GENERATING SOFT SHADOWS EFFICIENTLY

Gordon Fossum and Donald Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-22 June 1987

of any geometry, and of any luminous intensity distribution [16].

2 Previous Methods

As pointed out in [12], several techniques have been used to produce shadows, only some of which produce soft shadows. These methods fall into just a few categories.

Shadow Volumes Shadow volumes have been used extensively to determine whether a point is in shadow [6]. The idea is to take the point light source and the object that will create the shadow, and calculate, in world-space coordinates, the polygons that bound the shadow thus created. If these polygons are then rendered into a modified depth buffer, the point visible at each pixel in the buffer can be painted as shadowed or not depending on how its depth compares with the shadow depths for that pixel. This method can be extended to model a non-point light source by taking several sample positions on the surface of the light source. Then each such point source sample can create its own set of shadow volumes, the number of light source samples visible to the pixel can be computed, and the pixel can be illuminated appropriately. If the light source is to be modeled with, say, 80 points, then each polygon in the scene will generate 80 shadows, and each such shadow must be rendered into the depth buffer [1]. If curved surfaces are to be represented, they must be first transformed into polygonal facets. Since this results in a very large number of surfaces, each of which generates at least three shadow polygons, the technique becomes extremely expensive for modelling scenes with curved surfaces illuminated by extended light sources, though it has been done [20]. Proper organization of the data can be used to ameliorate the computation time [12].

Shadow Polygons Another method computes the portions of polygons which are hidden from the light source using a hidden-surface computation in object space from the point of view of the light source [6] [17] and represents the results as polygons which can be painted as surface detail on the original polygons in the scene when the scene is rendered. This method does not appear to be well-suited to extension for soft shadows and has not been so extended.

Depth Buffer Techniques A third technique first scan converts the scene from the light source point of view using a depth buffer algorithm and then scan converts again from the observer's viewpoint [19]. If the visible surface at a given pixel transformed into the light source coordinate system is behind the visible surface at that same point in the light source depth buffer, the point is in shadow and is attenuated, otherwise not. This technique has not previously been viewed as suited to generating soft shadows, but our new method can be seen as such an extension since its basic philosophy is similar even though the algorithms and data structures are significantly different. Such algorithms are well-suited to exploiting area coherence properties of objects in the scene and to using scan-line techniques to perform many computations iteratively and thus to operate at reasonable speed.

Ray Tracing Ray tracing produces a variety of illumination effects from specular reflection of light among objects to refraction to shadows [18]. Simple ray tracing only traces a single ray through each pixel and therefore cannot sample an area of a non-point light source to produce soft shadows. If however a bundle of rays is shot from a given point in a scene to various locations on a non-point light source, the fraction of those rays which arrive at the light source without intersecting other objects in the scene can serve as the desired attenuation value for producing soft

Generating Soft Shadows Efficiently

Gordon Fossum   Donald Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712

Abstract

Historically, generating images with soft shadows has been computationally expensive. This paper describes a new method which uses two techniques: light buffers to organize and speed up the computation of soft shadows for selected pixels in an image; and an adaptive interpolation algorithm to determine upon which pixels the computation will be performed. For these pixels, the exact occlusion is computed by determining the regions of the non-point light source that can be seen from the pixel. Linear interpolation of these occlusion percentages is used elsewhere.

Using this algorithm, soft shadows can be accurately computed with less overall cost than with previous methods. The method models light passing from a light source which can be modelled as an illumination function of its surface through a window which can represent a transmission function over its surface and into the scene. Furthermore, the light intensity can easily be made a function of the location in the scene, independent of the shadowing computations. As a result, the method is suitable for generating a number of realistic lighting effects without undue overhead.

1 Introduction

Many methods for generating realistic shaded images achieve computational efficiency through the assumption that all objects are illuminated only by point light sources and a constant ambient light. Unfortunately, most real-world light sources (other than such things as stars and carbon arc lamps) cannot be accurately modelled as point sources, which means that illumination models of this type produce a

number of rendering defects. One noteworthy example of such a defect is the generation of unrealistic shadows with sharp boundaries. The world around us contains sharp shadows only very rarely. What usually happens is that there is a penumbra at the edge of a shadow where the illumination by the light source attenuates gradually from full intensity to full shadow. In order to generate realistic shadows it is necessary to determine what fraction of each light source illuminates each visible point in a scene.

In recent years a number of illumination models and associated rendering algorithms have been developed which remedy the defects caused by point light sources [13] [1] [5] [8] [4] [11] [14] [15]. These methods can produce strikingly realistic images, but they entail extreme computational expense. Here we describe an efficient rendering technique for scenes containing both polygonal and curved surfaces illuminated by non-point light sources. The method is based partially on the use of a modified form of the light buffer [10], in which the idea is to project the scene from the point of view of the light source, and build a sorted data structure in light source coordinates which is then used to speed up shadow computations. Once the objects of interest are determined, they are projected in a novel way so as to simplify the calculation of the percentage of light source occlusion, which in turn is used to calculate the intensity of light reaching the point from this partial light source.

The second technique employed by our method is a form of adaptive interpolation, in which the level of occlusion of a given pixel from a given light source is determined through values of nearby pixels, when possible. This results in a further significant improvement in the performance of the algorithm.

In addition to allowing the computation of soft shadows with significantly greater efficiency than previous methods, our algorithm is designed to model a light source

shadows [5]. Typically, the "target points" on the light source are selected randomly, rather than as a grid of points, in an effort to avoid an effect similar to mach-banding in the penumbra.

Each ray from a point in the scene to a selected point on this non-point light source must be checked against any polygons or patches with which it might intersect. A variety of bounding box techniques exist which can reduce the required computations, but even so, checking hundreds of rays against thousands of objects for each visible pixel in the image is a heavy computational burden. It is possible to exploit ray-to-ray coherence [21], with some increase in efficiency, but the results are still extremely slow. Of course this technique is good for producing quite a number of illumination effects besides soft shadows.

Radiosity Radiosity is a method for determining the energy exchange between the elements of any closed environment based on their bi-directional reflectance functions [8] [4] [11]. The solution to the radiosity equation describes the activity at the center of each polygon, and (typically) Gouraud shading [9]is used to interpolate between these point samples. For shadows to look realistic, the polygons in the penumbra should be subdivided to fine enough detail so that such interpolation methods will give the desired result. Unfortunately, finely subdivided polygons result in a massive increase in the computation time, as radiosity is an inherently n^2 process.

This technique has been extended to process non-Lambertian objects, but as such is probably the most computationally expensive rendering technique ever developed.

Although images produced using radiosity methods can be impressive in their realism and like ray traced images contain many illumination effects besides soft

shadows, the computational expense of rendering even moderately complex scenes is normally prohibitive.

