and in fact all our images were produced from samples obtained in scan-line order. Direct displacement mapping, by contrast, would have to build a model and then render it at a later stage. In this context shadow-casting is problematical, but not impossible[12]. Like other texture-mapping methods our method is $O(n^2)$, and in particular our method converges on a solution in constant time for a given texture resolution, although it has to be said that our undeveloped renderer is consistently at least one order of magnitude slower than a similar bump mapping renderer. However, developments are in hand to exploit Mip-Mapping[13] to effect significant improvements in both speed and image quality.

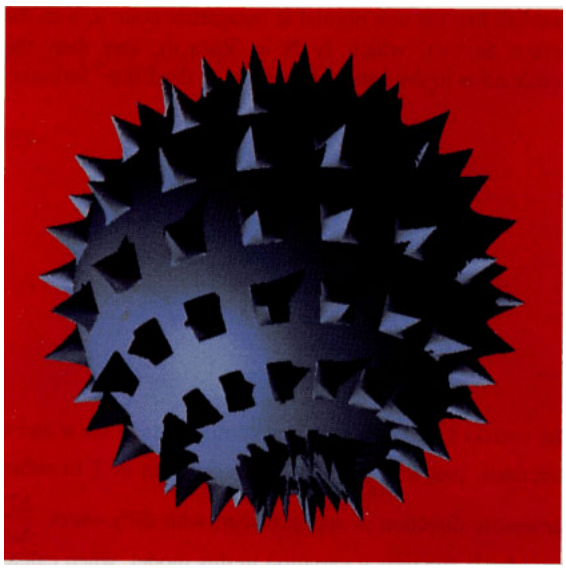

Figure 6a. Displacement map with shadows

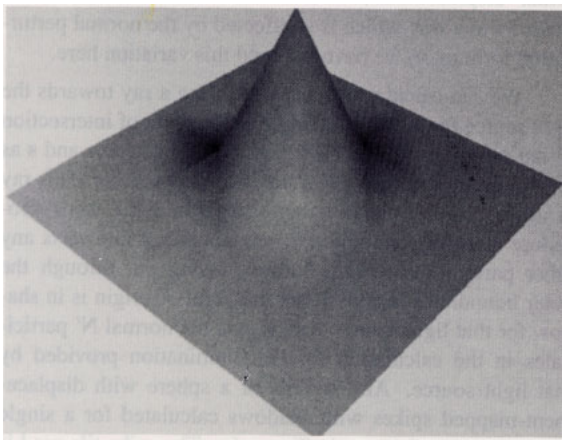

Figure 6b. Tilefor spike

3. Application to Quadrics

Our method for solving for the intersection of the ray in texture space is repetitive and relies upon bounding volumes as described in the next section. A reasonably efficient way of calculating the intersection of a ray of the form of equation (2) with a bounding volume as described by equation (1) is essential to the success of the method. If $S(u,v)$ is linear, that is, it defines a plane, then $\mathbf{S'}_h(u,v)$ also defines a plane displaced a distance h from the original. The case of inverse displacement mapping on a plane is straightforward, for example rays in texture space are still straight lines, and we will not deal with it further here. The methods we described are quite powerful enough to deal with this case as well.

If, however, $S(u,v)$ is quadratic then for arbitrary quadrics $\mathbf{S'}_h(u,v)$ is of degree six, and for $S(u,v)$ cubic, $\mathbf{S'}_h(u,v)$ is of degree ten. Neither of these cases are at all pleasant to deal with and we have avoided tackling even the general closed quadric form so far. However, for regular quadrics; spheres, semi-cones, and cylinders of circular section, the bounding volumes are all quadratic, being scaled or translated versions of the original surface. We have therefore, for the time being, restricted our attention to these surfaces, although quite satisfactory results can be obtained by subsequently applying non-uniform scaling to produce near-spheres etc., or by restricting the range of relative heights in the texture map T.

As introduced elsewhere (eg Hanrahan[14]), it is very convenient to be able to handle the many coefficients of a three-dimensional quadric in matrix notation, and also to use 4-space or homogeneous coordinates. The equation of a sphere radius g is given by:

$$\mathbf{x} \,.\, \mathbf{A}_g \,.\, \mathbf{x}^T = 0 \qquad (4)$$

where $\mathbf{x}$ is the homogeneous vector $[x, y, z, w]$ and the coefficient matrix is

$$\mathbf{A}_g = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -g^2 \end{bmatrix}$$

