GRUMMAN AEROSPACE CORPORATION
BETHPAGE, NEW YORK 11714

Oct 2, 1984

Dear Oliver

Here is the corrected copy of "Beethoven's 6th in C/G". I hope this will suit your purposes.

I am delighted that you would like to show my work and I would like to know how it works out.

Sincerely

Jeff Gardner
MS A08-35
(516) 575-4791

GRUMMAN AEROSPACE CORPORATION
BETHPAGE, NEW YORK 11714

8/16/84

Dear Olivier,

Here are 3 video tapes of my Computer Image Generation work.

"Beethovens Sixth in CIG" is an original tape and my latest & best, but there may be technical video problems.

The X-29 tape is a copy (I need the original Kill Sept. 13) showing sequences of an advanced aircraft in flight.

The IMAGE 3 tape is a copy (I need the original Kill Sept 5) showing training applications such as low level attack and maps of the earth flight.

I hope this material is satisfactory. If I can be of any further service please let me know.

Sincerely

Jeff Gardner
MS A02-35
(516)575-4791

P.S. Please return the tapes as soon as it is convenient.

Simulation of Natural Scenes Using Textured Quadric Surfaces

Geoffrey Y. Gardner

Grumman Aerospace Corporation
Research & Development Center
Bethpage, New York 11714

ABSTRACT

Because of the high complexity of the real world, realistic simulation of natural scenes is very costly in computation. The topographical subtlety of common natural features such as trees and clouds remains a stumbling block to cost-effective computer modeling. A new scene model, composed of quadric surfaces bounded with planes and overlaid with texturing, provides an efficient and effective means of representing a wide range of natural features. The new model provides a compact and functional data base which minimizes the number of scene elements. Efficient hidden surface algorithms for quadric surfaces bounded by planes are included. A mathematical texturing function represents natural surface detail in a statistical manner. Techniques have been developed to simulate natural scenes with the artistic efficiency of an impressionist painter.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - curve, surface, solid and object representations; geometric algorithms, languages and systems; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - color, shading, shadowing, and texture; visible line/surface algorithm; J.7 [Computers in Other Systems]: Military, real time.

1. INTRODUCTION

Realistic simulation of natural scenes is one of the greatest challenges facing computer graphics. Exact mathematical representation of the extreme complexity of nature is generally not cost-effective because of the high computation load. Nonetheless, a wide range of applications, including training, scientific modeling, and entertainment, have created a demand for effective and efficient computer techniques for simulating natural scenes. In attacking the problem, the choice of data base is critical. A great deal of effort has been applied employing a wide range of data bases including points, planar surfaces, and curved surfaces.

The point is mathematically the simplest data base primitive. Dungan [6] and Spooner, et al 16 have developed techniques to generate perspective images of Defense Mapping Agency elevation data points. Csuri, et al [5] have used points to model smoke, and Reeves [13] has used points to model fire and grass. The problem with the point data base is that very large numbers of scene elements have to be transformed to a perspective projection, producing extremely high computation loads.

The next simplest data base primitive, and the most commonly used, is the planar face bounded by straight edges [17]. Because of its mathematical simplicity this linear approach has allowed real-time implementation and is widely used in flight simulation [15]. An elegant application of the linear data base has been the construction of fractal surfaces to model terrain with unprecedented realism [7,9]. Marshall, et al [10] have also used a linear data base to model trees with great detail. Like the point data base, however, the linear data base requires very large numbers of scene elements to represent the nonlinear complexity of the real world. The realism of fractal surface images is achieved only by rendering hundreds of thousands of planar faces, and a single linear tree requires thousands of faces. The number of scene elements is critical to the efficiency of the image generation approach. The greater the number of scene elements, the greater the number of surfaces and boundaries that must be computed, resulting in greater computation costs for sorting, priority determination, and antialiasing. Limiting the number of scene elements limits the realism of the linear model. For this reason, current flight simulation systems have been criticized for being too cartoonish.

Various approaches to scene simulation using curved surfaces have been developed. Quadric surfaces have been used effectively by the Mathematical Applications Group, Inc. (MAGI) [8] and the New York Institute of Technology (NYIT) [18] to model complex man-made objects. MAGI also used a large number of quadric surfaces to model a tree. In a landmark application of computer image image generation, Blinn used quadric surfaces with texture maps to model Jupiter and its moons for the Voyager flyby [2]. Quadrics have also been used to model molecules and are common data base primitives in CAD/CAM. None of these applications, however, has exploited the potential of quadric surfaces for modeling a wide range of natural features.

More complex curved surfaces, in particular bicubic surfaces, have been studied extensively [1,3,4]. Such surfaces provide great modeling flexibility because they allow for continuity of slope between adjoining surfaces. However, despite the development of several clever image generation algorithms, the mathematical complexity of these surfaces results in severe computation loads for complex scenes.

2. AN EFFICIENT DATA BASE FOR SIMULATING NATURAL SCENES

In selecting a data base for simulating natural scenes, we must contend with the inevitable tradeoff between realism and computation load. We must define a level of realism that we desire and then choose the data base that will produce this level of realism most efficiently. Efficiency is particularly important when our goal is to generate long sequences of images to produce dynamic presentations. To achieve real-time image generation, efficiency is critical. In defining a desired level of realism which would reduce the computation problem to manageable proportions, we have adopted an approach used successfully by impressionist painters: to represent the essence of natural scenes as simply as possible. An impressionist painter produces a very effective picture of a tree by representing the shape and texture of its foliage without expending effort on precise delineation of individual twigs and leaves. With this modest level of realism, the painter quickly produces a strong impression of the essence of a tree. A similar result can be produced by a computer using simple curved surfaces and texture patterns. Quadric surfaces, in particular, lend themselves to this approach because a single quadric surface can be used to model a natural scene feature such as a tree, a hill, or a rock. This simplifies scene modeling and reduces the number of scene elements required to model complex scenes. Furthermore, quadaric surfaces are mathematically the simplest form of curved surface and therefore provide an efficient means of representing natural topographical curvature without piecewise-linear approximation.

Texture patterns can be mapped onto quadric surfaces to simulate natural scene detail. The most commonly used technique in texture mapping is to store images of texture patterns [1,3]. This approach is inefficient for simulating natural scenes because too much storage would be required to represent the wide variety of patterns necessary. In addition, arbitrary views of natural scenes require perspective manipulation and antialiasing of the texture patterns. Furthermore, in dynamic presentations it is desirable to include complex motion of texture patterns to simulate flowing rivers and blowing trees. To provide on-line control of perspective validity, antialiasing, and dynamics, we produce texturing by means of a mathematical texturing function which can be mapped onto scene surfaces in such a way as to modulate surface shading and translucence. Because only 25 parameters are required to define a natural-looking texture pattern, a wide variety of patterns can be stored.

2.1 The Geometric Data Base

A geometric data base composed of quadric surfaces only would not provide the topographical variety required for natural scenes. To provide greater flexibility in our topographical model and to allow for adjoining quadric surfaces, we include in our geometric data base the option of bounding each quadric with a finite number of planes. We then define the geometric data base to be a set of discrete convex objects, with each object defined by one quadric surface and N bounding planes, where N can be zero. This avoids the costly computation of intersections between quadrics and ensures that all surface boundaries will be at most second-order, allowing for scan-line intercept determination in closed form.

We define a three-dimensional scene coordinate space (X_s,Y_s,Z_s), with the X_s axis pointing east, the Y_s axis pointing north, and the Z_s axis pointing vertically up. We define a ground plane ($Z_s = 0$), and a unit light vector. For each two-dimensional image of the three-dimensional scene, we define an eyepoint and look angle. The coefficients of the quadric surface and bounding planes for each object are defined in scene coordinates and must be transformed to eye coordinates (X,Y,Z), centered at the eyepoint (0,0,0), with the Y axis pointing in the direction the eye is looking (Fig. 1).

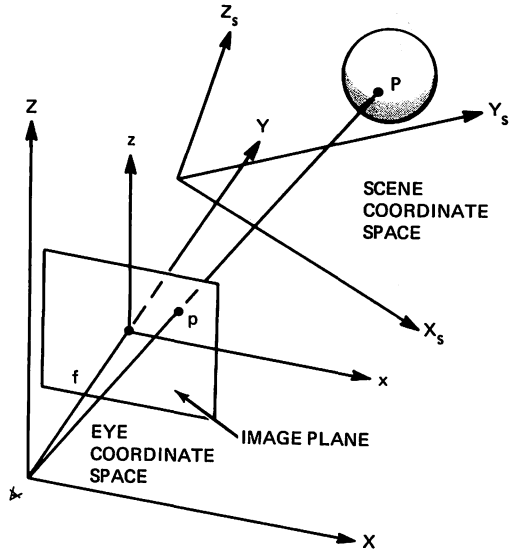


Fig. 1 Transformation From Scene to Eye Coordinates & Projection Onto Image Plane

For each quadric, we get an equation in eye coordinates of the form

$$Q(X,Y,Z) = Q_1X^2+Q_2Y^2+Q_3Z^2+Q_4XY+Q_5YZ+Q_6XZ +Q_7X+Q_8Y+Q_9Z+Q_0 = 0 \quad (1)$$

For each plane, we get an equation of the form

$$P(X,Y,Z) = B_1X+B_2Y+B_3Z+B_4 = 0 \quad (2)$$

The surface geometry of a quadric surface bounded by an arbitrary number of planes can be quite complex. Generating an image of such an object requires exact determination of which surface is visible at each pixel, but testing each surface at each pixel is inefficient. We can make use of area coherence and scan line coherence by noting that a given surface will cover many pixels and many scan lines. We can greatly reduce visibility computation by noting that there are far fewer boundary points than there are surface points in a typical image, and that boundary information can be used to determine surface visibility. The key to efficient processing of the geometric data base is, then, to determine which portions of the boundary curves are visible in the image.

We define an image plane with coordinates (x,z) parallel to the XZ plane a distance f in front of the eyepoint such that the Y axis pierces the coordinate axes origin. We also define the image x axis to be parallel to the eye coordinate X axis and the image z axis to be parallel to the eye coordinate Z axis (Fig. 1). Then the transformation from eye coordinates to image coordinates can be represented as

$$X = kx$$
$$Y = kf$$
$$Z = kz \qquad (3)$$

Our strategy will now be to use Eq (1), (2), and (3) to project all surface boundaries onto the image plane. We will then determine all key points on each boundary, that is points at which boundary visibility, and therefore surface visibility, changes across scan lines. We will then use the key points to determine a scan line list of visible boundaries and surfaces.

The most important image curve is the limb curve, defined as the projection of the quadric silhouette (Fig. 2). The limb curve can be derived by substituting Eq (3) into Eq (1) to obtain a quadratic equation in k,

$$Ak^2 + Bk + C = 0 \qquad (4)$$

where A is a second-order expression in the image coordinates, x and z, B is a linear expression in x and z, and C is a constant. (Algebraic expansions will be omitted for the sake of brevity.) The parameter k relates to the distance from the eye, varying from a value of 0 at the eyepoint to a value of 1 at the image plane, and increasing along the ray out into the eye coordinate space. In general, a ray will intersect a quadric surface at two points, giving two distinct values of k from Eq (4). By definition, the limb curve is the set of image points for which the rays are tangent to the quadric surface. For these points k is single valued, so the discriminant of Eq (4) is zero.

$$B^2 - 4AC = 0 \qquad (5)$$

This then gives the limb curve as

$$f(x,z) = a_1x^2+a_2z^2+a_3xz+a_4x+a_5z+a_6 \qquad (6)$$

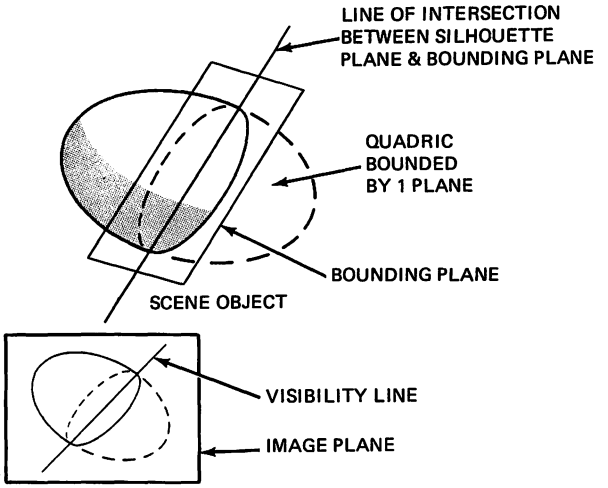


Fig. 2 Limb Curve, Intersection Curve and Visibility Line

where the coefficients are expressions containing only the quadric surface coefficients, and f, the distance from the eye to the image plane.