3 Light Buffers

In September, 1986, Haines and Greenberg of Cornell University first introduced the concept of a "light buffer" [10], in which objects are projected from the point of view of a (point) light source, thus organizing the scene in x and y, and sorting it in z. This, in effect, is a 2D bucket sort, which generates a significant improvement in the performance of algorithms which need to sort data in space. The method is very similar to that used by Randolph Franklin in 1980 [7] to accomplish hidden surface removal. Franklin referred to his creation as a "variable grid data structure". The light buffer was used by Haines and Greenberg to speed up the shadow computations from a point light source, in a ray-tracing environment. This paper represents a third application of this technique. Here, we organize the scene from the point of view of the center of a non-point light source, and use the geometric information thus gained to both limit the number of objects that need to be considered, and to simplify the calculations required to determine how much shadowing a given object does.

Assume that a scene is composed of opaque curved surface patches or polygons which are then decomposed into triangles, and that all objects that can cause shadows are within the viewing frustums of the eye and of each light source (note that our algorithm will extend to allow light sources within the scene by using six buffers per light source in the style of radiosity hemicubes [10], but the efficiency of the algorithm will suffer somewhat). The idea is to scan this scene from the point of view of the eye as well as from the point of view of the "center" of each light source. We will further assume that the light source can be modeled as planar. Of course,

since real world light sources are often three-dimensional, the planarity assumption can cause some theoretical error in the illumination, but this is negligible if the light source is somewhat removed from the scene, and will not degrade the appearance of the image even for a light source which is within the scene.

Note that the basic algorithm is not limited to triangles, but the present implementation uses surface patches subdivided until they meet a planarity criterion [22]. At that point, we are faced with a (probably not quite planar) quadrilateral, which we merely draw a diagonal across, creating two triangles. The algorithm would change in a straightforward manner to handle (possibly non-convex) polygons, as the only change would involve a somewhat more complex scan-conversion routine in the final step of the process.

Note further that enough parametric information is retained within the algorithm to allow computation of the exact normals at the vertices of each triangle, for the purpose of Phong shading [2] the surfaces.

3.1 Light Coordinate Systems

Let a point somewhere near the center of the scene define a *light vector* from the light source center to the scene center and assume that the *light source plane* is the plane containing the light source center which is orthogonal to this light vector.

A light source can be represented geometrically as a planar surface bounded by curves of any desired degree. It can also be represented in a discrete format as a set of point light sources. There are a number of ways to decide what point light sources to select. One solution is a grid of points chosen with fine enough granularity to avoid aliasing effects. A second solution is to choose a set of sample points randomly [5].

Such a point set should be selected to accurately represent the size and shape of the non-point light source since, for example, a long narrow light source should generate very large penumbrae along gradients parallel with the long axis of the light source and rather narrow, sharp penumbrae along gradients perpendicular with that axis. Further, points should be chosen so that as you move along the gradient of any penumbra the set of points which are visible will change as smoothly as possible. We have chosen the second solution, in which the light source is represented as a square bitmap with a 1 in each position which is within the light source area and a 0 elsewhere. This will always accurately represent the surface area of the light source and will not produce aliasing effects if enough points are used. Each bit represents a point on the light source plane, and we will refer to these points as *luxels*.

It would be easy to represent a light source with varying intensities across its surface by allocating, say, a full byte of information to each location in the "bitmap", so that the representation of a light source with around 144 points would occupy 36 full words of memory. Our implementation presently assumes a homogeneous source, though more complex sources are not noticeably more time-consuming.

An illumination frustum in light source space is constructed by first selecting a projection plane which is parallel to the light source plane and includes the point in the scene closest to the light source. Next a minimum size rectangular window centered on the light vector is selected to ensure that all objects in the scene are included. (Again, this would generalize to multiple windows, one for each face of a cube, if the light source is within or very close to the scene). The coordinates of a point in the scene in a *light coordinate system* are x, y, and d, where x and y are the coordinates of the point's perspective projection onto this window and d is the distance from the plane of the light source to the point in *world coordinates*. A light coordinate system is shown in Figure 1.

3.2 Scene Organization in Light Coordinates

A light buffer, in our usage, is a representation of the screen in light coordinates consisting of a two-dimensional array of linked lists. We will avoid the obvious acronym ("libel") and call such a list element an *lb-list*. Each lb-list is associated with a square (or rectangular) region of the light source's projection plane (its *lb-region*), and contains pointers to descriptions of the objects which project at least partially onto its lb-region. This structure is an analog to pixels in a frame buffer, where the "region" can be viewed as an address in the buffer, and the "list" is the collection of bits located at that address. There are two key differences between frame buffers and light buffers, however. The first is, of course, that light buffers are defined from the point of view of a light source rather than an observer. The second, less obvious difference is that the resolution of the light source window (i.e. the size of an lb-region) is variable, and can be selected to optimize the computations as a function of the distribution of objects in the scene *vis à vis* the light source, and has *no effect* on the final appearance of the image, because the light buffer in no way discretizes the scene. Each lb-list is ordered by increasing distance from the light source. The distance of an object from the light source as used in this ordering is the smallest depth in light coordinates of any point on the object. Each node in this list contains a pointer to the object and this distance. Note that the light coordinates are in floating point, and when x and y are truncated to integer, the result is the address of the appropriate lb-region/lb-list.

It is important to note that the depth that is stored is not the projected depth that results from a perspective transformation, but represents world-space depth from the light source plane to the object. As will be shown below, this format for describing which objects in the scene appear in each lb-list makes it easy to compute

light source occlusions by projecting objects from the point of view of any point in the scene.

In order to make use of light buffers, and to successfully interpolate between points where the occlusion levels are exactly calculated, some auxiliary data structures are required. These are

- an *object list* containing nodes describing the triangles in the scene, including the coordinates of each vertex in world-space and in each light source space along with information about normals for intensity calculations, and
- a traditional *depth buffer* from the observer's point of view, which contains the projected depth of the nearest intersection of a ray through the center of a pixel with an object in the scene, a pointer to this visible triangle, and the values of the parameters u and v at that point on the original surface if the triangle was obtained from a parametric patch.
- a linked list of records to describe where, along each scanline, one patch stops, and another begins. Furthermore, within each patch, the boundaries of fully lit, partially lit, and fully shadowed regions are stored.

4 Shading Using Adaptive Light Buffers

Shading of the scene using light buffers is done in two passes. In the first pass, a light buffer for each light source along with the depth buffer and object list are built. The second step is the actual shading calculation. The algorithm paints the picture in a series of strips, each several pixels wide. Refer to Figure 2. The steps taken for each strip are as follows: scan the first and last lines of the strip (in the

figure, these are lines A and B), computing shadow occlusions for all light sources for some subset of the pixels on the scan line, being sure to note (through the use of a binary search within the scan line) those locations where the scan line moves from one patch to another, and where the light intensity changes state. In the figure, the numbers on each line represent the order in which the points are investigated.

There are three states: umbra ($occlusion = 1.0$), penumbra ($0.0 < occlusion < 1.0$) and fully lit ($occlusion = 0.0$). Whenever an exact occlusion calculation is done, save the result. The occlusion calculations are detailed below. If the order in which these state changes occur in the two scan lines is not identical, perform a binary search on the scan lines between these two, to find the location of the discrepancy (in the figure, these are scan lines C, D and E). Next, linearly interpolate between all of the occlusion values computed so far, to generate values for every pixel for every light source within the strip. (Note here that, since the method is presently used on bi-cubic patches, that the shadow curves that result are cubic, and that a cubic interpolation could be used which would give exact results, but at the scale at which these calculations would occur, we have found linear interpolation to be quite satisfactory).