If we want to transform this sphere into something else we map each point $\mathbf{x}$ satisfying the equation (4) into the point $\mathbf{x'}$ on the new surface, by a transformation of the form

$$\mathbf{x} = \mathbf{x'} \,.\, \mathbf{M} \qquad (5)$$

where $\mathbf{M} = \mathbf{G.L.R.V}$ and $\mathbf{G}$ is the global scaling matrix, $\mathbf{L}$ the local scaling matrix, $\mathbf{R}$ the composite rotation matrix for rotations about the origin, and $\mathbf{V}$ the translation matrix. Since global scaling is normally carried in $\mathbf{A_8}$, we can take $G = \mathbf{I}$, the unit matrix, here. Postmultiplying (5) by $\mathbf{M}^{-1}$ gives

$$\mathbf{x} = \mathbf{x'} \,.\, \mathbf{M}^{-1} \qquad (6)$$

and substituting for $\mathbf{x}$ in **(4)** gives

$$\mathbf{x'} \,.\, \mathbf{M}^{-1} \,.\, \mathbf{A}_g \,.\, (\mathbf{M}^{-1})^T \,.\, \mathbf{x'}^T = 0$$

$$\text{or} \quad \mathbf{x'} \,.\, \mathbf{B}_g \,.\, \mathbf{x'}^T = 0 \qquad (7)$$

where $\mathbf{B}_g = \mathbf{M}^{-1} \,.\, \mathbf{A}_g \,.\, (\mathbf{M}^{-1})^T$. We can then make a sphere any radius we want by setting the appropriate value of g, and re-shape and reposition it by providing appropriate component matrixes for $\mathbf{M}$. Further, if there is no scaling inside $\mathbf{M}$, that is $\mathbf{L} = \mathbf{I}$, then, in our earlier notation,

$$\mathbf{S}() = \mathbf{x'}\mathbf{B}_g\mathbf{x'}^T$$

$$\text{and} \quad \mathbf{S'}_h = \mathbf{x'}\mathbf{B}_{g+h}\mathbf{x'}^T$$

To determine the intersections of rays of the form of (2) with *(7)* we substitute $\mathbf{e} + (\mathbf{s} - \mathbf{e}).t$ for $\mathbf{x'}$ yielding the quadratic

$$t^2(\mathbf{s}-\mathbf{e}) \,.\, \mathbf{B}_g \,.\, (\mathbf{s}-\mathbf{e})^T + 2t\,\mathbf{e} \,.\, \mathbf{B}_g \,.\, (\mathbf{s}-\mathbf{e})^T + \mathbf{e} \,.\, \mathbf{B}_g \,.\, \mathbf{e}^T = \mathbf{0} \qquad (8)$$

which gives solutions iff

$$|D| = \begin{vmatrix} \mathbf{e}\mathbf{B}_g\mathbf{s}^T & \mathbf{s}\mathbf{B}_g\mathbf{s}^T \\ \mathbf{e}\mathbf{B}_g\mathbf{e}^T & \mathbf{e}\mathbf{B}_g\mathbf{s}^T \end{vmatrix} \geq \mathbf{0} \qquad (9)$$

where D is the discriminant for the quadratic equation (8).

The equation $D = 0$ is also a quadratic (in screen coordinates x_0, y_0) which can be used to determine the scanline and y-bounds for rays which could intersect the mapped surface. The formulae for these are given in Appendix 1. For given x_0, y_0, values of t and hence $\mathbf{x}'$ can be obtained from (8).

The reason that we have been coy about the parameterization of **S** (and **S'**) is that these equations are really implicit equations, although we can turn them into the true parametric form by a modified form of the standard spherical polar mapping for a sphere. For a sphere radius g at the origin this is

$$\mathbf{x}^T = g \cdot \begin{bmatrix} \sin \pi u \cdot \cos\frac{\pi v}{2} \\ \sin\frac{\pi v}{2} \\ \cos \pi u \cdot \cos\frac{\pi v}{2} \\ 1 \end{bmatrix} \qquad (10)$$

where u,v lie within the usual parametric bounds $-1 \leq u < 1$, $-1 \leq v < 1$. It is evident from the form of (10) that a left-handed coordinate system is used with positive z going into the screen as is usual in 3-D graphics. The parametric form of the sphere equation is thus obtained by postmultiplying by **M** and applying equation **(5).** Usually we obtain surface or bounding volume intersections in terms of Cartesian coordinates and we need to obtain the corresponding u,v parameter values to continue the working in texture space. Safe formulae for deriving u,v are given in Appendix 2.

In order to be able to use the perturbation formula (3) we need to know the two vectors I, **m** ($\frac{\partial \mathbf{S}}{\partial u}$, $\frac{\partial \mathbf{S}}{\partial v}$ respectively) from which we can derive the surface normal for S at point u, v (or corresponding **x** or **x'**) as $\mathbf{N} = \mathbf{I} \times \mathbf{m}$. Here I and **m** are 3-space vectors, and since **N** is subsequently normalized, we can use equation **(5)** to derive them as

$$\mathbf{l} = \frac{\partial \mathbf{x}}{\partial u} \cdot \mathbf{M} \cdot \mathbf{I}_3, \quad \mathbf{m} = \frac{\partial \mathbf{x}}{\partial v} \cdot \mathbf{M} \cdot \mathbf{I}_3$$

$$\text{where} \quad \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Taking derivatives from equation (10), and substituting Cartesians, we get

$$\mathbf{l} = \pi g[-z, 0, x, 0] \cdot \mathbf{M} \cdot \mathbf{I}_3 \qquad (11)$$

$$\mathbf{m} = -\frac{\pi g}{2 \cdot (Q_2)^{\frac{1}{2}}}[Q_1, Q_2, Q_3, 0] \cdot \mathbf{M} \cdot \mathbf{I}_3$$

where $[Q_1, Q_2, Q_3] = [-z, 0, x] \times [x, y, z] = \frac{\partial \mathbf{x}_3}{\partial u} \times \mathbf{x}_3$

and $\mathbf{x}_3 = \mathbf{x} \cdot \mathbf{I}_3$.