The remaining image curves will be intersection curves, that is, projections of the curves of intersection between the quadric and its bounding planes (Fig. 2). To solve for the coefficients of an intersection curve, we must satisfy Eq (1), (2), and (3) simultaneously. Substituting Eq (3) into Eq (2), we get

$$k = -B_4/(B_1x+B_2f+B_3z) \qquad (7)$$

Then substituting for k in Eq (3) and using the result in Eq (1), we get the intersection curve as

$$g(x,z) = e_1x^2+e_2z^2+e_3xz+e_4x+e_5z+e_6 = 0 \qquad (8)$$

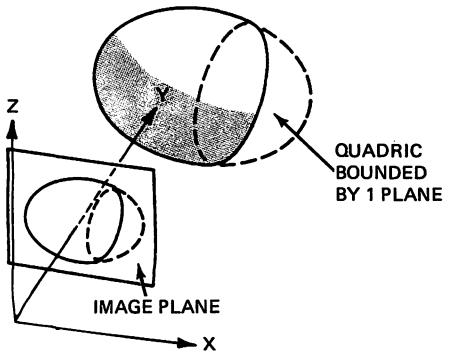
where the coefficients are all expressions containing the quadric surface coefficients, the bounding plane coefficients, and f.

Given a quadric surface with one or more bounding planes, we must determine which boundary curve segments are visible. To do this, we introduce the concept of a visibility line. We define a visibility line for each bounding plane as the projection on the image plane of the line of intersection between the quadric silhouette plane and the bounding plane (Fig. 2). Since the visibility line will be used to partition a particular curve into visible and invisible segments, its definition must include a sense or sign determined from the sense of the bounding plane. Rewriting Eq (2) as an inequality to include the bounding plane sense, we have

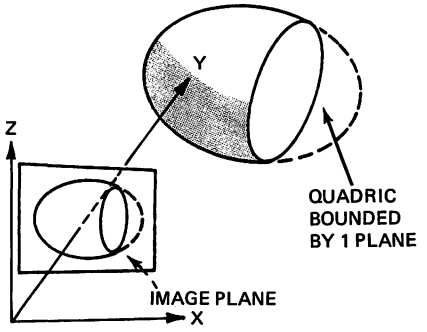
$$P(X,Y,Z) = B_1X+B_2Y+B_3Z+B_4 > 0 \qquad (9)$$

Substituting Eq (3) into Eq (9) gives

$$k(B_1x+B_2f+B_3z) + B_4 > 0 \qquad (10)$$



a. Bounding Plane Not Visible



b. Bounding Plane Visible

Fig. 3 Intersection Curve Visibility

The equation for the silhouette plane can be determined from Eq (4) and (5) to be

$$kB + 2C = 0 \tag{11}$$

Since B is a linear expression in x and z, and C is a constant, Eq (10) and (11) can be combined to give the visibility line as

$$v_1x + v_2z + v_3 > 0 \tag{12}$$

where the coefficients are expressions in the quadric surface and bounding plane coefficients and f.

The visibility line defined by Eq (12) can be used to determine the visibility of any point on the limb curve relative to a particular bounding plane. If the limb point satisfies Eq (12), it is defined to be visible relative to the plane used to define the line. If more than one bounding plane is used in the object definition, a limb point must satisfy the visibility line for each in order to be visible in the image.

The visibility line for each bounding plane can also be used to determine the visibility of any point on the intersection curve related to that plane. This visibility test is only necessary when the plane faces away from the eye, that is, when the eye is on the same side of the plane as the defined part of the quadric surface (Fig. 3a). When the eye is on the other side of the plane, no visibility test is necessary because the whole intersection curve is visible (Fig. 3b). To determine which case applies, we simply substitute the eyepoint (X,Y,Z) = (0,0,0) into Eq (9) to get

$$B_4 > 0 \tag{13}$$

If Eq (13) is satisifed, the eye is on the object side of the plane and a visibility test for the intersection curve is required. Figure 2 shows that in this case, the visible portion of the intersection curve lies on the opposite side of the visibility line as the visible portion of the limb curve. Therefore, for a point on the intersection curve to be visible, it must fail to satisfy Eq (12).

Thus, the visibility line can be used to determine portions of the limb curve defined to be visible as well as portions of the intersection curve whose visibility depends on eye position in the scene.

Images of objects with more than one bounding plane may include linear boundaries resulting from the intersection of two planes. We introduce the concept of an intersection line, which we define as the image of the intersection between two bounding planes. The intersection line will define that portion of the intersection curve of one bounding plane defined to be visible relative to the other (Fig. 4). In this sense, the intersection line is analogous to the visibility line with the first intersection curve replacing the limb curve. With a derivation similar to that used for the visibility line, we get the intersection line for two bounding planes as

$$x_1x + x_2z + x_3 > 0 \tag{14}$$

where the coefficients are in terms of the coefficients of the two bounding planes.

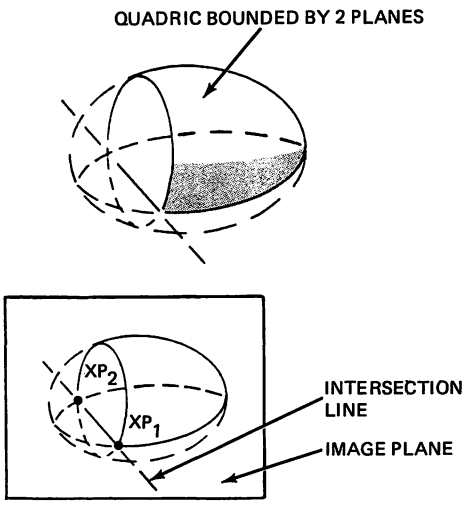


Fig. 4 Intersection Line Defines Intersection Curve Visibility Relative to Intersecting Bounding Plane and Contains Intersection Points XP_1, XP_2

In addition to its function as a visibility criterion, an intersection line may also be a visible image boundary. In order for a point on an intersection line to be visible it must satisfy the appropriate form of Eq (14) for all other intersection lines produced by a common bounding plane.

Having computed the equations for all image boundary curves and lines, we must now determine which boundary segments are visible. We note that boundary segments are generally visible over many scan lines and that their state of visibility changes only at certain key points. Determining these key points and restricting visibility tests to a few sample scan lines will greatly reduce visibility computations. The key points of boundary visibility are curve extrema (minimum and maximum z), contact points, intersection points, and triplet points.

Curve extrema can be computed from Eq (6) and (8) in the standard manner. Since curves don't exist on scan lines outside their extrema, they clearly are potentially visible only on scan lines in between.

We define a contact point as a point of tangency between the limb curve and an intersection curve. In general, two contact points exist for each intersection curve and can be determined from the intersection of the relevant visibility line and the limb curve (Eq (12) & (6)). Contact points are points of potential change of visibility state for limb and intersection curves.

We define an intersection point as a point of intersection between an intersection line and an intersection curve related to a common plane (Fig. 4). Intersection points are points of potential change of visibility for intersection curves and intersection lines.

We define a triplet point as the image point of the corner intersection of three bounding planes. Triplet points can be computed as the intersection of two intersection lines and are points of potential change of visibility for intersection lines.

Once we have computed all the potentially visible curves, lines, and key points for the image of an object, we are prepared to perform visibility tests. In determining what tests to make for particular boundaries, the following rules apply. Any image point relating to a point on the quadric surface must be tested against all visibility lines. Any image point relating to a point on a particular bounding plane must be tested against all intersection lines related to that plane (except for the line producing the point) and, if the plane is not visible, the visibility line related to that plane.

We begin the visibility testing by testing all key points, since only visible key points affect visibility changes in the boundaries. Editing out invisible key points, we compile a list of visible key points sorted on z in scan-line order. We call this a z-band list because it defines regions in the image z direction (vertical) in which boundary visibility remains constant from scan line to scan line.

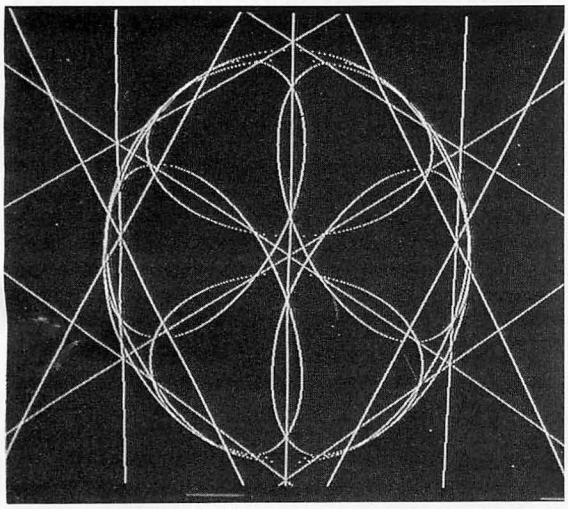
Within each z-band in the list, an average z value is computed to represent a typical scan line on which all boundary curve and intersection line intercept points are computed. These intercept points are tested for visibility, and codes for all visible boundary intercepts are entered in the z-band, sorted from left to right based on the intercept x value. Boundary curve intercepts are coded to define leftmost or rightmost intercept. Finally the sorted visible boundary intercepts are used to determine the visible surfaces in between. The surface between two boundary intercepts is assumed to be the quadric surface unless both boundaries are related to a common bounding plane.

The resulting object visibility list, consisting of sorted z-bands, each with x-sorted boundary intercept and surface codes, provides a very efficient image "blueprint" for directing scan-line surface shading of the object. As the image is generated, scan line by scan line, a particular object is considered only if the scan line falls within the z-band list. For each such object, the pertinent z-band is referenced, and, proceeding from left to right, boundary codes are referenced, intercepts computed, and surfaces shaded on pixels in between. Figure 5 shows how this approach simplifies the image generation of a complicated object by reducing a maze of potentially visible boundaries to a manageable list of visible boundaries and surfaces. Figure 5a shows the limb and intersection curves, and the intersection and visibility lines in an image of a sphere bounded by 6 planes. Figure 5b shows the z bands indicated by horizontal lines drawn through all visible key points. Note how within each z band the visibility of boundaries and surfaces remains constant. Figure 5c shows the final shaded object with hidden surfaces removed.

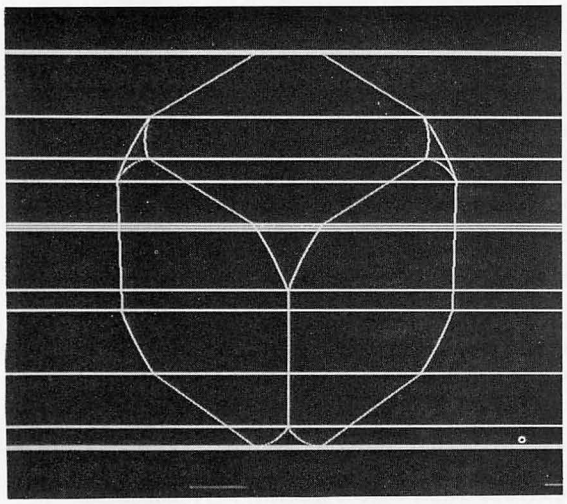
In addition to streamlining intraobject visibility, that is, the determination of visible surfaces for a single object, the visibility list simplifies interobject visibility, the determination of priority between different objects. Because our geometric data base includes only convex objects, we need not compute the distance to the visible object surface at every pixel. Instead, we can use the z-band lists to compute the leftmost and rightmost surface distance on a scan line and use linear interpolation in between. Then, when two objects overlap on a scan line we can use the interpolated distances to determine priority within the overlap region.

2.2 The Texture Data Base

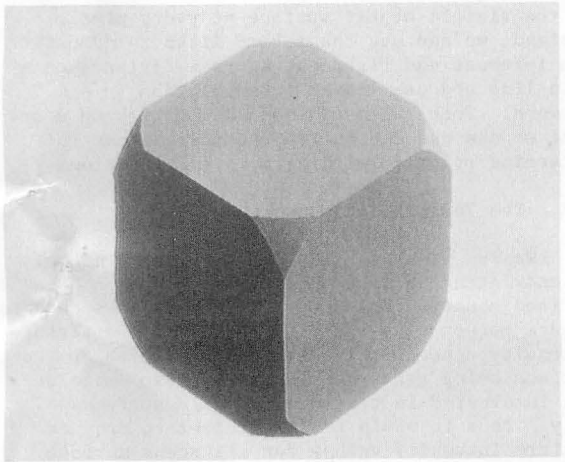
We can simulate textural detail efficiently by modulating surface shading intensity in a defined manner. In so doing, we must take care to assure perspective validity by making the texture intensity depend on the scene coordinates of the surface being textured. For any given image we are interested in texturing visible surfaces only. Thus it would be inefficient to produce texture intensity values for all scene surface points. The most practical approach is to produce texture values only for visible scene surface points corresponding to image sample points. Since the image sample points depend on the viewing perspective, we must be able to produce



a. All Curves & Lines



b. Visible Boundary Segments & Z Bands



c. Shaded Visible Surfaces

Fig. 5 Visibility Determination for Sphere With 6 Bounding Planes

texture values for arbitrary scene points. For this reason we have chosen a mathematical function to produce texture. A mathematical texturing function offers the additional advantage of requiring a minimal data base to produce a wide variety of texture patterns, each of which can cover an unlimited region in the scene. The control inherent in a mathematical function, computed during image generation, also provides for straightforward antialiasing of texture patterns and allows implementation of complex texture motion.