Set these interpolated results aside for the moment. For each pixel visible to the observer on the screen, the values for u and v stored in the depth buffer for that pixel are used to calculate the world space location of the visible surface and its tangents along u and v. These are used to calculate unit vectors to the eye and to each light source, the normal vector to the surface at that point, and the (x, y, d) coordinates of the point for each light source.

Now the contribution of each light source to the intensity of the pixel proceeds as follows. For a given light source, the intensity of the light illuminating the pixel

is a product of two factors. First there is an attenuation factor which is a function of the location of the pixel being painted (this location already being known, this factor is effectively free). This factor can be used to model directional light sources, as well as the effects of increasing distance from the light source. Second comes the occlusion factor, which is the data which we previously computed and set aside. Having computed the intensity as the attenuation factor times (1 - the occlusion factor), our algorithm multiplies it by the standard cosine and cosine-raised-to-a-power terms to yield diffuse and specular components, and proceeds to paint the pixel.

4.1 Computation of Light Source Occlusion

Occlusion is the fraction of a light source's full intensity, and is necessarily a function of the visibility of each luxel as seen from the point being illuminated. For light sources of constant intensity, this fraction can be easily determined as the number of visible luxels divided by the number of luxels on the light source. If the light source is not homogeneous, the computation is more involved, but the base problem remains to determine luxel visibility in an efficient manner.

This is accomplished in two steps. In the first step, an attempt is made to limit the number of world-space objects that will be considered as possible shadowers. In the second step, each of these objects is analyzed geometrically to determine whether it contributes to the occlusion of the light source.

The triangles which might occlude each light source are selected by examining each lb-list in the light buffer in a neighborhood of the light-space (x, y) position of the point being shaded. A point's neighborhood is that collection of lb-regions in a light source's screen within which objects might occlude the light source as seen

from the point. Any object which is totally outside of the neighborhood can be ignored when calculating shadows for the point from that light source. The size of the neighborhood is computed from the size and shape of the light source and the distances of the point being shaded to the light source plane and to the projection plane in light space. For instance, if the light source is roughly circular, then the radius of the light source, r, the viewing angle, θ, the number of luxels across one scanline of the light source's screen, p, and the ratio between the point's distance from the minimum z and its distance to the light source make it possible to calculate an upper bound of $rp(d_l - d_h)/(d_l d_h tan(\theta/2))$ on the radius of the neighborhood. This situation is illustrated in Figure 3.

It is in the limitation of the number of triangles that must be considered in the occlusion computation that light buffers play their key role in the rendering process. The organization of the data in the light buffer makes it possible to quickly determine the set of triangles which have a reasonable probability of occluding at least part of the light source for any point in the scene. It is easy to see that light buffers are most effective for scenes where the ratio of the size of the light source to its distance from the scene is small, since in such cases the neighborhood of a point covers a small portion of the overall area of the window, and it is possible to select a correspondingly small percentage of the objects in the scene as candidates for occlusion when apprpriate using light buffers.

For each lb-list in the neighborhood, each triangle it references is examined in turn to see if it occludes any of the light source. This process continues until all the lb-lists in the neighborhood are visited or the light source is completely occluded. Note that each lb-list is traversed only to the depth of the point being rendered. To compute occlusion, it is necessary to project each triangle in the neighborhood onto the plane of the light source from the point of view of the point being shaded, since

the area on the light source surface that is occluded by a triangle is a function of the locations of the triangle and the point being rendered. (Occlusion can be determined analytically with our model if the light source is represented geometrically.) In the implementation described below, we use a discrete light source representation with the goal of speeding up the computation. While this method has the potential to produce aliasing effects, we have not yet found them to be a problem, as long as a reasonable number of luxels is chosen.

Once the triangles are projected, they are scanned onto the light source screen, and each point in the triangle is checked to see if it covers part of the light source. This is done by constructing a working light source bitmap with each location within the unoccluded light source set to 1 and other locations set to 0. Once the endpoints of a scan segment of the projected triangle are computed, a precomputed boolean vector can be selected and bitwise ANDed into the scanline. For small light sources this often allows the comparisons of all the positions on a scanline of the light source against the projected triangle to be done in a single operation. When the entire neighborhood has been processed, the unobscured luxels are counted and the result is used to determine an occlusion factor as the fraction of the uncovered light source luxels to the total number of luxels on the light source. (Again, for non-homogeneous light sources, the computation would be more complicated, but the method by which it could be done should be obvious). The full intensity of the light determined previously is multiplied by this attenuation factor, and the resulting corrected intensity is used to determine the specular and diffuse contributions to the intensity of the pixel being painted for that light source using a traditional illumination model such as Phong's [2].

In order to calculate the amount of the planar light source which is shadowed by a triangle T as seen from a point p in light space, it is necessary to know the

distances from T and p to the light source plane in *world-space coordinates.* Each vertex of a triangle is stored as seen from the center of the light source, as (x, y) on the light source's screen, and as a world-space distance from the plane of the light source to the vertex. As illustrated in Figure 4, the vertex can be projected onto the plane of the light source as seen from p as follows

$$x_l = \frac{cf(x_t - x_p)d_t d_p}{d_p - d_t}$$

$$y_l = \frac{cf(y_t - y_p)d_t d_p}{d_p - d_t}$$

Here, d_p is the depth of the point p from the plane of the light source, d_t is the depth of the vertex under consideration, x_t is the x location of the vertex in the light source's screen in *floating point* and x_p is the x location of p on the screen. The conversion factor, cf, is a scale factor, calculated once, to accommodate the different coordinate systems in use (one system has units of the size of an lb-region on the light source's screen, and the other has units of the size of a luxel on the plane of the light source). These are independent of each other, but their ratio is a constant for each light source. The computation of a projected vertex costs 3 subtractions, one division, and 4 multiplications.

In order to perform these projections, light buffer data structures mix screen-space (x, y) values with world-space depths. In this mixed coordinate system, care must be taken in interpolating values during the scan conversion of the triangle onto the light source plane. If two points at different (x, y) locations with different depths are connected by a straight line in this space, the depth changes linearly with linear parametric movement along the line, but the x and y values change non-linearly. This can be seen most easily be noticing that the midpoint of the projection of a line segment ($t' = 0.5$) is not the projection of the midpoint of the line segment itself ($t = 0.5$) as is shown in Figure 5. As a result, if interpolations are to be done,

a reparametrization of the line is required.