Figure 7 shows the orientations of tangent vectors I and **m** with associated normal **N** on the surface of a sphere. If we take $\mathbf{M} = \mathbf{I}$ this sphere is the unit sphere at the origin. Taking g = 1 makes no difference for **l** and **m** if **N** is subsequently normalized. Solutions in object space involve solutions of equation (7), but equations (5) and (6) allow us to move in and out of another space in which the sphere is centred at the origin with a consistent size and orientation. This space, which we call 'sphere space', is more convenient to work with and we can use it to map both arbitrarily-sized and -oriented spheres and near-spheres without noticeable errors. However, the affine transformations carried in the matrix **M** do not preserve angles so solving in sphere space and transforming back into object space at the end does not give correct results if the sphere has been noticeably scaled in a non-uniform manner. In fact, this is the source of the restrictions we stated earlier.

The final step which is dependent on the geometry of **S** is the projection of the ray into texture space. This is done from three-dimensional' sphere space (that is, not invoking the homogeneous coordinate w) as the parameterization is straightforward there. Let us say that the ray in sphere space is given by

$$\mathbf{r} = \mathbf{p} + \mathbf{q}t \qquad (12)$$

where (in the terminology of equations (2) and **(5)),**

$$\mathbf{p} = \mathbf{e} \cdot \mathbf{M}^{-1} = [p_1, p_2, p_3],$$

$$\mathbf{q} = (\mathbf{s} - \mathbf{e}) \cdot \mathbf{M}^{-1} = [q_1, q_2, q_3]$$

When projected into texture space, the ray represented by equation (12) is no longer a straight line. In order to establish the projected ray's intersection with the texture we will be interested at all times in the texture space coordinates (u, v, h) of a typical point on the line. We are helped in this by the observation that the position vector **x** in sphere space is also a normal to the unit sphere at the origin. That is, all

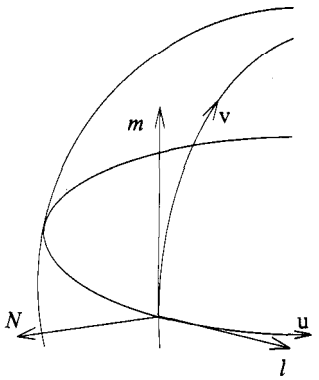

Figure 7. Tangent vectors l,m on sphere

points along the line of the vector have the same u, v coordinates and distances along it are texture-space h-values. Assuming we have these coordinates then we get h from (10) as

$$h = \frac{u}{\sin\frac{\pi v}{2}} - 1$$

Values of u and v can be obtained from **x** (or originally **x'**) using the safe equations in Appendix 2, with (6) if necessary.

We can obtain the equation of the projection of the ray onto the plane u, v directly as follows from an argument in sphere space. If **r** is a radius vector to the unit sphere at the origin which interesects the ray at point R (in figure 8), then it is given by equation (12). Figure 8 shows the geometry of the situation where the ray passes by the sphere (although this is not significant of itself). In the figure OE is the vector **p** in sphere space (originally **e** in object space) and the line ER gives the direction of the vector q (ES, with **S** being originally **s** in object space). Since **p x q** is normal to the plane OES,

$$(\mathbf{p}\times\mathbf{q}).\mathbf{r} = 0$$

Taking $\mathbf{p}\times\mathbf{q} = [P_1, P_2, P_3]$ in which case

$$P_1 = \begin{vmatrix} p_2 & p_3 \\ q_2 & q_3 \end{vmatrix},\quad P_2 = \begin{vmatrix} p_3 & p_1 \\ q_3 & q_1 \end{vmatrix},\quad P_3 = \begin{vmatrix} p_1 & p_2 \\ q_1 & q_2 \end{vmatrix},$$

and substituting for $\mathbf{r} = [x, y, z]$ using (lo), we get

$$P_1 . \sin\pi u . \cos\frac{\pi v}{2} + P_2 . \sin\frac{\pi v}{2} = P_3 . \cos\pi u . \cos\frac{\pi v}{2},$$

which simplifies to

$$v = \frac{2}{\pi}\tan^{-1}(k . \cos(\pi u + \alpha)) \qquad (13)$$

where $k = \dfrac{(P_1^2 + P_2^2)^{\frac{1}{2}}}{P_3}$ and $\alpha = \tan^{-1}\left[\dfrac{P_1}{P_3}\right]$.