In choosing the exact form of our texturing function, we decided that it would be most efficient to represent real-world detail at a statistical level. An effective way to do this is to use the principle of Fourier expansion, [11, 14]. After investigating different expressions of various waveforms, we found a very effective texturing function to be defined as

$$T(X_s,Y_s,Z_s) = \sum_{i=1}^{n} C_i \frac{[(\mathrm{Sin}(\omega_i X_s + PX_i) + 1]}{2} \times \sum_{i=1}^{n} C_i \frac{[(\mathrm{Sin}(\omega_i Y_s + PY_i) + 1]}{2} \quad (15)$$

Where PX_i and PY_i represent phase shift functions to avoid a tartan-like regularity of the pattern. We have found that defining PX_i as a sinusoidal function of Y_s and PY_i as a sinusoidal function of X_s produces natural-looking patterns for low values of n (Fig. 6).

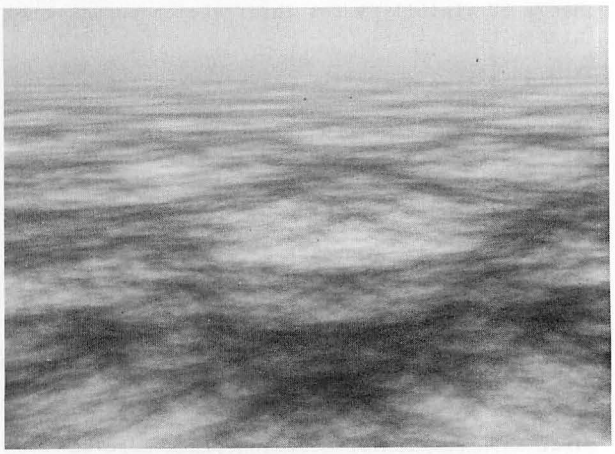


Fig. 6 Texture Pattern on Ground Plane From Texture Function Eq (15) With n = 7

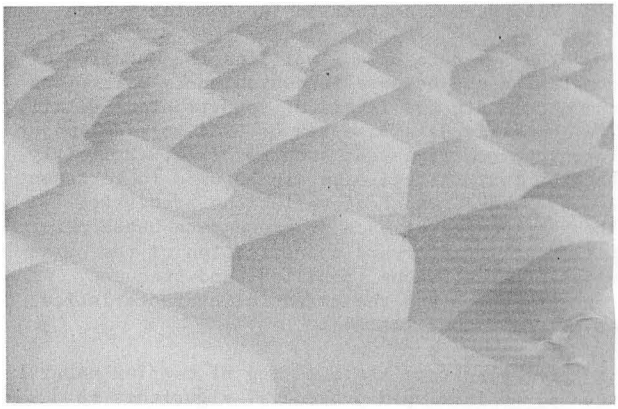
The primary use of the texturing function is to simulate surface detail by modulating shading intensity. This is done by computing a weighted average of the surface shading intensity and the texture function value at each visible point. A texture weighting parameter is defined for each object to provide flexibility in scene modeling. A secondary use of the texture function is to simulate boundary irregularity and amorphousness of certain natural features, such as trees and

clouds. We accomplish this by treating locally dark texture regions on an object's surface as though they were holes in the object. This effect is achieved quite simply by assigning a threshold value for the texture function and defining an object to be translucent at any image point where the texture function falls below the threshold. The artificial boundaries produced between the visible and invisible portions of the texture surface can be softened by varying the translucence linearly as the texture function crosses the threshold. This technique is demonstrated in Fig. 7, which shows a sky plane textured with variable translucence to simulate a cloud layer.

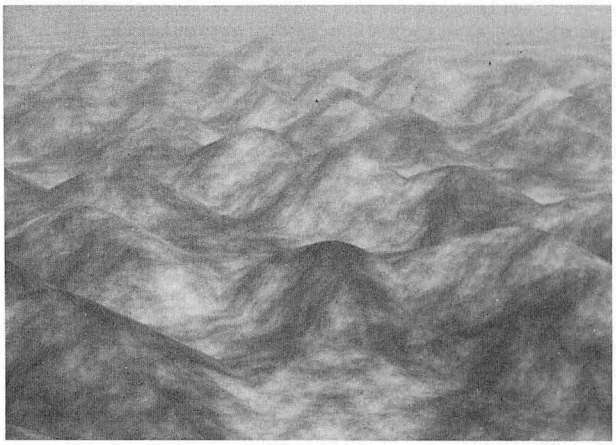
Fig. 7 Cloud Layer Simulated by Texture Function Modulating Shading & Translucence

The texturing function greatly enhances the realism of objects defined by the geometric data base. Figure 8a shows mountains modeled by the geometric data base only, and Fig. 8b shows the same scene enhanced by the texturing function. In addition to adding simulated topographical detail, the texturing blends surface shading across boundaries between abutting objects. This unifying effect is due to the fact that all objects, as well as the ground plane, are textured with the same texture function parameters so that the texturing function maps the same pattern continuously across all scene surfaces as a function of scene coordinates.

The combination of the geometric and texture data bases is particularly effective in simulating amorphous objects, such as trees and clouds, whose boundaries are both complex and subtle (Fig. 9). The trick of simulating such features so efficiently is to control the translucence at an object's silhouette. This capability is provided as much by the geometric data base as by the texturing function, for it is the definition of the limb curve (Eq 6) that allows us to vary the threshold of the texturing function to increase translucence in a straightforward manner at image points near the object boundary. Because the translucence can be increased smoothly and continuously, the image will have soft boundaries which will require no antialiasing.



a. Scene Without Texture Function



b. Scene With Texture Function

Fig. 8 Mountains Modeled by Hyperboloids

The control inherent in the mathematical texturing function has two more advantages over less flexible texturing techniques, such as stored texture maps. Antialiasing of texture patterns can be achieved simply by testing sine wave frequencies and dropping those that exceed the image sampling frequency (projected into scene space). In addition, any of the texturing function image parameters can be varied from frame to frame, allowing the simulation of a wide range of dynamic effects. Thus it would require little additional frame computation to simulate trees blowing, smoke rising, clouds drifting, or rivers flowing.

The texture data base required to implement this simulation capability is extremely modest. A natural-looking texture pattern can be defined by 25 parameters, including sine coefficients and frequencies, phase shifts, and translucence thresholds. A given texture pattern can be used for any number of objects covering any region in the scene. Thus, all trees of a particular type could be simulated using one pattern, all rivers using another pattern, etc. We have found that complex and varied natural scenes can be simulated effectively using only 10 texture patterns. This compactness of the texture data base simplifies both scene modeling and image generation.

2.3 Data Base Construction for Complex Scenes

The new data base simplifies the modeling of complex scenes because it conveniently partitions the model into two levels of topographical detail. The geometric data base can be used to model major topographical features, such as hills, explicitly, and the texture data base can be used to represent secondary topographical variations statistically. The compactness of each of these data bases permits a straightforward specification of the parameters of size, shape, position, and frequency content, which are the essential characteristics of natural scene features.

To facilitate the modeling of complex natural scenes, we developed procedural algorithms to generate clusters of scene features. Only two types of quadric surfaces are required to produce a wide variety of scene features. Hyperboloids of two sheets are very effective in simulating hills and mountains, and ellipsoids are efficient for modeling trees, rocks, and clouds. For each feature cluster, we define a region on the ground plane over which the cluster, will be generated. We also define a typical spacing between features in the cluster as well as size, shape, and color parameters for a typical template object. The algorithm places the template object within the defined region at positions determined from the defined spacing. As the object modeling each feature is generated, the algorithm perturbs its position, size, shape, and color parameters randomly to produce natural statistical variations within the cluster. Adjacent objects are tested for intersection, and bounding planes are computed for abutting objects. This permits the modeling of topographic structures, such as rolling terrain, mountain ranges, and forests, which are too complex to be simulated by isolated objects. The algorithm also allows us to define features in a cluster as "terrain objects" upon which other scene objects will lie. Terrain objects are generated first so that the objects composing subsequent clusters can be raised to the appropriate terrain elevation after they are positioned on the ground plane. The simplicity of quadric surface shape and position definition makes this process quite straightforward. Similarly, clusters can be defined to be positioned at a fixed altitude above the ground plane, a capability useful in modeling clouds.

To use the mathematical texturing function to model secondary topographical variations we must define sets of function parameters, with each set chosen to simulate a desired texture pattern. The individual parameters in each set can be determined from an analysis of the spatial frequency content of features being modeled. In general, natural features have a power spectrum whose amplitude decreases as frequency increases. Natural-looking texture patterns can be generated using from 3 to 7 sine waves whose frequencies increase by a factor of approximately 2 and whose amplitudes decrease by a factor of approximately one half the square root of 2. The complete texture data base is defined by a list of parameter sets. As clusters of scene features are generated, each object is assigned a specified texture parameter set number. A particular texture pattern can be assigned to any number of clusters, minimizing the size of the overall data base.

The clustering algorithm can be extended to generate clusters of clusters. Using the extended clustering algorithm, we can quickly generate and change models of complex natural scenes. Figures 10 and 11 show two examples of complex natural scenes simulated using these algorithms.

3. CONCLUSIONS

We have described efficient hidden surface algorithms for complex curved objects composed of quadric surfaces bounded by planes. We have demonstrated the effectiveness of a texturing function which modulates the shading intensity and translucence of scene surfaces. We have shown how these tools can be incorporated in procedural algorithms to simulate complex natural scenes efficiently.

Textured quadric surfaces provide a means of bridging the gap between computationally cheap, but cartoonish, scene simulation and highly realistic, but costly, scene simulation. Textured quadric surfaces produce a compact, functional data base related directly to the most significant topographical characteristics of scene features. This approach reduces image generation computation because it reduces the number of scene elements that must be processed. Textured quadric surfaces allow us to represent the essential realism of natural scenes as an impressionist painter would, thus avoiding the costly replication of unimportant details. The new scene model is particularly effective for modeling amorphous objects, such as trees and clouds, which continue to be stumbling blocks for other approaches. The new model is, however, comprehensive because it can model man-made as well as natural features. The inclusion of bounding planes even permits modeling linear features, such as buildings.

As a quantitative measure of the computational efficiency of this approach, the images in Figs. 10 and 11 took 10 min. 29 sec. and 9 min. 40 sec., respectively, to generate on a dedicated Data General Eclipse S/250 16-bit minicomputer with 512 KBYTE memory and floating point accelerator. Image resolution is 480 scan lines by 640 pixels by 24 bits. The image generation routines were programmed in Fortran V using floating point arithmetic. (Runs for figure 10 at 512 x 512 resolution took 5 min. 34 sec. on a VAX 11/760 and 6.49 sec. on a CRAY 1M.) The current programs are in no way claimed to be optimal. On the contrary, there is much room for improvement in both computation time and image quality, and the author hopes that this paper will stimulate others to explore and extend this technology.

Scene simulation using textured quadric surfaces has application in many diverse fields, including art, entertainment, advertising, scientific simulation, and training. At Grumman we have used it in a public relations film to show an advanced concept, forward-swept-wing aircraft in flight before it was built (Fig. 12). We have also applied the technology to pattern recognition research in target tracking by a missile [12]. We

Fig. 9 Trees, Clouds & Hills Modeled by Textured Quadric Surfaces

Fig. 10 Clusters of Hills & Trees Generated With Cluster Algorithm

Fig. 11 Clusters of Hills, Trees & Clouds Generated With Extended Cluster Algorithm

Fig. 12 Frame From X-29 Promotional Film. The X-29 aircraft is modeled by 40 quadric surfaces without texturing. The clouds are modeled by quadric surfaces with texturing.

are currently investigating real-time implementation of the algorithms for flight simulators.

4. ACKNOWLEDGEMENTS

Significant contributions to this work were made by Tom Jacquish, Ed Berlin, Jr, Bob Gelman, and Mike Gershowitz. I am indebted to my wife Joan for pointing out that hyperboloids make better mountains than paraboloids.

The research described in this paper was sponsored in part by the Air Force Human Resources Laboratory (AFHRL) under contract F33615-79-C-0029.