Since the length of the projection of a line segment is proportional to the distance of the projection plane from the center of projection, if we assume unit width at depth d_1, the plane containing the closest endpoint of the line segment, the width at d_2, the depth of the plane containing the farthest endpoint of the line segment (that is, the length of the base of the whole triangle of Figure 5) is d_2/d_1. To calculate t' as a function of t, pick any point p along the line. It has some value of t between 0 and 1. A ray from the eye through this point forms a pair of similar triangles, as shown with the dotted lines in Figure 5. Now, t' is just the distance along the d_1 projection from d_1 to the projection of p and the length of the d_2 projection from the projection of p to d_2 is $(1-t')(d_2/d_1)$. Since the height of the first triangle is t, and the height of the second is $1-t$, the similarity of the triangles gives us

$$t'/((1-t')d_2/d_1) = t/(1-t)$$

$$t'(1-t) = (d_2/d_1)t(1-t')$$

$$d_1t' - d_1t't = d_2t - d_2t't$$

$$t'(d_1 + (d_2 - d_1)t) = d_2t$$

$$t' = d_2t/(d_1 + (d_2 - d_1)t).$$

Another complication: it is possible that, given a shadowing triangle, one or two of its vertices will be at a depth greater than the point whose shadow we are computing. In this event, it is necessary to compute the two points where the shadowing triangle intersects the plane normal to the lookat vector which contains the point being shadowed. Then these two intersection points can be compared to the point being shadowed to yield two direction vectors which can be hung off of the one or two valid transformed triangle vertices, yielding the correct infinite planar

[4] Cohen, Michael and Donald Greenberg, "A Radiosity Solution for Complex Environments," *Computer Graphics*, **19**(3), July 1985.

[5] Cook, Robert, Thomas Porter and Loren Carpenter, "Distributed Ray Tracing," *Computer Graphics*, **18**(3), July 1984.

[6] Crow, Franklin, "Shadow Algorithms for Computer Graphics," *Computer Graphics*, **11**(3), July 1978.

[7] Franklin, W. R., "A Linear Time Exact Hidden-Surface Algorithm," *Computer Graphics*, **13**(3), July 1980.

[8] Goral, Cindy, Kenneth Torrance, Donald Greenberg and Bennett Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics*, **18**(3), July 1984.

[9] Gouraud, Henri, "Continuous Shading of Curved Surfaces,"*IEEE Transactions on Computers,* **20**(6), June 1971.

[10] Haines, Eric and Donald Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," *IEEE Computer Graphics and Applications*, **6**(9), September 1986, pp. 6–16.

[11] Immel, David, Michael Cohen and Donald Greenberg, "A Radiosity Method for Non-Diffuse Environments," *Computer Graphics*, **20**(3), August 1986, pp. 133–142.

[12] Max, Nelson, "Atmospheric Illumination and Shadows," *Computer Graphics*, **20**(3), August 1986, pp. 117–124.

[13] Nishita, Tomoyuki and Eihachiro Nakamae, "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources," IEEE Compsac

techniques such as distributed ray tracing or radiosity. The use of adaptive interpolation has been found to greatly enhance the speed of this technique, by minimizing the amount of work done in non-complex areas. It should be noted that both aspects of this technique are applicable to ray-tracing with soft shadows. Note also that a conscious decision to trade off memory for computation time was made. Thus the light buffers approach is quite memory intensive, but since memory is a much more readily available resource than computation speed today, this type of tradeoff is quite a reasonable one to make, as 1 megabit memory chips are widely available now, and 4 and 16 megabit chips will be available from a variety of sources in the near future.

7 Acknowledgements

Many thanks are due to A. T. Campbell, who freely offered constructive criticism and suggestions.

References

[1] Brotman, Lynne Shapiro and Norman I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics and Applications*, October 1984.

[2] Bui Tuong, Phong, *Illumination for Computer Generated Images*, PhD Dissertation, University of Utah, July 1973.

[3] Carpenter, L., "The A-buffer, An Antialiased Hidden Surface Method," *Computer Graphics*, 18(3), July 1984.

region to which the triangle maps as seen from the point being shadowed. Figure 6 shows an example of such a situation. This requires a few more operations than a simple projection of a vertex, but it does not happen often.

5 Implementation and Results

Our light buffer rendering implementation serves as the back end of an object design program which allows users to design and manipulate scenes composed of cubic B-spline patches and then render them either in wireframe or with shading. All software in the system is written in "c", and runs on a Silicon Graphics IRIS 2500 workstation with a 68020 CPU, 2 megabytes of main memory, and a Weitek floating point accelerator. This workstation is rated in the range of 1.5 to 2.0 times the speed of a VAX 780 on floating point and non-floating point computations. In wireframe mode, interactive display update speeds can be achieved to allow effective interactive patch manipulation. Stereo pairs are also used to enhance depth cueing with stereopsis. Figure 7 contains a stereoscopic wireframe examples of a simple scene designed with this system. The reader is invited to try out the stereopsis by viewing the figure with red/blue glasses. The scene is composed of three patches which subdivided comprise 1200 triangles and cover about 25% of the pixels on a 768×768 screen. Figure 8 shows the subdivided triangles. Notice that in areas of high curvature the patches are subdivided more finely as is expected.

The remaining figures were rendered using our algorithm. Each light source is a circular configuration of 100 intensities embedded in a 16×16 array. While the resolution of the light sources was not determined in any theoretically justifiable way, the results are certainly acceptably free of aliasing defects. Figure 9 shows the rendered scene with two point-light sources (actually, sources with radii approximately

one screen pixel in size). Figure 10 shows the light sources changed to have radii of 0.5 and 1.5 units (at a distance of 17.0 from the center of the scene). (The largest patch in the scene is about one unit across.) Figure 11 shows the same scene with light sources of radii 1.5 and 4.5. Figure 12 is an adaptively interpolated version of figure 10, using a sampling density of 8x8. Note that the timing results reported below are for versions of the figures which were not anti-aliased for publication. The figures were photographed directly from the graphics screen, while it was displaying images which were oversampled by a factor of nine (3x3 grid).

Figure 13 shows the timing results for several executions of the program. There are three variables under consideration. These are: size of light source, sampling density, and light buffer granularity. The figure shows the point light sources on the left, the medium light sources (0.5 and 1.5) in the middle, and the large light sources (1.5 and 4.5) on the right.

Note that performance improves greatly with the use of either of our tools in isolation, but when used together, the results are most gratifying: what took about 830 minutes with no assistance from our method was reduced to 90 minutes with an 8x8 sampling density (and no light buffer assistance), to 55 minutes with a 64x64 light buffer (and full saturation sampling density), and to 11.7 minutes when both 64x64 buffer and 8x8 sampling density are included.

6 Conclusions

Our results demonstrate that soft shadowed illumination effects can be generated efficiently using light buffers and adaptive interpolation. The goal of the research was to develop a method that permits more realistic renderings than those achieved using point light source techniques without the overhead of existing more advanced

proceedings, 1983, pp. 237–241.

[14] Nishita, Tomoyuki, I. Okamura and Eihachiro Nakamae, "Shading Models for Point and Linear Sources," *ACM Transactions on Graphics*, **4**(2), 1985, pp. 124–146.

[15] Nishita, Tomoyuki and Eihachiro Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *Computer Graphics*, **19**(3), August 1985, pp. 23–30.