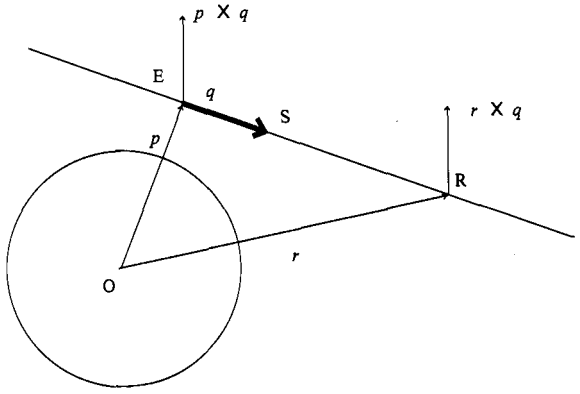

Figure 8. Geometry for computation of ray projection

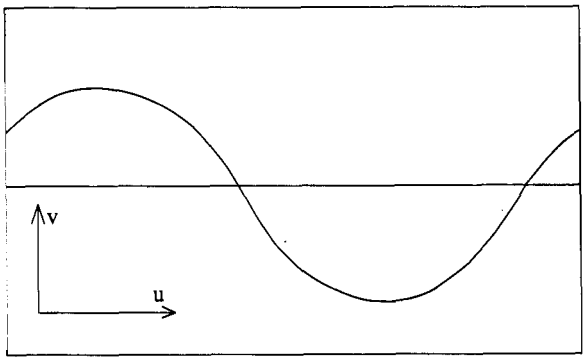

Figure 9. Appearance of a typical projection of a ray into texture space

The derivatives of this are

$$\frac{dv}{du} = -2k\cos^2\left[\frac{\pi v}{2}\right] . \sin(\pi u + \alpha) \qquad (14)$$

and

$$\frac{d^2v}{du^2} = 2k\,\pi\cos^2\left[\frac{\pi v}{2}\right](\sin(\pi u + \alpha) + \cos^2\pi v(\pi u + \alpha))$$

from which we can get slopes, maxima, minima, and turning points for the projection of the ray into the plane u,v. The typical appearance of a plot of equation (13) in texture space is shown in Figure 9.

From equation (13) we see that there are four cases which will guide an implementation, shown as i, ii, iii and iv in Figure 10. In the upper hemisphere ($u > 0$) this arises, i: when the ray passes through the axis of the sphere, ii: in the unexceptional cas: *not* near the edge of the hemisphere map, iii: when the ray passes near the pole, and iv: when the ray crosses the equator. A similar set of cases arise in the lower hemisphere ($u < 0$).

4. Implementation Issues

The mechanism for finding the intersection of the projected ray with the texture map $T(u,v)$ is the dominant component of the inverse displacement mapping algorithm and all the novel implementation issues revolve around this component. We rely on no particular assumptions about the surface whose sample heights are in the texture map other than that it is a C^0 continuous sheet. There is no require-

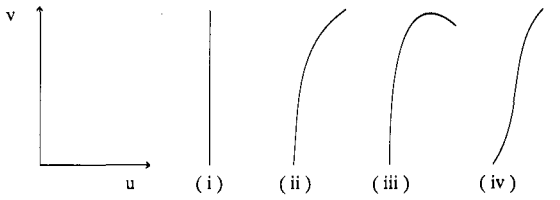

Figure 10. Projections of the ray onto the plane u, v

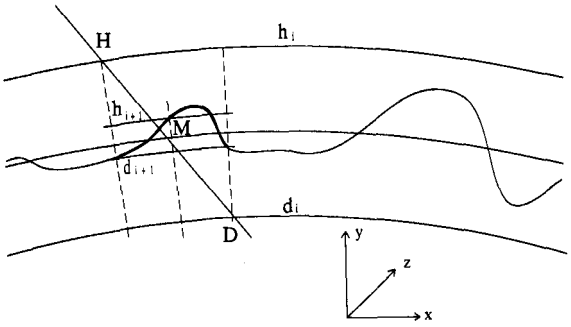

Figure 1I. Ray intersections with bounding volumes

ment, for example, for differentiability everywhere although the differences $\Delta T/\Delta u$, $\Delta T/\Delta v$ gloss over this point. The continuous sheet assumption is necessary to ensure that if we have two points on the projected ray and one is unambiguously above the local texture, that is, higher than all the heights around it, and the other is clearly below; that is, lower than all the heights around it, then there is guaranteed to be a point on the ray between them which actually intersects the texture surface.

Although there are a number of special cases that need to be examined in the end our description will start with the most common and straightforward type of intersection illustrated in Figures *10* ii, *11* and *12.* In this case the ray in sphere space intersects both of the bounding volumes, between which all intersections are constrained to lie, and has a trajectory in the *u, v* plane of the appearance shown in Figure *10* ii, that is, not like that in Figure 10 i, iii or iv. In all cases the algorithm finds intersections by first establishing the intersection of the ray with two bounding volumes in sphere space. These two bounding volumes just contain all of the height values for the portions of the texture within which an intersection could still be found. Initially these bounding volumes are defined by $A_{g+h_{max}}$, $A_{g+h_{min}}$ and the values of h_{max} and h_{min} are established *(a priori)* by searching the entire texture, as suggested by the discussion surrounding equation (1). This algorithm progresses by repetitively refining these bounding volumes until the *u, v* parameters of their intersections with the ray

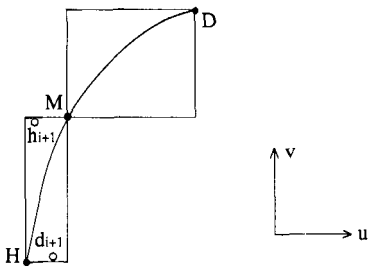

Figure 12. Texture fields for bounding volume calculations

in sphere space lie within a single texel (a region in texture space bounded by just four texture samples). These *u, v* values are then used to produce a synthetic *u, v* solution within the texel. The method used to produce the pictures in this paper was to synthesize a mid-point and solve for the intersection of the projected ray with one of the four triangles defined by each edge of the texel and the mid-point. The *u,v* values resulting from this intersection was then used to synthesize $\Delta T/\Delta u$, $\Delta T/\Delta v$ by linear interpolation.