5. REFERENCES

1. Blinn, J., Computer display of curved surfaces, PhD Thesis, Computer Science Department, U. of Utah (Dec 1978).

2. Blinn, J., Voyager 2. SIGGRAPH Video Review, Issue 1, May 1980.

3. Catmull, E., A subdivision algorithm for computer display of curved surfaces. UTEC-CSc-74-133, PhD Thesis, Computer Science Department, U. of Utah (Dec 1974).

4. Clark, J., Designing surfaces in 3-D. Comm. ACM 19, 8 (Aug 1976), 454-460.

5. Csuri, C., Hackathorn, R., Parent, W., Carlson, W., and Howard, M. Towards an interactive high visual complexity animation system. Computer Graphics 13, 2 (Aug 1979), 289-299.

6. Dungan, W., Jr., A terrain and cloud computer image generation model. Computer Graphics 13, 2 (Aug 1979), 143-147.

7. Fournier, A., Fussell, D., and Carpenter, L., Computer rendering of stochastic models. Comm. ACM 25, 6 (June 1982), 371-384.

8. Goldstein, R. A., and Nagel, R., 3-D visual simulation. Simulation 16, 1 (Jan 1971), 25-31.

9. Mandelbrot, B. B., Fractals: Form, Chance and Dimension. Freeman, San Francisco, (1977).

10. Marshall, R., Wilson, R., and Carlson, W., Procedure models for generating three-dimensional terrain. Computer Graphics 14, 3 (July 1980), 154-159.

11. Max, N., Vectorized procedural models for natural terrain: waves and islands in the sunset. Computer Graphics 15, 3 (Aug 1981) 317-324.

12. Mendelsohn, J., and Leib, K. G., Optical correlation module design study. Air Force Systems Command Contract AF 33615-82-C-1727 Final Report (Sept 1983).

13. Reeves, W. T., Particle systems - a technique for modeling a class of fuzzy objects. Computer Graphics 17, 3 (July 1983), pp 359-376.

14. Schacter, B. J., Long-crested wave models. Computer Graphics and Image Processing 12 (1980), 187-201.

15. Schacter, B. J., Computer Image Generation. Wiley-Interscience, New York, 1983.

16. Spooner, A. M., Breglia, D. R., and Patz, B. W., Realscan - a CIG system with increased image detail. Proc 2nd Interservice/Industry Training Equipment Conf, Salt Lake City, Utah (Nov 1980) 110-116.

17. Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., A characterization of ten hidden surface algorithms. ACM Computing Surveys 6, 1 (May 1974), 1-55.

18. Williams, L., Casting curved shadows on curved surfaces. Computer Graphics 12, 3 (Aug 1978), 270-274.

August 6 1984

Geoffrey Y Gardner
Grumman Aerospace Corp
MS A-08/35
Bethpage
NY 11714

Dear Mr Gardner

Thank you for your letter of 6 March and the photographs and papers enclosed with it.

Having seen the films you presented at your talk and at the film show at SIGGRAPH, I think the best way to put over your work would be as film. Would it be possible to show some sequences in our gallery The Computer and the Image? I would like sequences that show the various terrain, tree and cloud texture as well as some that show aircraft in the scene.

The film would be shown in a section on simulation; we use 3/4 inch U-Matic video tape players.

I enclose the latest issue of The Computer Museum Report and a brochure prepared for our capital campaign to give you some background on the Museum.

Thank you for any help you might offer.

Yours sincerely

Dr Oliver Strimpel
Curator

enclosures

Grumman Aerospace Corporation
Bethpage, New York 11714

Research and Development Center
6 March 1984

Mr. Oliver Strimpel
Computer Museum
300 Congress Street
Boston, MA 02210

Dear Mr. Strimpel:

Enclosed is a copy of the paper I submitted to the SIGGRAPH '84 Conference. I am also enclosing a brief description of my approach to Computer Image Generation and a couple of photos illustrating the results.

Thank you for your interest. If I can be of any further help, please let me know.

Sincerely,

Jeff Gardner

Geoffrey Y. Gardner
Staff Scientist
M.S. A-08/35

GYG:amp
Enc.

Grumman Aerospace Corporation
Bethpage, New York 11714

GRUMMAN ADVANCED COMPUTER IMAGE GENERATION TECHNOLOGY

The Grumman Aerospace Corporation has developed unique and Advanced Computer Image Generation (CIG) techniques to generate realistic images of synthetic scenes. Our techniques are particularly effective in representing natural features such as terrain, trees, and clouds, which are stumbling blocks for other CIG approaches. Using simple curved surfaces overlaid with texturing, we are able to portray the essential curvature and natural detail of the real world with artistic efficiency. Because we developed our techniques for eventual real-time implementation, we have kept computation to a minimum. As a result, we can generate images of rich and complex scenes in a fraction of the time required by other CIG techniques. In addition, the simplicity of our geometric data base facilitates scene modeling, and our unique mathematical texturing function allows a straightforword means of producing dynamic special effects such as weapon explosions and smoke.

We model a scene using a geometric data base composed of quadric (second-order) surfaces bounded by planes. We add natural detail by means of a mathematical texturing function which modulates surface shading and translucence. We have written scene modeling software that allows quick creation and manipulation of clusters of features such as trees, hills, and clouds. Our techniques include the capability for the arbitrary orientation and movement of objects such as aircraft. We have developed an efficient antialiasing algorithm to remove quantization effects such as staircasing, and we have developed efficient image generation algorithms to produce dynamic, movie type sequences of images. Typical computation time for a single full-color image of 480 scan lines with 640 pixels per scan line and 24 bits per pixel is 10 minutes on a Data General Eclipse S/250 16[illegible] bit minicomputer.

SIMULATION OF NATURAL SCENES
USING TEXTURED QUADRIC SURFACES

Geoffrey Y. Gardner

Grumman Aerospace Corporation
Research & Development Center
Bethpage, New York 11714

ABSTRACT

Because of the high complexity of the real world, realistic simulation of natural scenes is very costly in computation. The topographical subtlety of common natural features such as trees and clouds remains a stumbling block to cost-effective computer modeling. A new scene model, composed of quadric surfaces bounded with planes and overlaid with texturing, provides an efficient and effective means of representing the essential realism of a wide range of natural features. The new model provides a compact and functional data base which minimizes the number of scene elements. A mathematical texturing function represents natural surface detail in a statistical manner. Techniques have been developed to simulate natural scenes with the artistic efficiency of an impressionist painter.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation - display algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - curve, surface, solid and object representations, geometric algorithms, languages and systems; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - color, shading, shadowing, and texture, visible line/surface elimination; J.7 [Computers in Other Systems]: Military, real time.

1. INTRODUCTION

Realistic simulation of natural scenes is one of the greatest challenges facing computer graphics. Exact mathematical representation of the extreme complexity of nature is generally not cost-effective because of the high computation load. Nonetheless, a wide range of applications, including training, scientific modeling, and entertainment, have created a demand for effective and efficient computer techniques for simulating natural scenes.

In attacking the problem, the choice of data base is critical. A great deal of effort has been applied employing a wide range of data bases including points, planar surfaces, and curved surfaces. Dungan [6] and Spooner, et al [16] have developed techniques to generate perspective images of Defense Mapping Agency elevation data points. Csuri, et al [5] have used points to model smoke, and Reeves [13] has used points to model fire and grass. The problem with the point data base is that very large numbers of scene elements have to be transformed to a perspective projection, producing extremely high computation loads.

The most commonly used data base is the linear data base, composed of planar faces bounded by straight edges [17]. Because of its mathematical simplicity this approach has allowed real-time implementation and is widely used in flight simulation [15]. The most elegant application of the linear data base has been in the construction of fractal surfaces to model terrain with unprecedented realism [9,7]. Marshall, et al [10] have also used a linear data base to model trees with great detail. Like the point data base, however, the linear data base requires very large numbers of scene elements to represent the non-linear complexity of the real world. The realism of fractal surface images is achieved only by rendering hundreds of thousands of planar faces; a single linear tree requires thousands of faces. The number of scene elements is critical to the efficiency of the image generation approach. The greater the number of scene elements, the greater the number of surfaces and boundaries that must be computed, resulting in greater computation costs for sorting, priority determination, and antialiasing. Limiting the number of scene elements limits the realism of the linear model. For this reason, current flight simulation systems have been criticized for being too cartoonish.

Various approaches to scene simulation using curved surfaces have been developed. Quadric surfaces have been used effectively by the Mathematical Applications Group, Inc. (MAGI) [8] and the New York Institute of Technology (NYIT) [18] to model complex man-made objects. In one of the most significant applications of computer image generation to date, Blinn used quadric surfaces with texture maps to model Jupiter and its moons for the Voyager flyby [2]. Quadrics have also been used to model molecules and are common data base primitives in CAD/CAM. None of these applications, however, has exploited the potential of quadric surfaces for modeling a wide range of natural features.

More complex curved surfaces,in particular bicubic surfaces, have been studied extensively [3,1,4]. Such surfaces provide great modeling flexibility because they allow for continuity of slope between adjoining surfaces. Despite the development of several clever image generation algorithms, however, the mathematical complexity of these surfaces results in severe computation loads for complex scenes.

2. AN EFFICIENT DATA BASE FOR SIMULATING NATURAL SCENES

In selecting a data base for simulating natural scenes, we acknowledged the necessity of compromising perfect realism for computational efficiency. Efficiency is particularly important when long sequences of images are to be generated to produce a dynamic presentation. We therefore set our goal to be to represent the essential realism of the real world as efficiently as possible. Studying natural scenes we noticed that their essential characteristics were surface curvature and textural detail. We also noticed that an observer perceives complex scenes as sets of isolated features, such as hills, trees, rocks, and lakes. Reviewing the candidate data bases in light of these observations, one can see that quadric surfaces hold the potential for an efficient natural scene model. Quadrics provide the simplest form of curved surface. Moreover, a single quadric surface can be used to model an individual feature such as a hill or tree. A geometric data base composed of quadric surfaces will therefore minimize the number of scene elements and allow straightforward computation of boundaries and surface shading.

In selecting a model for texturinig we noted that the textural detail of the real world was statistical in nature. This indicated that it was not necessary to store exact texture maps, which would limit the variety of texture patterns in the model. To provide on-line control for antialiasing, perspective validity, and dynamics, we chose to produce texturing by means of a mathematical function which could be mapped onto scene surfaces in such a way as to modulate surface shading and translucence.

2.1 The Geometric Data Base

A geometric data base composed only of quadric surfaces would not provide the topographical variety required for natural scenes. To allow for adjoining quadric surfaces we include in our geometric data base the option of bounding each quadric with a finite number of planes. We then define the geometric data base to be a set of discrete convex objects, with each object defined by one quadric surface and N bounding planes, where N can be zero. This ensures that all surface boundaries will be at most second-order, allowing for scan-line intercept determination in closed form.

We define a scene coordinate space (X_S, Y_S, Z_S), with the X_S axis pointing east, the Y_S axis pointing north, and the Z_S axis pointing vertically up. We define a ground plane $(Z_S = 0)$, and a unit light vector. For each image of the scene, we define an eyepoint and look angle. The coefficients of the quadric surface and bounding planes for each object are defined in scene coordinates and must be tranasformed to eye coordinates (X,Y,Z), centered at the eyepoint $(0,0,0)$, with the Y axis pointing in the direction the eye is looking (Fig. 1).

For each quadric, we get an equation in eye coordinates of the form

$$Q(X,Y,Z) = Q_1X^2+Q_2Y^2+Q_3Z^2+Q_4XY+Q_5YZ+Q_6XZ+Q_7X+Q_8Y+Q_9Z+Q_0=0 \qquad (1)$$

For each plane, we get an equation of the form

$$P(X,Y,Z) = B_1X+B_2Y+B_3Z+B_4=0 \qquad (2)$$

The surface geometry of a quadric surface bounded by an arbitrary number of planes can be quite complex. Generating an image of such an object requires exact determination of which surface is visible at each pixel, but testing each surface at each pixel is inefficient. We can make use of area coherence and scan line coherence by noting that a given surface will cover many pixels and many scan lines. We can greatly reduce visibility computation by noting that there are far fewer boundary points than there are surface points in a typical image, and that boundary information can be used to determine surface visibility. The key to efficient processing of the geometric data base is, then, to determine which portions of the boundary curves are visible in the image.

We define an image plane with coordinates (x,z) parallel to the XZ plane a distance f in front of the eyepoint such that the y axis pierces the coordinate axes origin. We also define the image x axis to be parallel to the eye coordinate X axis and the image z axis to be parallel to the eye coordinate Z axis (Fig. 1). Then the transformation from eye coordinates to image coordinates can be represented as

$$\begin{aligned} Z &= kx \\ Y &= kf \\ Z &= kz \end{aligned} \qquad (3)$$

Our strategy will now be to use equations (1), (2), and (3) to project all surface boundaries onto the image plane. We will then determine all key points on each boundary, that is points at which boundary visibility, and therefore surface visibility, changes across scan lines. We will then use the key points to determine a scan line list of visible boundaries and surfaces.