[16] Verbeck, Channing and Donald Greenberg, "A Comprehensive Light-Source Description for Computer Graphics" *IEEE Computer Graphics and Applications*, **4**(7), July 1984, pp. 66–75.

[17] Weiler, Kevin, Peter Atherton and Donald Greenberg, "Polygon Shadow Generation," *Computer Graphics*, **11**(3), July 1978, pp. 275–281.

[18] Whitted, Turner, "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, **23**(6), June 1980, pp. 343–349.

[19] Williams, Lance, "Casting Curved Shadows on Curved Surfaces," *Computer Graphics*, **12**(3), August 1978.

[20] Bergeron, Philippe, "General Version of Crow's Shadow Volumes," *IEEE Computer Graphics and Applications*, **6**(9), September 1986, pp. 17–28.

[21] Speer, L. Richard, Tony DeRose, Brian Barsky, "A Theoretical and Empirical Analysis of Coherent Ray-Tracing," *Graphics Interface '85*, May 1985.

[22] Lane, Jeff and Loren Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing* **11**, 1979, pp. 290-297.

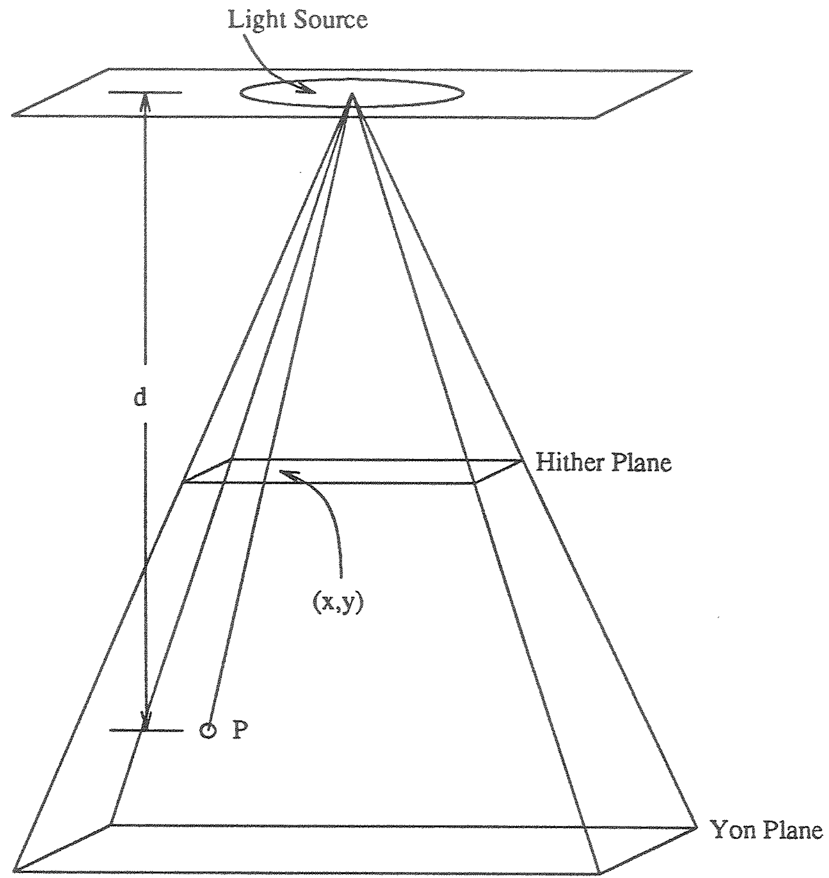

Figure 1: A Light Coordinate System

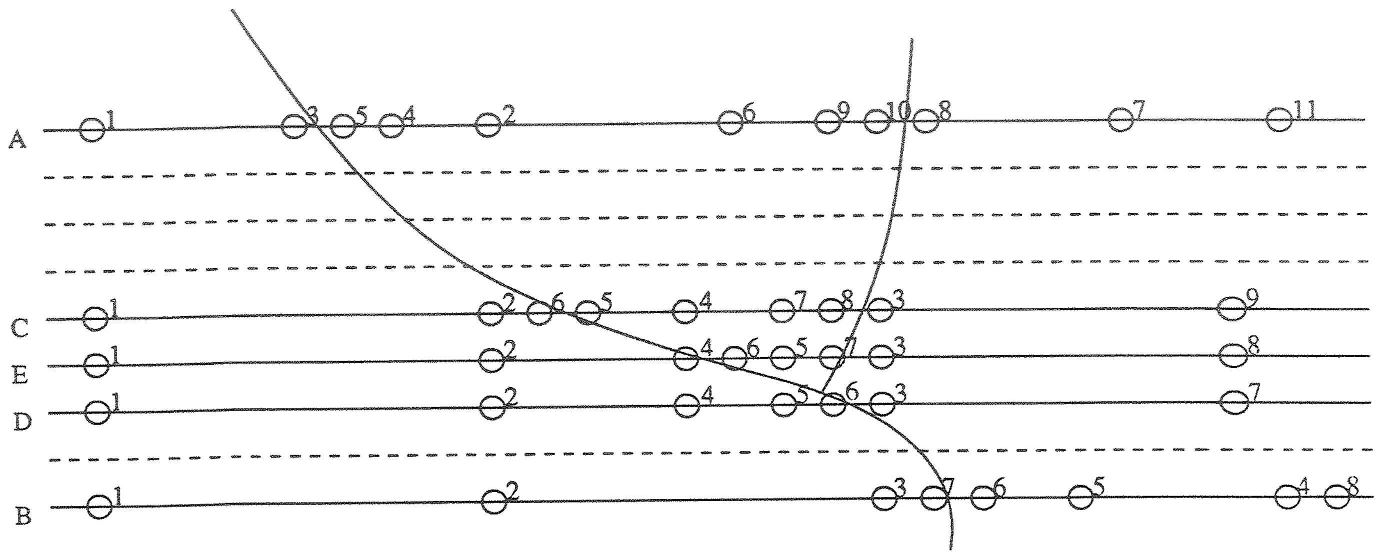

Figure 2: Adaptively Selecting Data Points

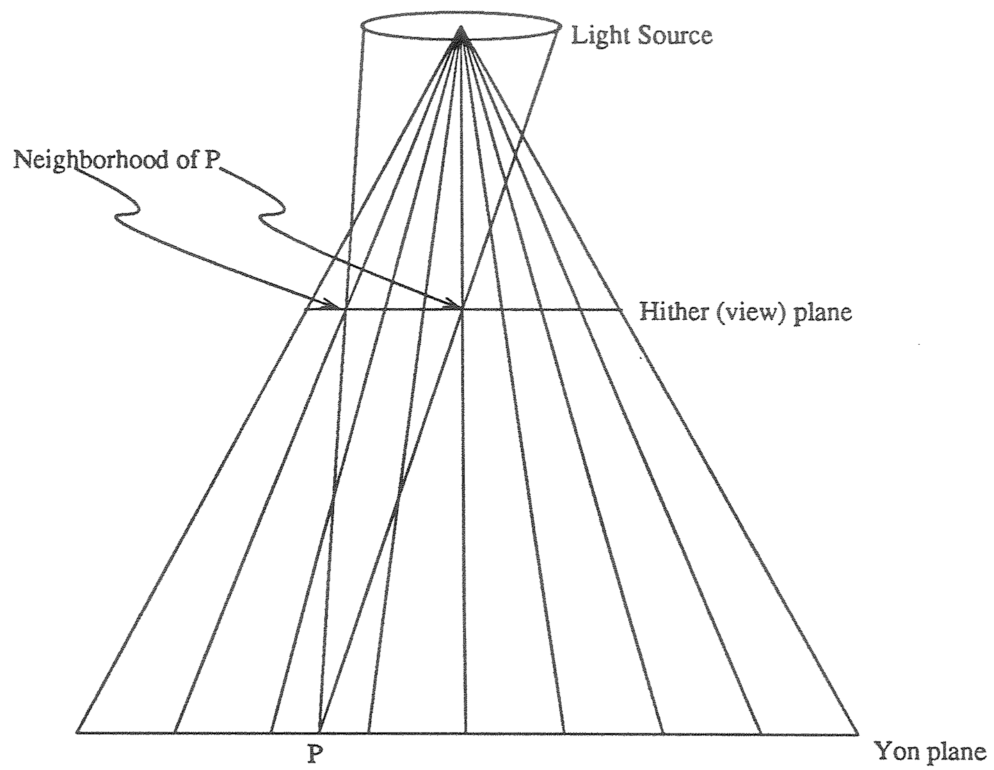

Figure 3: Neighborhood of a Point

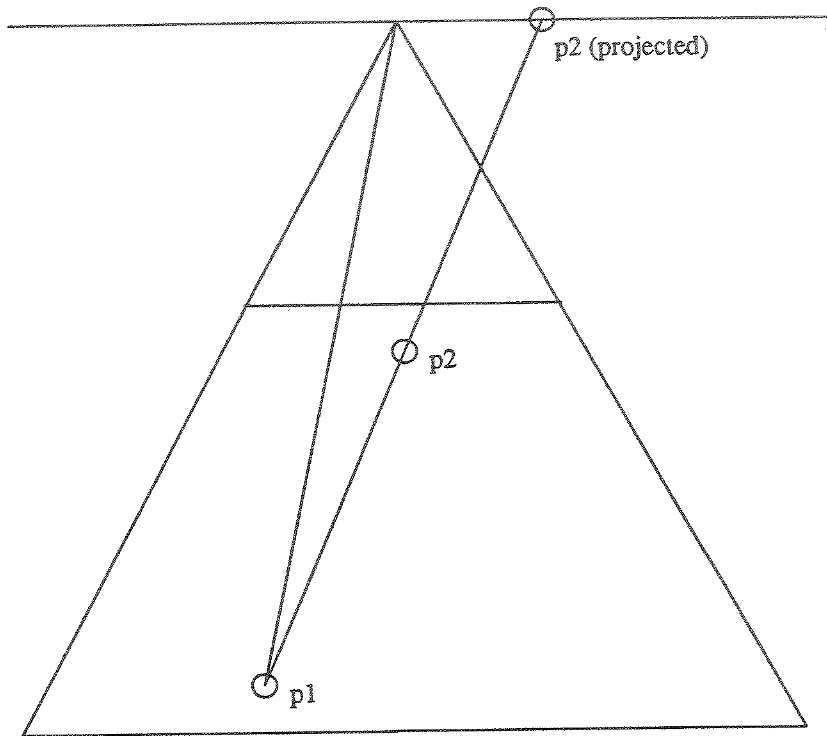