The $i + 1^{th}$ repetitive step in the process leading to this conclusion starts with two bounding volumes, an outer bounding volume at height h_i above the unperturbed surface, and an inner bounding volume at a depth $-d_i$ below the unperturbed surface. The incident ray intersects these at points H and D in Figure 11. To guarantee convergence the intersection with a third surface typically midway between the outer and inner bounding volumes (at consistent height $(h_i + d_i)/2$) is also calculated. This gives us point M. These points H, M, D are now projected into texture space as shown in Figure *12* and the next part of the algorithm is carried out there. The pairs of points H, M and M, D can be seen in Figure *12* to define two rectangles of sample points in texture space each of which is available to establish new bounding volumes. The rectangle defined by HM is used first, and is searched to find new maxima and minima h_{i+1}, d_{i+1}. If h_M, the height for point M, is not greater than h_{i+1} then there is a possible intersection with the surface along arc HM. The values h_{i+1}, d_{i+1} are now used to establish new bounding volumes ($h_{i+1} < h_i$, $d_{i+1} > h_M > d_i$), and the process is repeated, starting with establishing a new half-way volume at $(h_{i+1} + d_{i+1})/2$. However, it is possible that h_M is greater than h_{i+1}, in which case no intersection is possible as the projection of the ray segment HM lies in this case wholly above the texture surface in this part of the texture space, so segment HM is abandoned and segment MD examined in the same way in its place. The repetition is most conveniently implemented recursively and if neither HM or MD yields an intersection solution then the recursive step returns to the previous level at which there may be an alternative to be tried.

The first difficulty with this process is illustrated by the *u, v* trajectory of Figure *10* iii, where there is a turning point in the trajectory between points H and D at some stage in the iteration. Such turning points can arise when $v = O$ (which fortunately is dealt with by the normal mechanisms) or $u = -\alpha/\pi$ in the terminology of equation **(13),** which gives us a trajectory like Figure 10 iii. In this case the search areas we have assumed in Figure *12* are wrong. This can be corrected by calculating tangents at H and M (or M and D) and seeing if they are of the same sign. If they are, the search area is correct, but if they are not we have to calculate the intersections of the tangents to the trajectory in texture space and determine the search box size as the smallest rectangle which encloses three points, the

two points for which tangents are determined, and the intersection of these tangents. This works everywhere except where the points M,D straddle the equator (i.e. the v coordinates have opposite signs). In this case there is a point of inflexion at the equator which could invalidate the foregoing method of calculating a search box. The solution here is to take the equator as the mid point and calculate the u value for $v = 0$. This is given by

$$h = \frac{1}{2} - \frac{1}{\pi}\tan^{-1}\left[\frac{P_1}{P_3}\right] \quad \text{if } P_3 < P_1$$

$$= \frac{1}{\pi}\tan^{-1}\left[\frac{P_3}{P_1}\right] \quad \text{otherwise}$$

Another case which can cause the tangent test to give invalid results is the one in which $P_2 \cong 0$. In this case the slopes of the tangents are infinite (or close to it) and this is caused by the ray trajectory being in the plane containing the axis of the sphere in sphere space. In texture space this could appear either as a continuous vertical trajectory between two points U and D with the same u coordinate, or a split vertical trajectory which goes to or comes back from the top or bottom boundary of the texture map at $v = \pm 1$, in which case the u coordinates for U and D will differ by exactly 1.

All these special cases have to be handled separately for the case in which the incident ray does not intersect the initial inner bounding volume. This could typically involve features seen near the silhouette edge. These rays also have trajectories in u, v space but they might not lead to intersections. The process starts by splitting the ray in two at the point at which the ray is nearest to the inner bounding volume which, in sphere-space, will be the midpoint between the two intersections with the other bounding volumes. Each half of the ray is then examined separately starting from the segment which has the outer bounding volume intersection nearest to the eyepoint $\boldsymbol{e}$. The mid point substitutes for the inner bounding volume intersection in the first repetition of the ray -surface intersection finder. Although this case has to be handled separately, it only causes problems when the ray also crosses the equator. In this case the ray has to be split into three segments, separated internally by the $v = 0$ point and the mid-point.