The most important image curve is the limb curve, defined as the projection of the quadric silhouette. The limb curve can be derived by substituting equation (3) into equation (1) to obtain a quadratic equation in k,

$$Ak^2+Bk+C = 0 \tag{4}$$

where A is a second-order expression of the image coordinates, x and z, B is a linea expression of x and z, and C is a constant. (Algebraic expansions will be omitted for the sake of brevity.) The parameter k relates to the distance from the eye, varying from a value of 0 at the eyepoint to a value of 1 at the image plane, and increasing along the ray out into the eye coordinate space. In general, a ray will intersect a quadric surface at two points, giving two distinct values of k from equation (4). By definition, the limb curve is the set of image points for which the rays are tangent to the quadric surface. For these points k is single valued, so the discriminant of equation (4) is zero.

$$B^2 - 4AC = 0 \tag{5}$$

This then gives the limb curve as

$$f(x,z) = a_1x^2+a_2z^2+a_3xz+a_4x+a_5z+a_6 \tag{6}$$

where the coefficients are expressions containing only the quadric surface coefficients, and f, the distance from the eye to the image plane.

The remaining image curves will be intersection curves, that is, projections of the curves of intersection between the quadric and its bounding planes. To solve for the coefficients of an intersection curve, we must satisfy equations (1), (2), and (3) simultaneously. Substituting equation (3) into equation (2), we get

$$k = -B_4/(B_1x+B_2f+B_3z) \tag{7}$$

Then substituting for k in equation (3) and using the result in equation (1), we get the intersection curve as

$$g(x,z) = e_1x^2+e_2z^2+e_3xz+e_4x+e_5z+e_6=0 \quad (8)$$

where the coefficients are all expressions containing the quadric surface coefficients, the bounding plane coefficients, and f.

Given a quadric surface with one or more bounding planes, we must determine which boundary curve segments are visible. To do this, we introduce the concept of a visibility line. We define a visibility line for each bounding plane as the projection on the image plane of the line of intersection between the quadric silhouette plane and the bounding plane (figure 2). Since the visibility line will be used to partition a particular curve into visible and invisible segments, its definition must include a sense or sign determined from the sense of the boundng plane. Rewriting equation (2) as an inequality to include the bounding plane sense, we have

$$P(X,Y,Z) = B_1X+B_2Y+B_3Z+B_4>0 \quad (9)$$

Substituting equation (3) into equation (9) gives

$$k(B_1x+B_2f+B_3z] + B_4>0 \quad (10)$$

The equation for the silhouette plane can be determined from equations (4) and (5) to be

$$kB+2C = 0 \quad (11)$$

Since B is a linear expression of x and z, and C is a constant, equations (10) and (11) can be combined to give the visibility line as

$$v_1x+v_2z+v_3>0 \quad (12)$$

where the coefficients are expressions of the quadric surface and bounding plane coefficients and f.

The visibility line defined by equation (12) can be used to determine the visibility of any point on the limb curve relative to a particular bounding plane. If the limb point satisfies equation (12), it is defined to be visible relative to the plane used to define the line. If more than one bounding plane is used in the object definition, a limb point must satisfy the visibility line for each in order to be visible in the image.

The visibility line for each bounding plane can also be used to determine the visibility of any point on the intersection curve related to that plane. This visibility test is only necessary when the plane faces away from the eye,

that is, when the eye is on the same side of the plane as the defined part of the quadric surface (fig. 3a). When the eye is on the other side of the plane, no visibility test is necessary because the whole intersection curve is visible (fig. 3b). To test which side of the plane the eye is on we simply substitute the eyepoint $(X,Y,Z) = (0,0,0)$ into equation (9) to get

$$B_4>0 \tag{13}$$

if equation (13) is satisfied, the eye is on the object side of the plane and a visibility test for the intersection curve is required. Figure 3a shows that in this case, the visible portion of the intersection curve lies on the opposite side of the visibility line as the visible portion of the limb curve. Therefore, for a point on the intersection curve to be visible, it must fail to satisfy equation (12).

Thus, the visibility line can be used to determine portions of the limb curve defined to be visible as well as portions of the intersection curve whose visibility depends on eye position in the scene.

Images of objects with more than one bounding plane may include linear boundaries resulting from the intersection of two planes. We introduce the concept of an intersection line, which we define as the image of the intersection between two bounding planes. The intersection line will define that portion of the intersection curve of one bounding plane defined to be visible relative to the other (figure 4). In this sense, the intersection line is analogous to the visibility line with the first intersection curve replacing the limb curve. With a derivation similar to that used for the visibility line, we get the intersection line for two bounding planes as

$$x_1x+x_2z+x_3>0 \tag{14}$$

where the coefficients are in terms of the coefficients of the two bounding planes.

In addition to its function as a visibility criterion, an intersection curve may also be a visible image boundary. In order for a point on an intersection line to be visible it must satisfy the appropriate form of equation (14) for all other intersection lines produced by a common bounding plane.

Having computed the equations for all image boundary curves and lines, we must now determine which boundary segments are visible. We note that boundary segments are generally visible over many scan lines and that their state of visibility changes only at certain key points. Determining these key points and restricting visibility tests to a few sample scan lines will greatly reduce visibility computations. The key points of boundary visibility are curve extrema (minimum and maximum z), contact points, intersection points, and triplet points.

Curve extrema can be computed from equations (6) and (8) in the standard manner. Since curves don't exist on scan lines outside there extrema, they clearly are potentially visible only on scan lines in between.

We define a contact point as a point of tangency between the limb curve and an intersection curve (figure 5). In general, two contact points exist for each intersection curve and can be determined from the intersection of the relevant visibility line and the limb curve (equations (12) & (6)). Note that certain object geometries and viewing aspects may produce a single contact point or no contact points for a given intersection curve (figure 5). Contact points are points of potential change of visibility state for limb and intersection curves.

We define an intersection point as a point of intersection between an intersection line and an intersection curve related to a common plane (figure 6). Intersection points are points of potential change of visibility for intersection curves and intersection lines.

We define a triplet point as the image point of the corner intersection of three bounding planes. Triplet points can be computed as the intersection of two intersection lines and are points of potential change of visibility for intersection lines (figure 7).

Once we have computed all the potentially visible curves, lines, and key points for the image of an object, we are prepared to perform visibility tests. In determining what tests to make for particular boundaries, the following rules apply. Any image point relating to a point on the quadric surface must be tested against all visibility lines. Any image point relating to a point on a particular bounding plane must be tested against all intersection lines related to that plane (except for the line producing the

point) and, if the plane is not visible, the visibility line related to that plane.

We begin the visibility testing by testing all key points, since only visible key points affect visibility changes in the boundaries. Editing out invisible key points, we compile a list of visible key points sorted on z in scan-line order. We call this a z-band list because it defines regions in the image z direction (vertical) in which boundary visibility remains constant from scan line to scan line.

Within each z-band in the list, an average z value is computed to represent a typical scan line on which all boundary curve and intersection line intercept points are computed. These intercept points are tested for visibility, and codes for all visible boundary intercepts are entered in the z-band, sorted from left to right based on the intercept x value. Boundary curve intercepts are coded to define leftmost or rightmost intercept. Finally the sorted visible boundary intercepts are used to determine the visible surfaces in between. The surface between two boundary intercepts is assumed to be the quadric surface unless both boundaries are related to a common bounding plane.

The resulting object visibility list, consisting of sorted z-bands, each with x- sorted boundary intercept and surface codes, provides a very efficient image "blueprint" for directing scan-line surface shading of the object. As the image is generated, scan line by scan line, a particular object is considered only if the scan line falls within the z-band list. For each such object, the pertinent z-band is referenced, and, proceeding from left to right, boundary codes are referenced, intercepts computed, and surfaces shaded on in between pixels. Figure 8 shows how this approach simplifies the image generation of a complicated object by reducing a maze of potentially visible boundaries to a manageable list of visible boundaries and surfaces. Figure 8a shows the limb and intersection curves, and the intersection and visibility lines in an image of a sphere bounded by 6 planes. Figure 8b shows the z bands indicated by horizontal lines drawn through all visible key points. Note how within each z band the visibility of boundaries and surfaces remains constant. Figure 8c shows the final shaded object with hidden surfaces removed.

In addition to streamlining intraobject visibility, that is, the determination of visible surfaces for a single object, the visibility list simplifies interobject visibility, the determination of priority between different objects. Because our geometric data base includes only convex objects, we need not compute the distance to the visible object surface at every pixel. Instead, we can use the z-band lists to compute the leftmost and rightmost surface distance on a scan line and use linear interpolation in between. Then, when two objects overlap on a scan line we can use the interpolated distances to determine priority within the overlap region.

2.2 The Texture Data Base

We can simulate textural detail efficiently by modulating surface shading intensity in a defined manner. In so doing, we must take care to assure perspective validity by making the texture intensity depend on the scene coordinates of the surface being textured. For any given image we are interested in texturing visible surfaces only. Thus it would be inefficient to produce texture intensity values for all scene surface points. The most practical approach is to produce texture values only for visible scene surface points corresponding to image sample points. Since the image sample points depend on the viewing perspective, we must be able to produce texture values for arbitrary scene points. For this reason we have chosen a mathematical function to produce texture. A mathematical texturing function offers the additional advantage of requiring a minimal data base to produce a wide variety of texture patterns, each of which can cover an unlimited region in the scene. The control inherent in a mathematical function, computed during image generation, also provides for straightforward antialiasing of texture patterns and allows implementation of complex texture motion.

In choosing the exact form of our texturing function, we decided that it would be most efficient to represent real-world detail at a statistical level. A rather obvious way to do this is to use the principle of Fourier expansion, and, indeed, this approach to texturing is not unique [14, 11]. After investigating different expressions of various waveforms, we found a very effective texturing function to be defined as

$$T(X_s,Y_s,Z_s) = \sum_{i=1}^{n} C_i \frac{[\mathrm{Sin}(\omega_i X_s + PX_i) + 1]}{2} \sum_{i=1}^{n} C_i \frac{[\mathrm{Sin}(\omega_i Y_s + PY_i) + 1]}{2} \quad (15)$$

(15)

here PXi and PYi represent phase shift functions to avoid a tartan-like regularity of the pattern. We have found that defining PXi as a sinusoidal function of Y_s and PYi as a sinusoidal function of X_s produces natural-looking patterns for low values of n (Figure 9).

The primary use of the texturing function is to simulate surface detail by modulating shading intensity. This is done by computing a weighted average of the surface shading intensity and the texture function value at each visible point. A texture weighting parameter is defined for each object to provide flexibility in scene modeling. A secondary use of the texture function is to simulate boundary irregularity and amorphousness of certain natural features, such as trees and clouds. We accomplish this by treating locally dark texture regions on an object's suface as though they were holes in the object. This effect is achieved quite simply by assigning a threshold value for the texture function and defining an object to be translucent at any image point where the texture function falls below the threshold. The artificial boundaries produced between the visible and invisible portions of the texture surface can be softened by varying the translucence linearly as the texture function crosses the threshold. This technique is demonstrated in Figure 10, which shows a sky plane textured with variable translucence to simulate a cloud layer.

The texturing function greatly enhances the realism of objects defined by the geometric data base. Figure 11a shows mountains modeled by the geometric data base only, and Figure 11b shows the same scene inhanced by the texturing function. In addition to adding simulated topographical detail, the texturing blends surface shading across boundaries between abutting objects. This unifying effect is due to the fact that all objects, as well as the ground plane, are textured with the same texture function parameters so that the texturing function maps the same pattern continuously across all scene surfaces as a function of scene coordinates.

The combination of the geometric and texture data bases is particularly effective in simulating amorphous objects, such as trees and clouds, whose boundaries are both complex and subtle (figure 12). The trick of simulating such features so efficiently is to control the translucence at an object's silhouette. This capability is provided as much by the geometric data base as by the texturing function, for it is the definition of the limb curve

(Equation 6) that allows us to vary the threshold of the texturing function to increase translucence in a straightforward manner at image points near the object boundary. Because the translucence can be increased smoothly and continuously, the image will have soft boundaries which will require no antialiasing.

The control inherent in the mathematical texturing function has two more advantages over less flexible texturing techniques, such as stored texture maps. Antialiasing of texture patterns can be achieved simply by testing sine wave frequencies and dropping those that exceed the image sampling frequency (projected into scene space). In addition, any of the texturing function image parameters can be varied from frame to frame, allowing the simulation of a wide range of dynamic effects. Thus it would require little additional frame computation to simulate trees blowing, smoke rising, clouds drifting, or rivers flowing.