5. Conclusion

The kind of results we get with this algorithm can be seen in Figures 3, 6, 13 and 15. Despite appearances to the contrary, Figures 13 and 15 are both spheres which have been displacement mapped by our method. The texture maps in each case are slightly different. In Figure 13 the entries are all real numbers calculated by reverse-engineering a cube. The texture map for Figure 13 visualized here as a smooth-shaded surface, is shown in two views in Figure 14a, and b. The sharp edges and comers have been

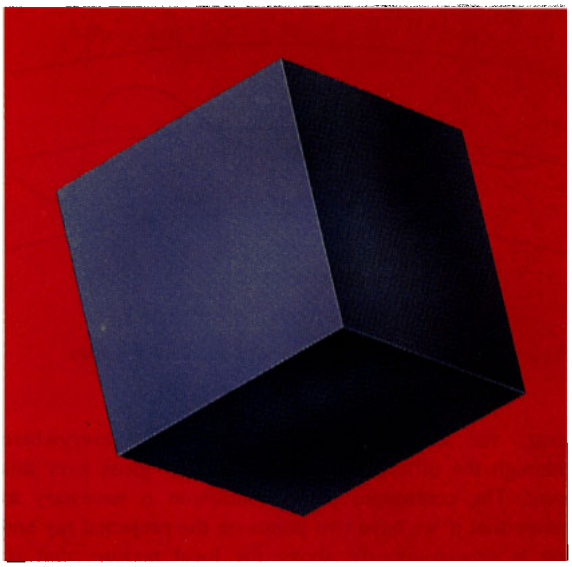

Figure 13, Another inverse displacement mapped sphere

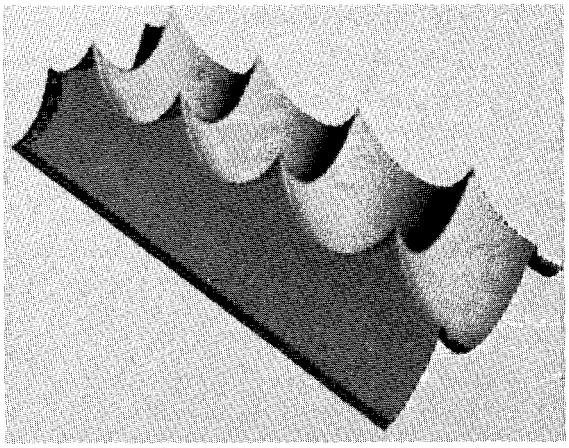

Figure 14a. Perspective view of texture map for Figure 13

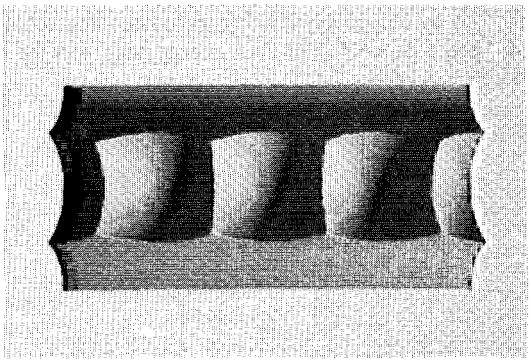

Figure 14b. Plan perspective view of texture map for Figure13

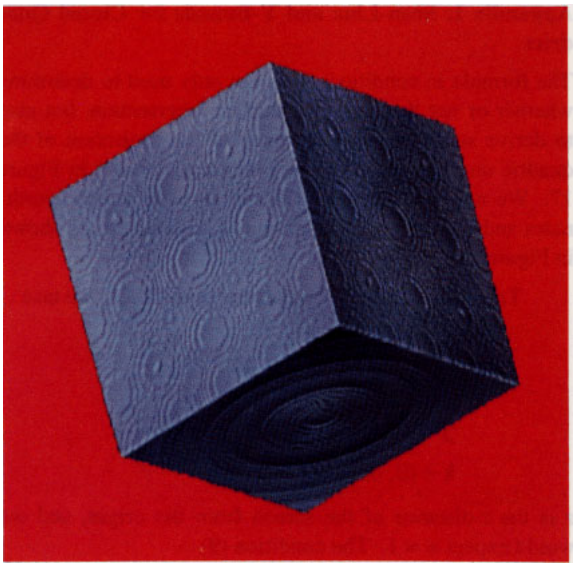

Figure 15. Sphere mapped with cube texture with truncated integer heights

Figure 16. Bump-mapped sphere using same map as Figure 15

retained only by virtue of the fact that the texture is a high-resolution texture of 800 x 400 texels. The outcome, in applying this texture to a sphere, is Figure 13 which is a scaled version of the cube used to determine the texture samples in the first place.

The slight bending of the large flat areas is caused by the use of linear interpolation to reconstruct the height sample within a texel. This results in a slight error in the $\Delta T/\Delta u$, $\Delta T/\Delta v$ terms which affects the final normal direction and results in the systematic variation in the shading value which shows up as an apparent bend. The effect is not caused by the approximations in the Blinn formula since the terms which are assumed to be zero are in fact exactly zero for the special case of normal perturbation on a sphere. It is instead a true aliasing effect because the errors reinforce each other rather than cancel one another out. A high-frequency variation is reappearing as a low-frequency one in texture-normal space.

In Figure 15, the texture map entries were produced in the same way, but in this case the real number height values were *truncated* to integers. The attractive patterns on the surface are caused entirely by truncation error, and the intriguingly inconsistent shading is caused by a similar kind of normal aliasing effect to that in Figure 13. It should be noted that neither the texture nor the surface it is being mapped on to are flat anywhere. One curved surface is being 'bent' onto another so that the curvature is neutralized. Nobody would model a cube like this and nobody would notice the slight error resulting from using linear interpolation (as Blinn[3], who also used linear interpolation, observes). However the problem could be solved directly by using cubic interpolation.

It may seem a little disappointing to have to work so hard to produce results for only a limited class of surfaces, but the various special cases can be dealt with, albeit more slowly, by more general methods which can be applied to a wider range of surfaces. Although our method is still not well developed, and quite slow as a consequence, it is still fast compared to the brute-force divide-and conquer technique one of us tried in order to see if it would be practicable to avoid handling any of the special cases. It wasn't. We note that Kajiya[15] used a non-linear space in which to ray-trace volumes of rotation. However, his space was designed just to simplify the computation rather than to correspond to some other specific space in which picture information could be exploited. Our method will also extend to texture-mapped cylinders, all the shapes Kajiya's method can handle, and asymmetric cylindrically-centred shapes as well.

To finish, Figure 16 shows the application of Blinn's bump-mapping technique to the database used to produce Figure 15. The remarkable thing about this picture is that the result is easily recognizable as a corner of a cube. However, some consequences of not performing any displacement can be clearly seen in that the visible parts of the 'edges' of the cube can be seen to be curving towards the edges of the (unperturbed) silhouette. Thus these two pictures (Figures 15 and 16) complement the pair (Figures 1 and 3) in showing the relative merits of displacement mapping over bump mapping in any form.

6. Acknowledgements

Two of us (SGH and JWP) gratefully acknowledge a 1988 Tektronix travelling scholarship in support of this work.

One of us (JRL) acknowledges a studentship from the SED and a research studentship from SERC during the time he has been involved in this work. We would also like to thank Colin Dunlop (Computing Science) for the shaded images of textures in Figure 2, 14a and 14b, and also our anonymous referees for their helpful suggestions.

References

1. Cook, R., "Shade Trees," *Computer Graphics, proc SIGGRAPH '84* 18(3), pp. 223-231 (July 1984).
2. Heckbert, P., "Survey of Texture Mapping," *IEEE CG & A*, pp. 56-67 (November 1986).
3. Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD Thesis, University of Utah (1976).
4. Blinn, J.F. and Newell, M.E., "Texture and Reflection in Computer Generated Images," *Comm ACM* 19(10), pp. 542-547 (Oct 10'6).
5. Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," *Computer Graphics, proc SIGGRAPH '77* 11(3), pp. 192-198.
6. Gardner, G.Y., "Visual Simulation of Clouds," *Computer Graphics, proc SIGGRAPH '85* 19(7), pp. 297- 303 (July 1985).
7. Blinn, J.F., "Simulation of Wrinkled Surfaces," *Computer Graphics, proc SIGGRAPH '78* 12(2), pp. 286-292 (August 1978).
8. Watt, A., *Fundamentals of Three-Dimensional Computer Graphics*, Addison-Wesley, Reading, Mass. (1989).
9. Upstill, S., *The RenderMan™ Companion*, Addison-Wesley, Reading, Mass (1990).
10. Heckbert, P., "Texture Mapping and Image Filtering Notes," in *State of the Art in Computer Graphics, International Summer Institute*, Edinburgh (1990).
11. Max, N.L., "Horizon Mapping Shadows for Bump-Mapped Surfaces," *The Visual Computer* **4(2),** pp. 109-117 (1988).
12. Reeves, W., et al., "Rendering Anti-Aliased Shadows with Depth Maps," *Computer Graphics, proc SIGGRAPH '87* 21(4), pp. 283-291.
13. Williams, L., "Pyramidal Parametrics," *Computer Graphics, proc SIGGRAPH '83* pp. 1-11.
14. Hanrahan, P., "A survey of Ray-Surface Intersection Algorithms," in *An Introduction to Ray Tracing*, ed. Glassner, A., Academic Press, San Diego (1989).
15. Kajiya, J.T., "New Techniques; for Ray Tracing Procedurally- Defined Objects," *Computer Graphics, proc SIGGRAPH '83* 17(3), pp.91-102 (July 1983).

Appendix 1: Scan-Line and Y-Bounds for Closed Quadrics

The formula in condition (9) is not only used to determine whether or not there is a ray-surface intersection, but also to derive scan-line and y-bounds for the projection of the quadric onto the screen. The geometry is shown in Figure 17. We note that the vector **S** contains the screen coordinates and they can be brought out as components as shown in Figure 17.