The texture data base required to implement this simulation capability is extremely modest. A realistic texture pattern can be defined by 25 parameters, including sine coefficients and frequencies, phase shifts, and translucence thresholds. A given texture pattern can be used for any number of objects covering any region in the scene. Thus, all trees of a particular type could be simulated using one pattern, all rivers using another pattern, etc. We have found that complex and varied natural scenes can be simulated effectively using only 10 texture patterns. This compactness of the texture data base simplifies both scene modeling and image generation.

2.3 Data Base Construction for Complex Scenes

The new data base simplifies the modeling of complex scenes because it conveniently partitions the model into two levels of topographical detail. The geometric data base can be used to model major topographical features such as hills explicitly, and the texture data base can be used to represent secondary topographical variations statistically. The compactness of each of these data bases permits a straightforward specification of the parameters of size, shape, position, and frequency content, which are the essential characteristics of natural scene features.

To facilitate the modeling of complex natural scenes, we developed procedural algorithms to generate clusters of scene features. Only two types

of quadric surfaces are required to produce a wide variety of scene features. Hyperboloids of two sheets are very effective in simulating hills and mountains, and ellipsoids are efficient for modeling trees, rocks, and clouds. For each feature cluster, we define a region on the ground plane over which the cluster, will be generated. We also define a typical spacing between features in the cluster as well as size, shape, and color parameters for a typical template object. The algorithm places the template object within the defined region at positions determined from the defined spacing. As the object modeling each feature is generated, the algorithm perturbs its position, size, shape, and color parameters randomly to produce natural statistical variations within the cluster. Adjacent objects are tested for intersection, and bounding planes are computed for abutting objects. This permits the modeling of topographic structures, such as rolling terrain, mountain ranges, and forests, which are too complex to be simulated by isolated objects. The algorithm also allows us to define features in a cluster as "terrain objects" upon which other scene objects will lie. Terrain objects are generated first so that the objects composing subsequent clusters can be raised to the appropriate terrain elevation after they are positioned on the ground plane. The simplicity of quadric surface shape and position definition makes this process quite straightforward. Similarly, clusters can be defined to be positioned at a fixed altitude above the ground plane, a capability useful in modeling clouds.

To use the mathematical texturing function to model secondary topographical variations we must define sets of function parameters, with each set chosen to simulate a desired texture pattern. The individual parameters in each set can be determined from an analysis of the spatial frequency content of features being modeled. In general, natural features have a power spectrum whose amplitude decreases as frequency increases. We have found that natural-looking texture patterns can be generated using from 3 to 7 sine waves whose frequencies increase by a factor of approximately 2 and whose amplitudes decrease by a factor of approximately one half the square root of 2. The complete texture data base is defined by a list of parameter sets. As clusters of scene features are generated, each object is assigned a specified texture parameter set number. A particular texture pattern can be assigned to any number of clusters, minimizing the size of the overall data base.

Using the clustering algorithm, we can quickly generate and change models of complex natural scenes. Figure 13 shows an example of a scene model including clusters of hills, mountains, trees and clouds. Figure 14 shows a variation of the model altered by a small number of parameter changes in the data base.

3. CONCLUSIONS

Textured quadric surfaces provide a means of bridging the gap between computationally cheap, but cartoonish, scene simulation and highly realistic, but costly, scene simulation. Textured quadric surfaces produce a compact, functional data base related directly to the most significant topographical characteristics of scene features. This approach minimizes image generation computation because it minimizes the number of scene elements that must be processed. Textured quadric surfaces allow us to represent the essential realism of natural scenes as an impressionist painter would, thus avoiding the costly replication of unimportant details. The new scene model is particularly effective for modeling amorphous objects, such as trees and clouds, which continue to be stumbling blocks for other approaches. The new model is, however, comprehensive because it can model man-made as well as natural features. The inclusion of bounding planes even permits modeling linear features, such as buildings.

Scene simulation using textured quadric surfaces has application in many diverse fields, including art, entertainment, advertising, scientific simulation, and training. At Grumman we have used it in a public relations film to show an advanced concept, forward-swept-wing aircraft in flight before it was built (figure 15). We have also applied the technology to pattern recognition research in target tracking by a missile [12]. We are currently investigating real-time implementation of the algorithms for flight simulators.

4. ACKNOWLEDGEMENTS

Significant contributions to this work were made by Tom Jacquish, Ed Berlin, Jr., Bob Gelman, and Mike Gershowitz. I am indebted to my wife Joan for pointing out that hyperboloids make better mountains than paraboloids.

The research described in this paper was sponsored in part by the Air Force Human Resources Laboratory (AFHRL) under contract F33615-79-C-0029.

5. REFERENCES

1. Blinn, J., Computer Display of Curved Surfaces, PhD Thesis, Computer Science Department, U. of Utah, Dec 1978.

2. Blinn, J., Voyager 2. SIGGRAPH Video Review, Issue 1, May 1980.

3. Catmull, E., A Subdivision Algorithm for Computer Display of Curved Surfaces. UTEC-CSc-74-133, PhD Thesis, Computer Science Department, U. of Utah, Dec 1974.

4. Clark, J. Designing Surfaces in 3-D. Comm. ACM 19,8 (Aug 1976), pp 454-460.

5. Csuri, C., Hackathorn, R., Parent, W., Carlson, W., and Howard, M., Towards an Interactive High Visual Complexity Animation System. Computer Graphics 13,2 (Aug 1979), pp 289-299.

6. Dungan, W., Jr., A Terrain and Cloud Computer Image Generation Model. Computer Graphics 13,2 (Aug 1979), pp 143-147.

7. Fournier, A., Fussell, D., and Carpenter, L., Computer Rendering of Stochastic Models. Comm. ACM 25,6 (June 1982), pp371-384.

8. Goldstein, R. A., and Nagel, R., 3-D Visual Simulation. Simulation 16,1 (Jan 1971), pp 25-31.

9. Mandelbrot, B. B., Fractals: Form, Chance and Dimension. Freeman, San Francisco,1977.

10. Marshall, R., Wilson, R., and Carlson, W., Procedure Models for Generating Three-Dimensional Terrain. Computer Graphics 14,3 (July 1980), pp 154-159.

11. Max, N., Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sunset. Computer Graphics 15,3 (Aug 1981) pp 317-324.

12. Mendelsohn, J. and Leib, K. G., Optical Correlation Module Design Study. Air Force Systems Command Contract AF 33615-82-C-1727, Final Report, Sept 1983.

13. Reeves, W. T., Particle Systems- A Technique for Modeling a Class of Fuzzy Objects. Computer Graphics 17,3 (July 1983), pp 359-376.

14. Schacter, B. J., Long-Crested Wave Models. Computer Graphics and Image Processing 12 (1980), pp 187-201.

15. Schacter, B. J., Computer Image Generation. Wiley - Interscience, New York, 1983.

16. Spooner, A. M., Breglia, D. R., and Patz, B. W., Realscan - A CIG System With Increased Image Detail. Proc. 2nd Interservice/Industry Training Equipment Conf., Salt Lake City, Utah, Nov 1980, pp 110-116.

17. Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., A Characterization of Ten Hidden Surface Algorithms. ACM Computing Surveys 6,1 (May 1974), pp 1-55.

18. Williams, L., Casting Curved Shadows on Curved Surfaces. Computer Graphics 12,3 (Aug 1978), pp 270-274.

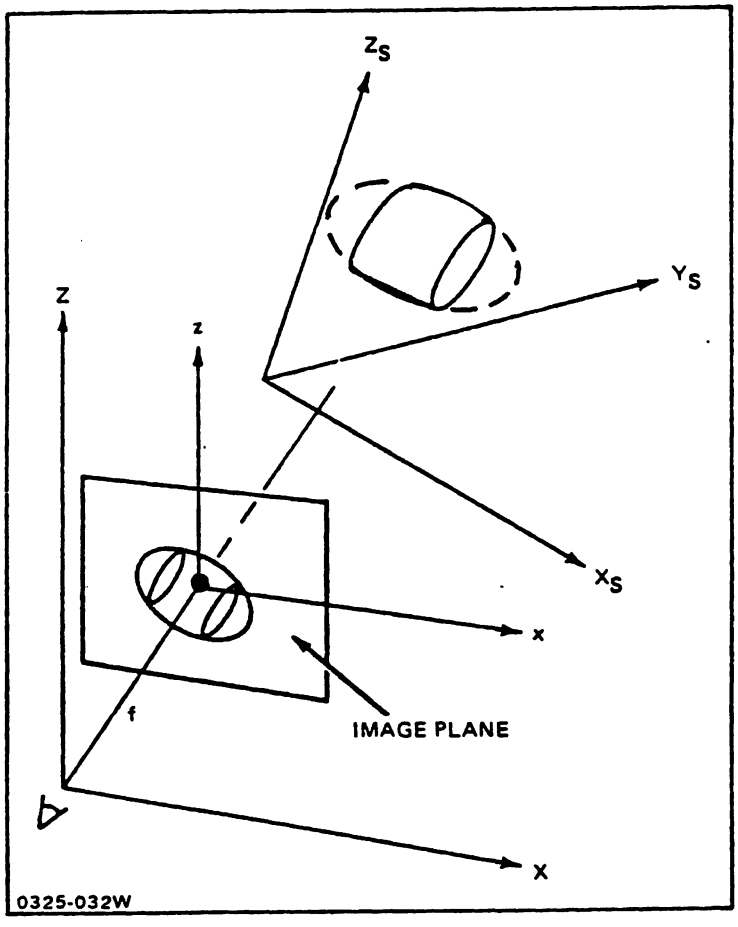


Figure 1. Transformation from scene to eye coordinates and projection onto image plane.

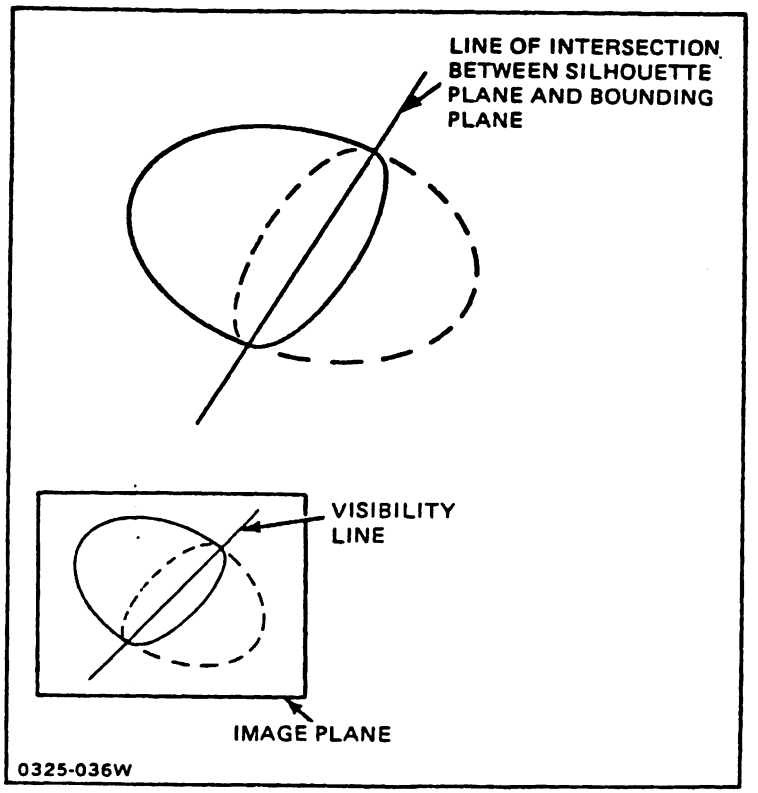


Figure 2. Visibility line defined as the image of the line of intersection between the silhouette plane and the bounding plane.

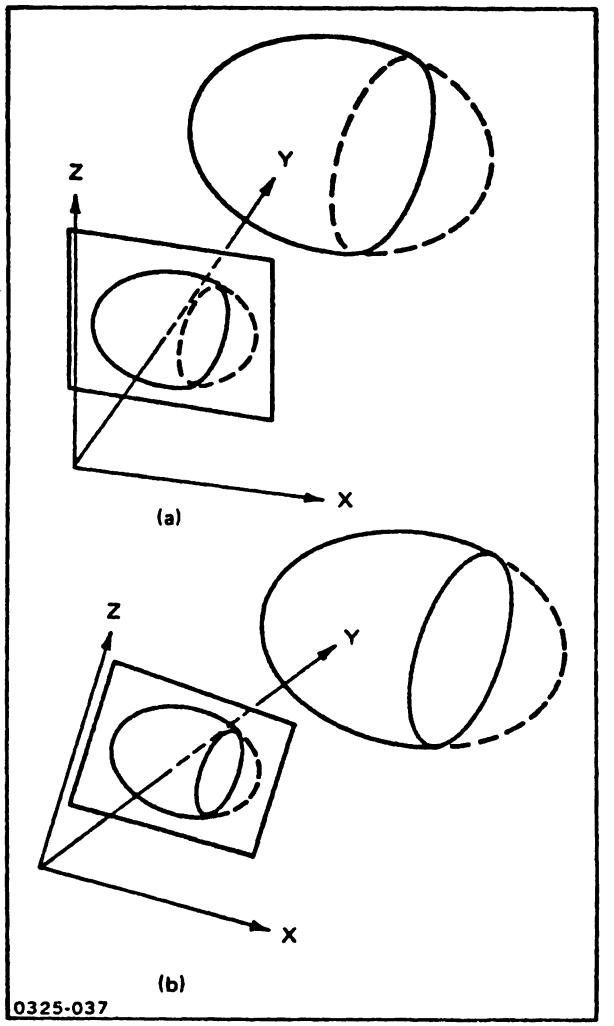


Figure 3. Intersection curve visibility.