To get the scan-line bounds for scan-line y_0 we take

$$\mathbf{S} = x\mathbf{i} + y_0\mathbf{j} + \mathbf{k}$$

where $\mathbf{i} = [1, 0, 0, 0]$,

$\mathbf{j} = [0, 1, 0, 0]$,

$\mathbf{k} = [0, 0, z, w]$,

z is the z-distance of the screen from the origin, and we would expect $w = 1$. The condition (9)

$$|D| = 0$$

can now be expressed as a quadratic equation in x whose solutions (x_{min}, x_{max}) are the bounds for the scan-line $y = y_0$. If this quadratic is

$$x^2 . a_x + x . b_x + c_x = 0$$

then

$$a_x = Y_0 = \begin{vmatrix} \mathbf{eBi}^T & \mathbf{iBi}^T \\ \mathbf{eBe}^T & \mathbf{eBi}^T \end{vmatrix}, \text{ noting that } \mathbf{iBi}^T = b_{11}$$

$$b_x = 2 . (Y_1 . y_0 + Y_2)$$

$$c_x = Y_3 . y_0^2 + 2 . Y_4 . y_0 + Y_5$$

and

$$Y_1 = \begin{vmatrix} \mathbf{eBi}^T & \mathbf{jBi}^T \\ \mathbf{eBe}^T & \mathbf{eBj}^T \end{vmatrix}, \quad Y_2 = \begin{vmatrix} \mathbf{eBi}^T & \mathbf{kBi}^T \\ \mathbf{eBe}^T & \mathbf{eBk}^T \end{vmatrix},$$

$$Y_3 = \begin{vmatrix} \mathbf{eBj}^T & \mathbf{jBj}^T \\ \mathbf{eBe}^T & \mathbf{eBj}^T \end{vmatrix}, \quad Y_4 = \begin{vmatrix} \mathbf{eBj}^T & \mathbf{jBk}^T \\ \mathbf{eBe}^T & \mathbf{eBk}^T \end{vmatrix},$$

$$Y_5 = \begin{vmatrix} \mathbf{eBk}^T & \mathbf{kBk}^T \\ \mathbf{eBe}^T & \mathbf{eBk}^T \end{vmatrix},$$

We can now get the y-bounds as the values of y (y_{max}, y_{min}) for which the equation in x has equal roots. That is when

$$\begin{vmatrix} b_x & 2 . a_x \\ 2 . c_x & b_x \end{vmatrix} = 0$$

This equation is of the form

$$y^2 . a_y + y . b_y + c_y = 0$$

where

$$a_y = \begin{vmatrix} Y_1 & Y_0 \\ Y_3 & Y_1 \end{vmatrix}, \quad b_y = \begin{vmatrix} Y_1 & Y_0 \\ Y_4 & Y_2 \end{vmatrix}, \quad c_y = \begin{vmatrix} Y_2 & Y_0 \\ Y_5 & Y_2 \end{vmatrix}.$$

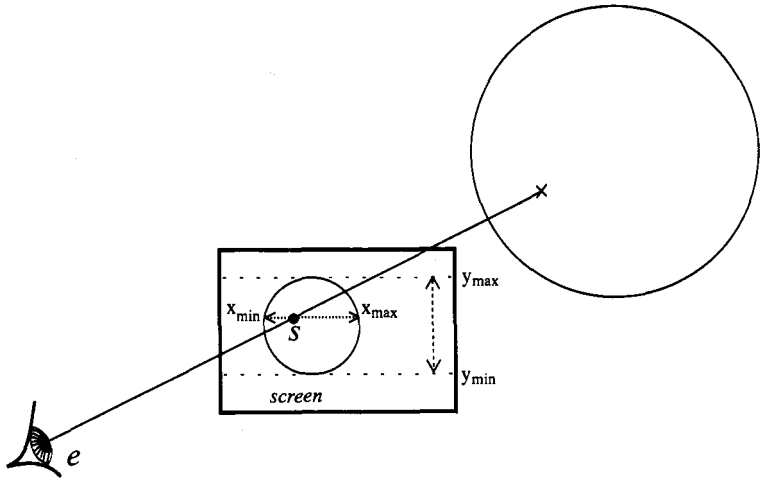

Figure 17. Geometry for scan-line and y-bounds calculations

Appendix 2: Safe Calculation of *u, v* from x,y, *z*

The safe formulae for *u* and *v* given here are calculated for **x** rather than **x'** in the terminology of equations (6) and (7). Equation (7) does the conversion from **x'** to **x.** If $\mathbf{x} = [x, y, z, 1]$ that is **x** is divided through by *w*, then equations for *u, v* corresponding to the parametric equations (10) can be computed as follows:

If

$$|z| \le |x|, \qquad u = \frac{1}{\pi}\tan^{-1}\left(\frac{z}{x}\right) + \frac{Sign(x)}{2}$$

$$|z| > |x|, z > 0 \qquad u = \frac{1}{\pi}\tan^{-1}\left(-\frac{x}{z}\right) + Sign(x)$$

$$|z| > |x|, z < 0 \qquad u = \frac{1}{\pi}\tan^{-1}\left(-\frac{x}{z}\right)$$

$$|y| \le |(x^2+z^2)^{\frac{1}{2}}| \qquad v = \frac{2}{\pi}\tan^{-1}\left(\frac{y}{(x^2+z^2)^{\frac{1}{2}}}\right)$$

$$|y| > |(x^2+z^2)^{\frac{1}{2}}| \qquad v = \frac{2}{\pi}\tan^{-1}\left(-\frac{(x^2+z^2)^{\frac{1}{2}}}{y}\right) + Sign(y)$$

where $Sign(a) = -1$ if $a < 0$
$= 0$ if $a = 0$
$= +1$ if $a > 0$