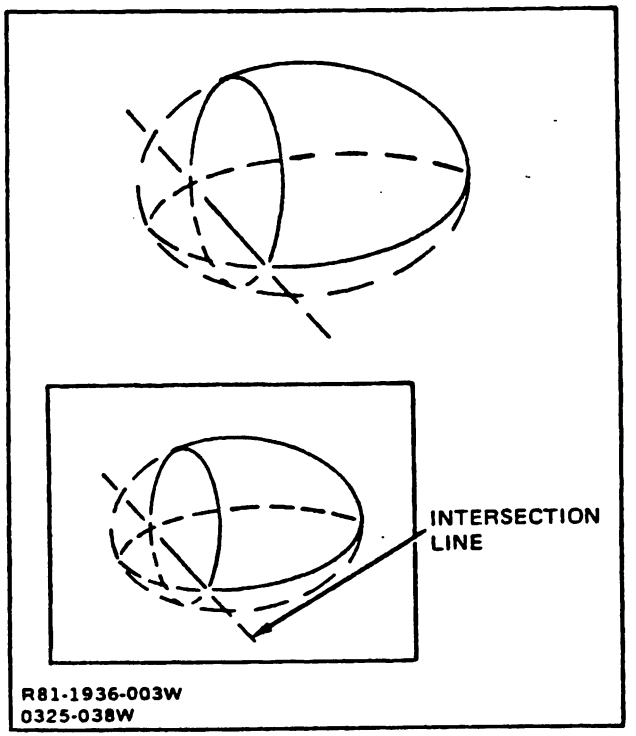


Figure 4. Intersection line defines intersection curve visibility relative to intersecting bounding plane.

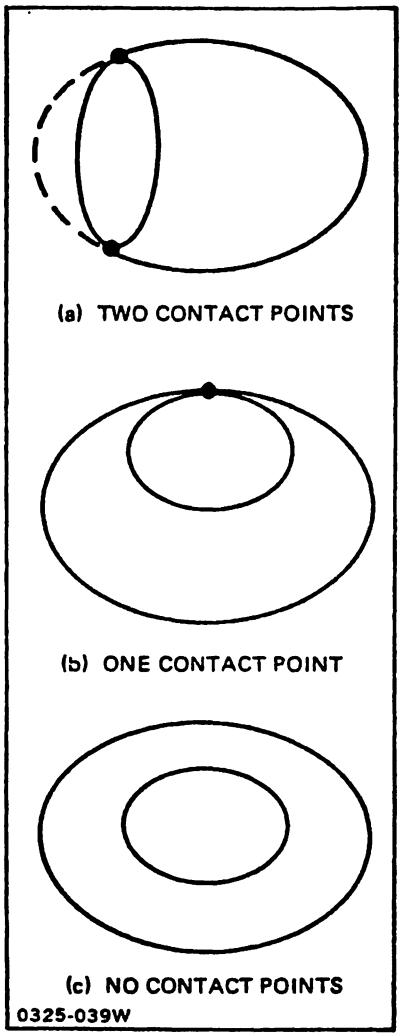


Figure 5. Contact points between limb curve and intersection curve.
(a) Two contact points.
(b) One contact point.
(c) No contact points.

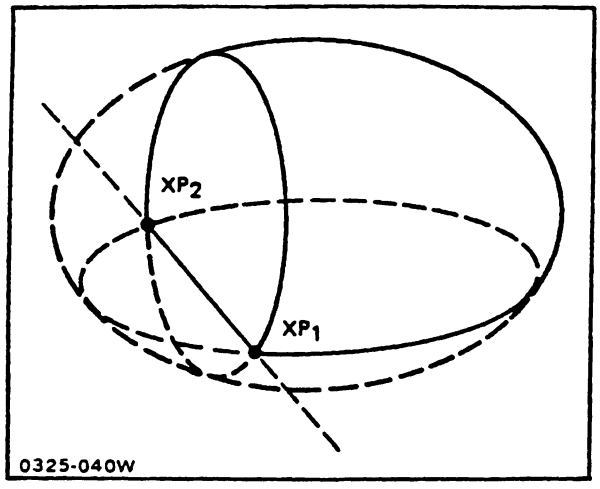


Figure 6. Intersection points.

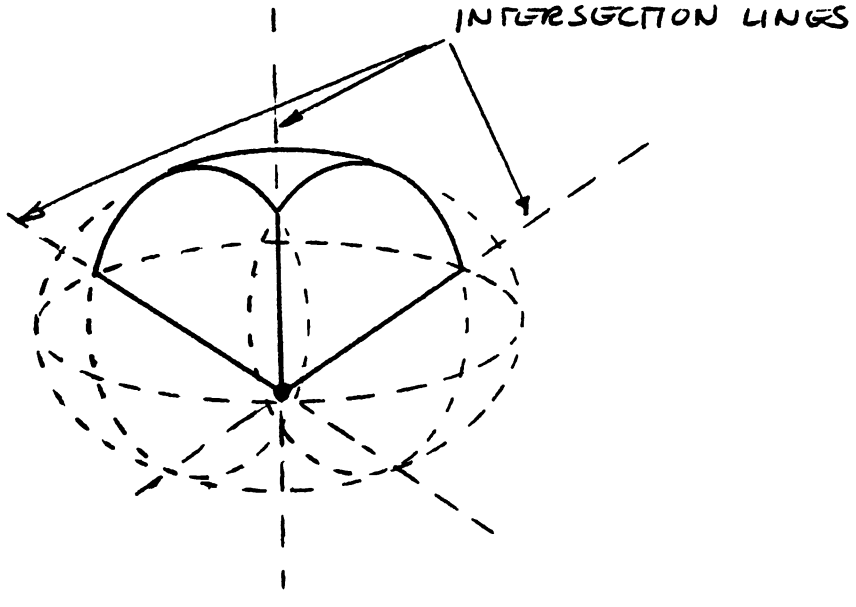
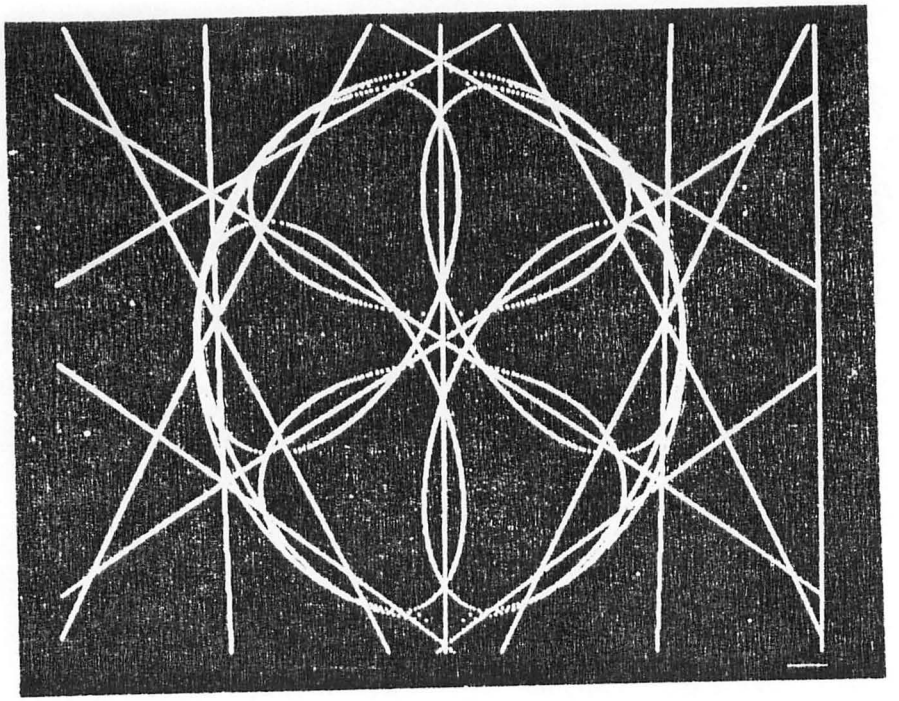
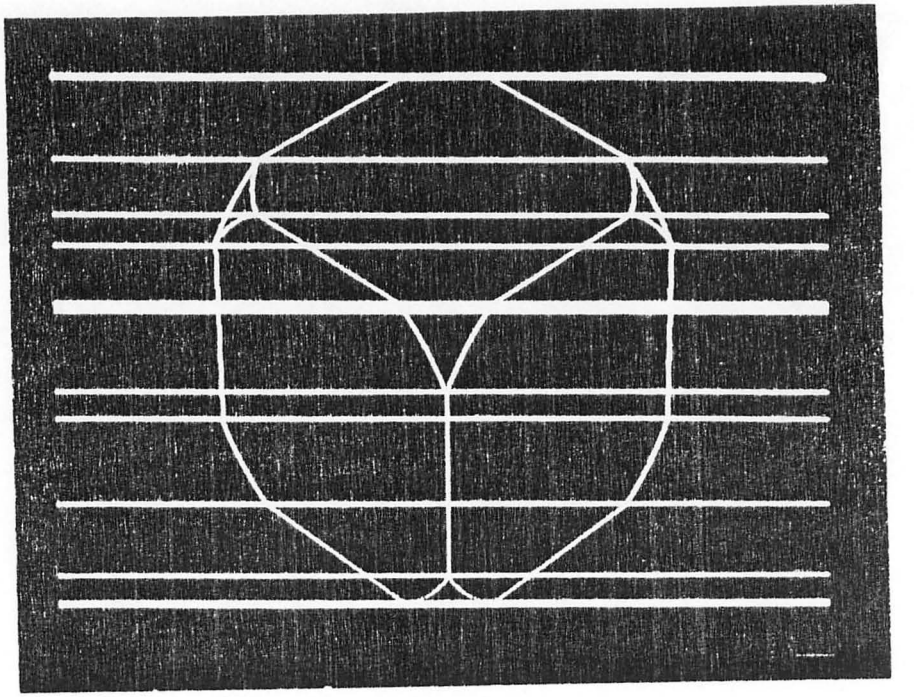


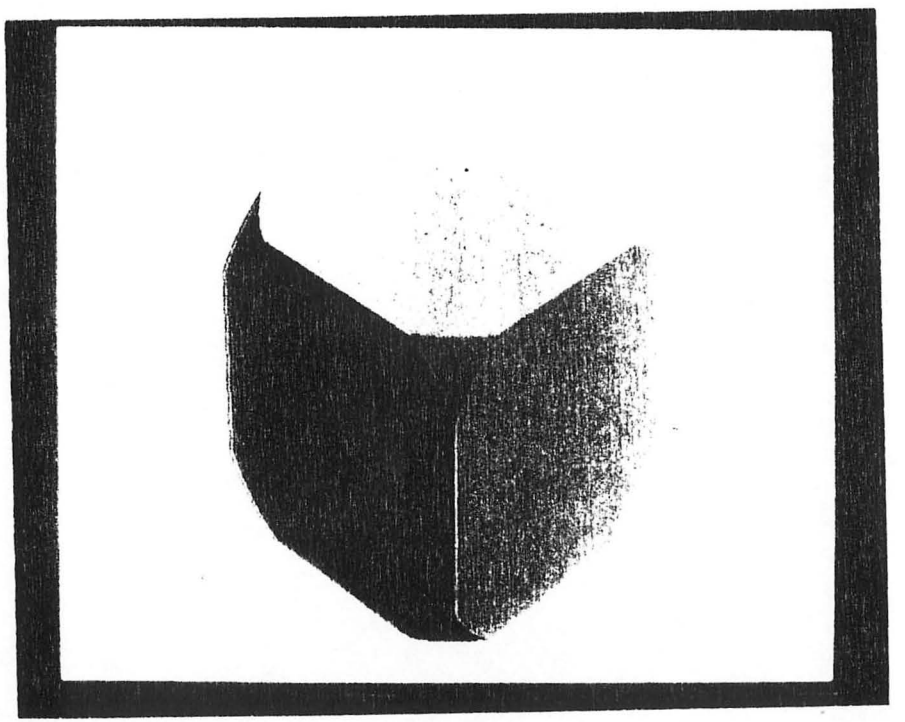
Figure 7. Triplet point computed from intersection of 2 intersection lines (any 2 of 3 shown).



(a)



(b)



(c)

Figure 8. Visibility Determination for Sphere with 6 Bounding Planes

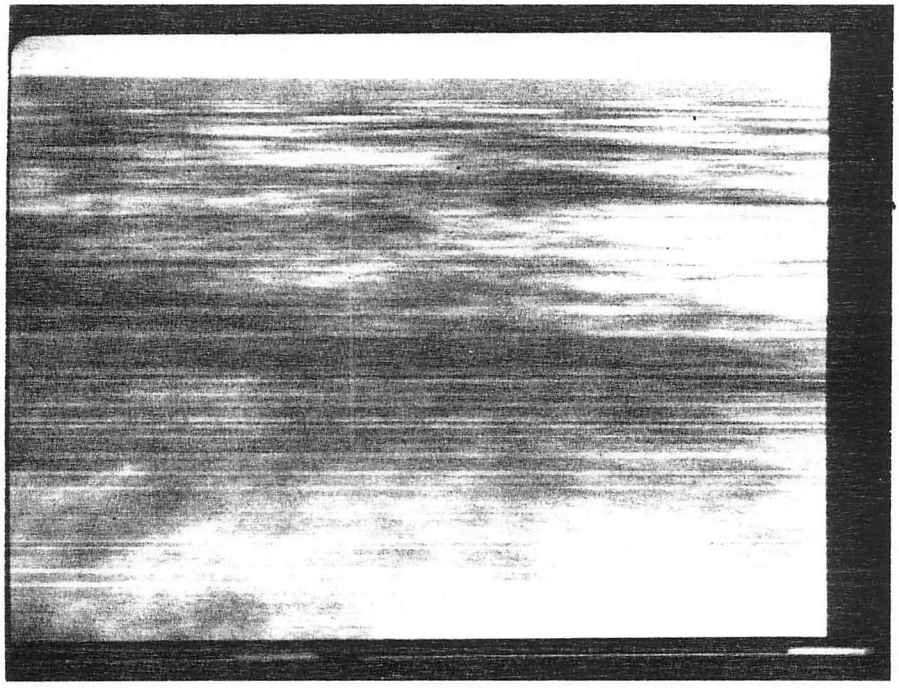


Figure 9. Texture Pattern on Ground Plane from Texture Function (Eqn. 15) with $n = 7$.

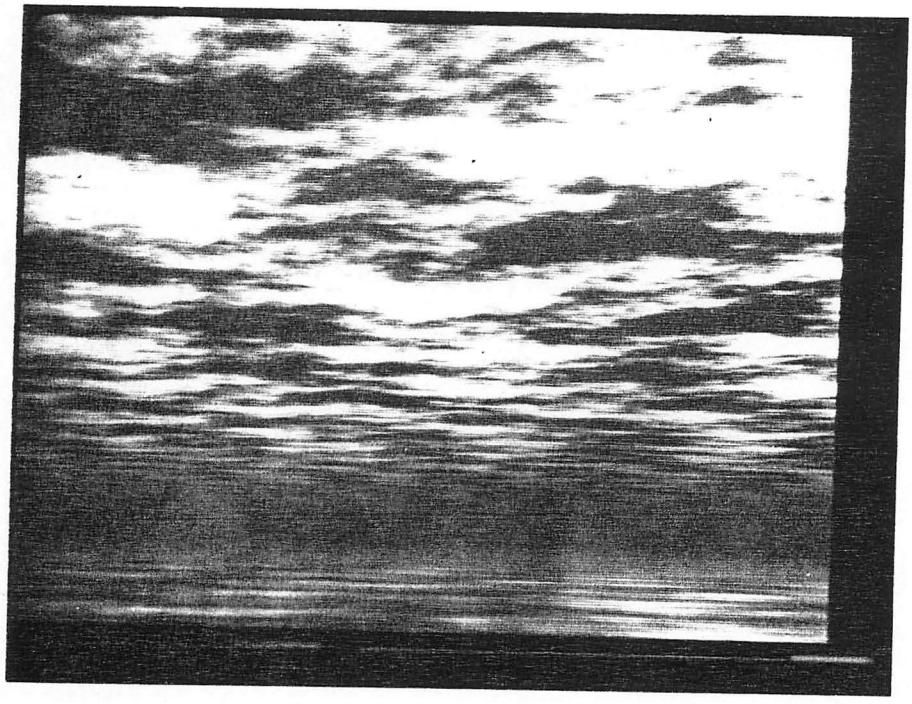
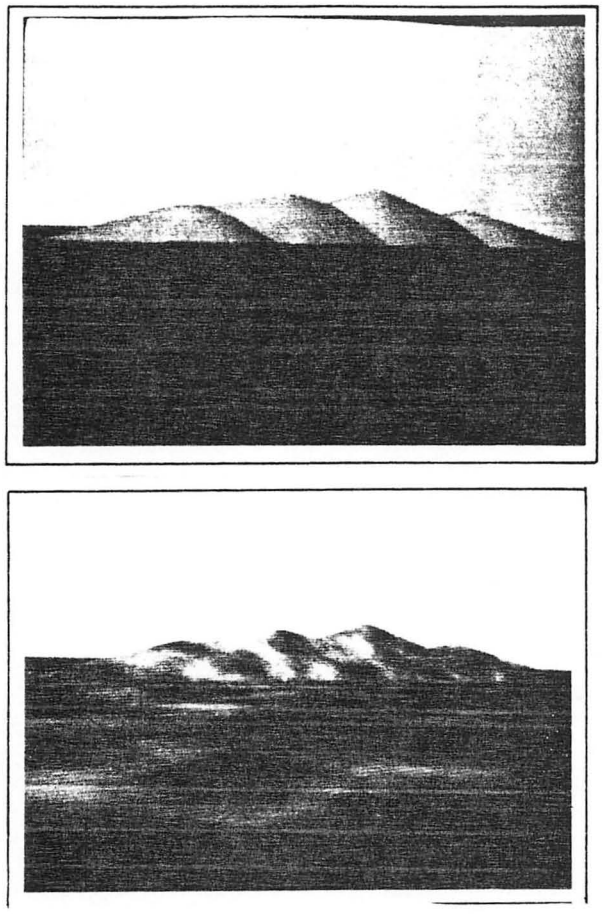


Figure 10. Cloud Layer Simulated By Texture Function modulating Shading and Translucence



(a)

(b)

Figure 11. Mountains modeled by (a) Quadric Surfaces
(b) Quadric surfaces with Texturing

Figure 12. Trees and Clouds

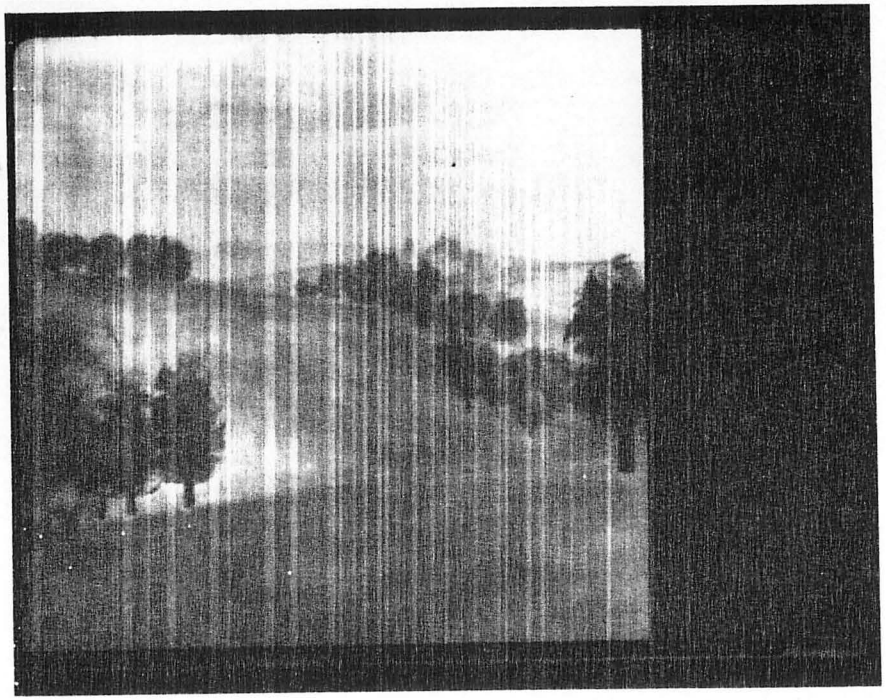


Figure 13

400 60

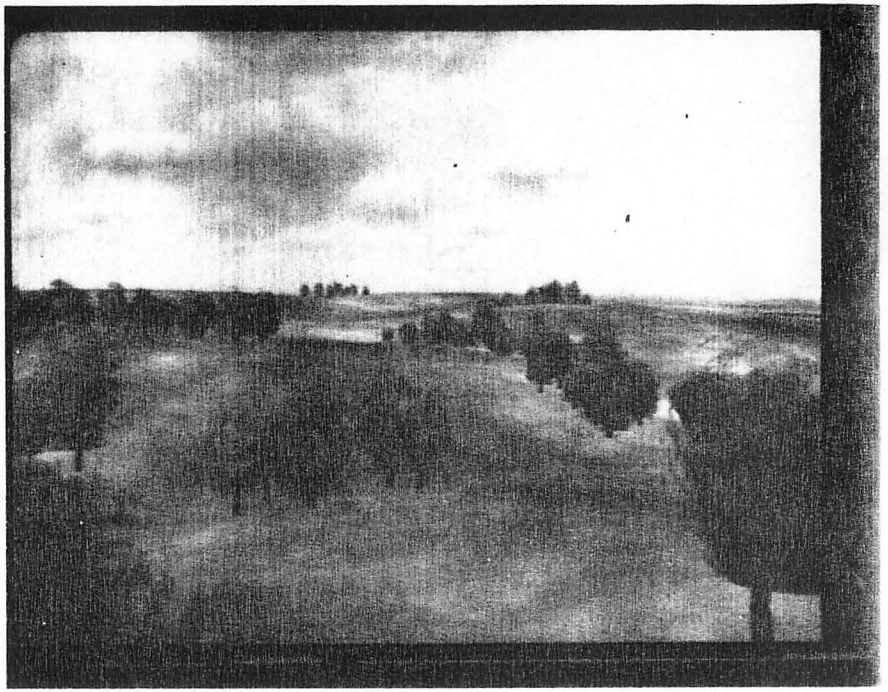
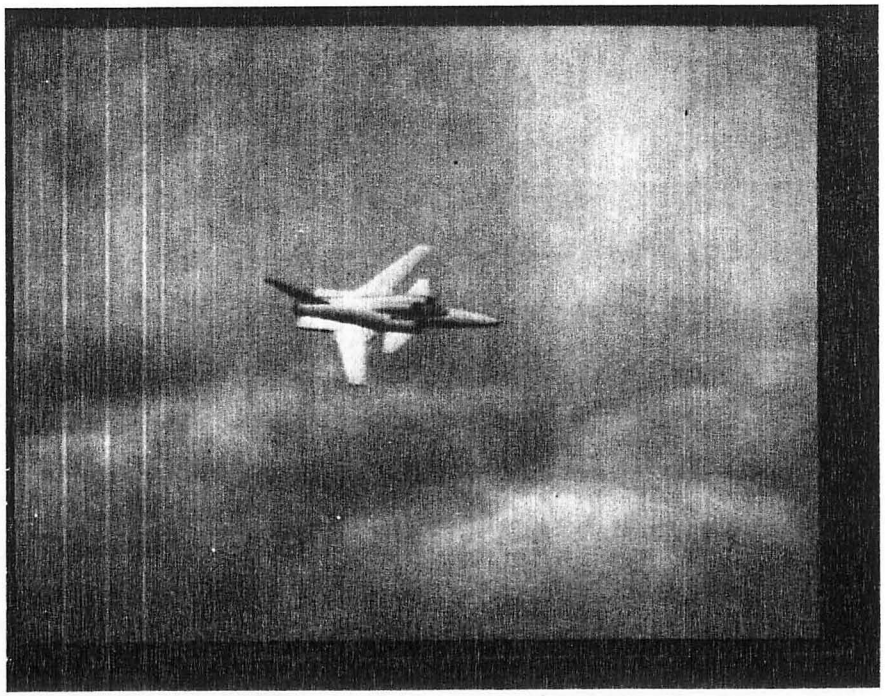


FIGURE 14



300
95