Figure 4: Projecting p2 from the point of view of p1

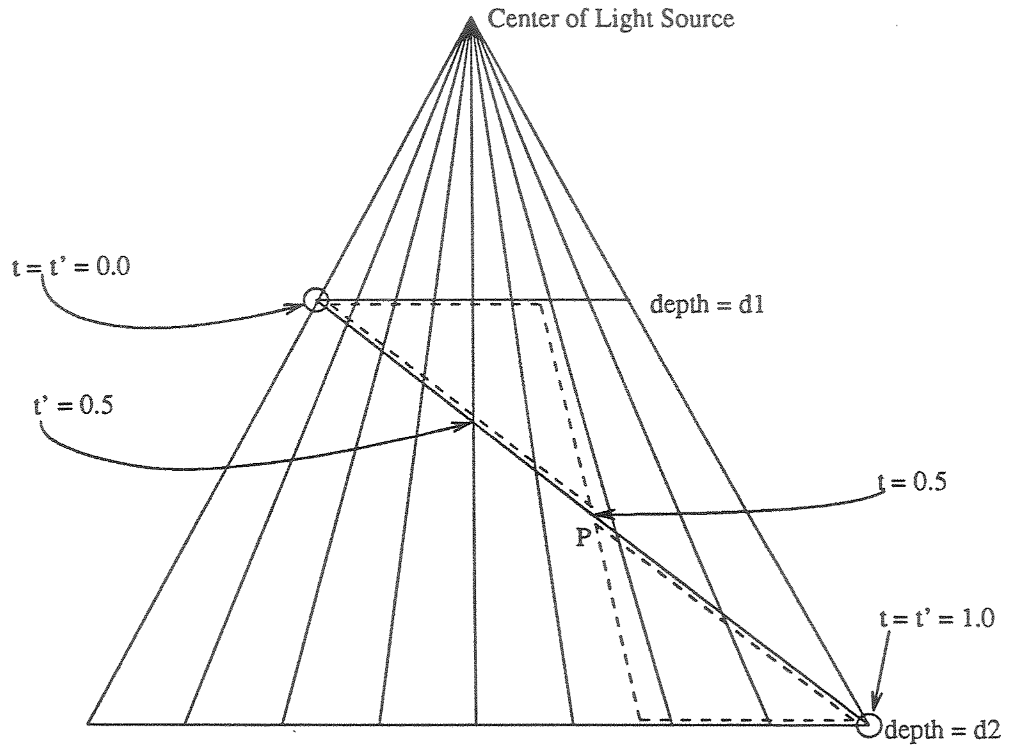

Figure 5: Parametric Non-Linearity

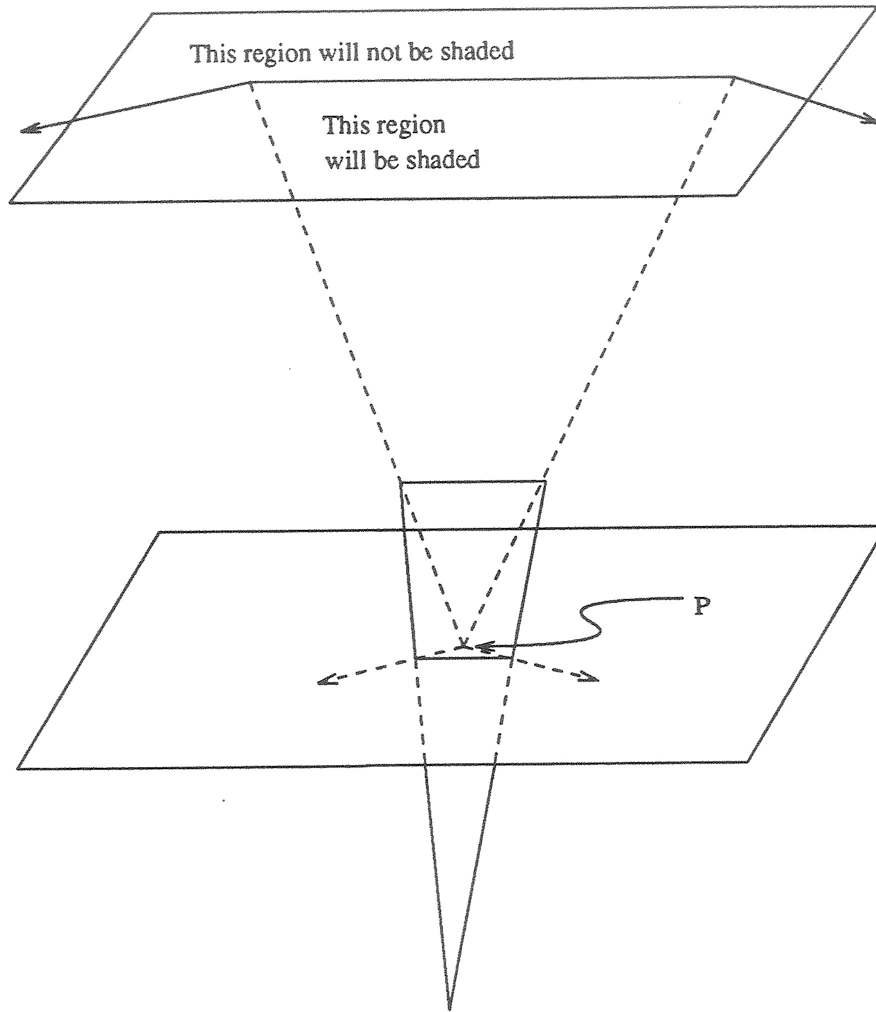

Figure 6: Calculation of infinite shading regions

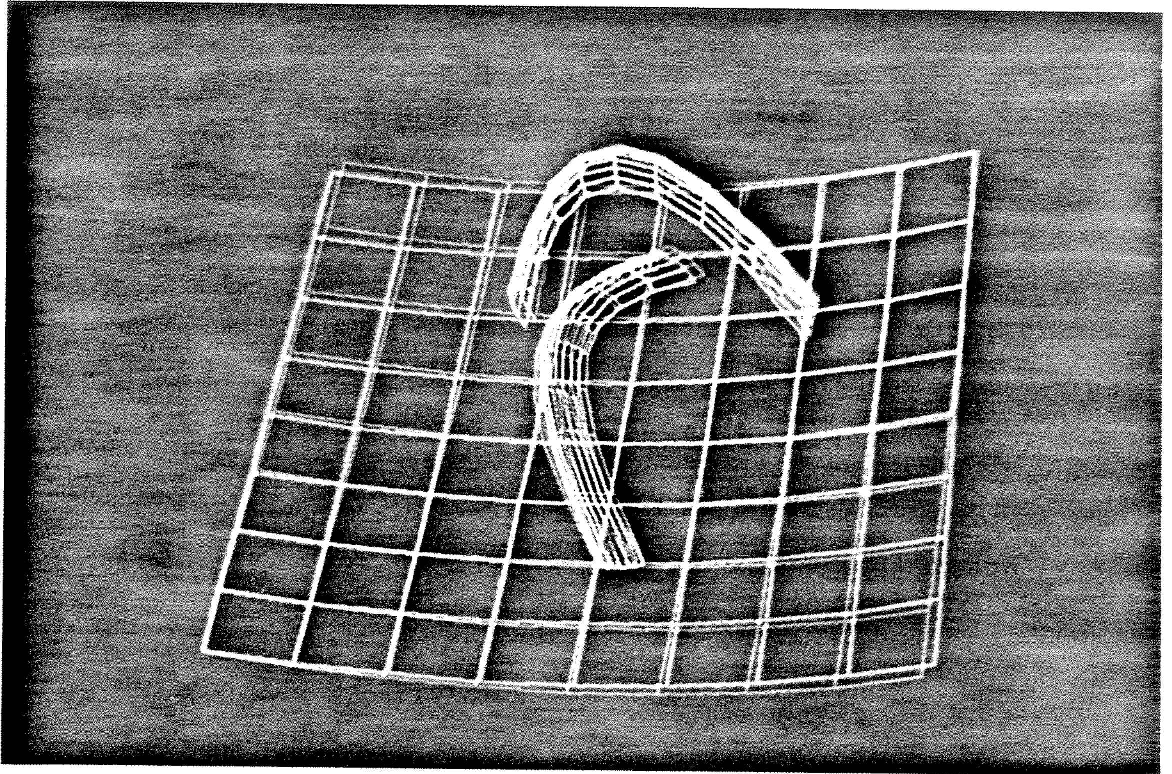

Figure 7

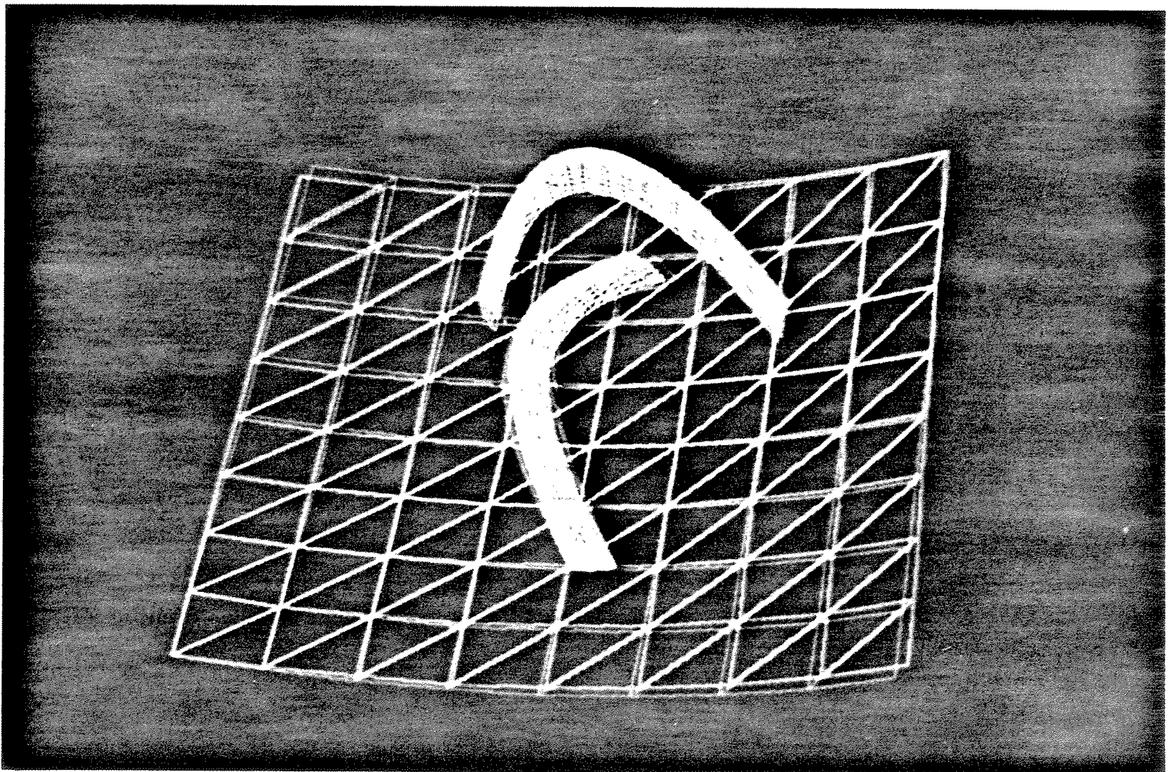

Figure 8

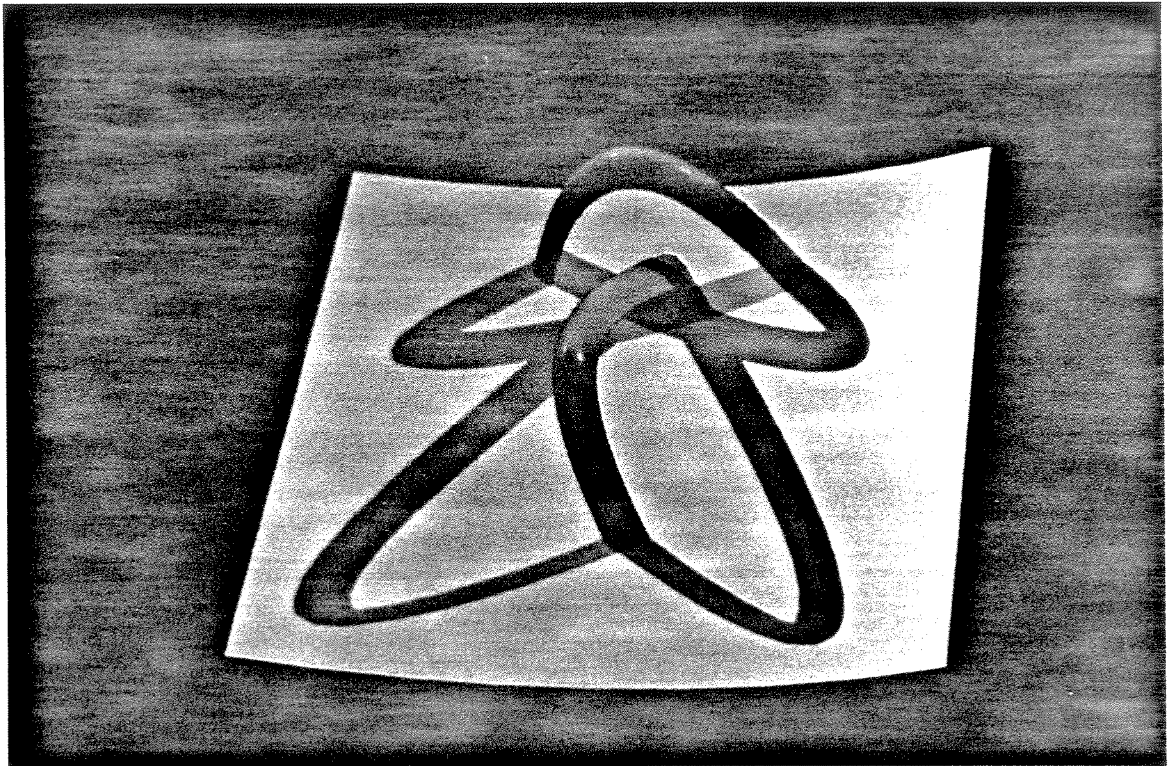

Figure 9

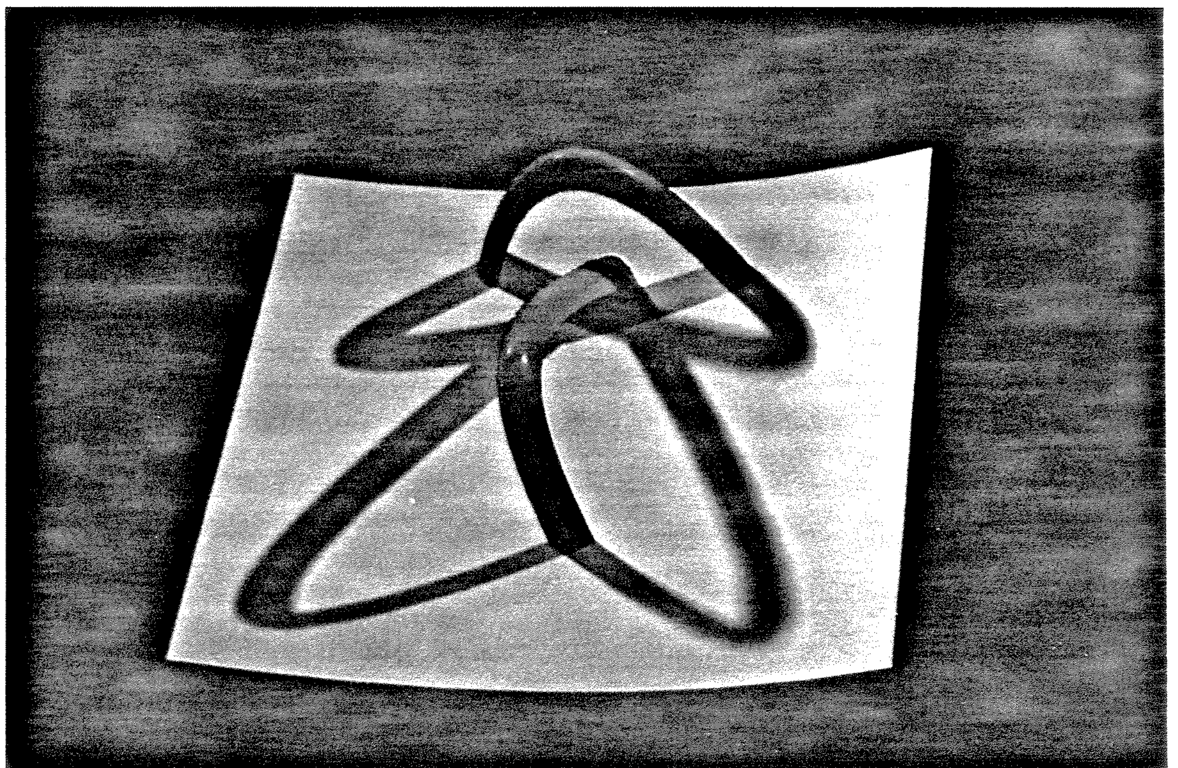

Figure 10

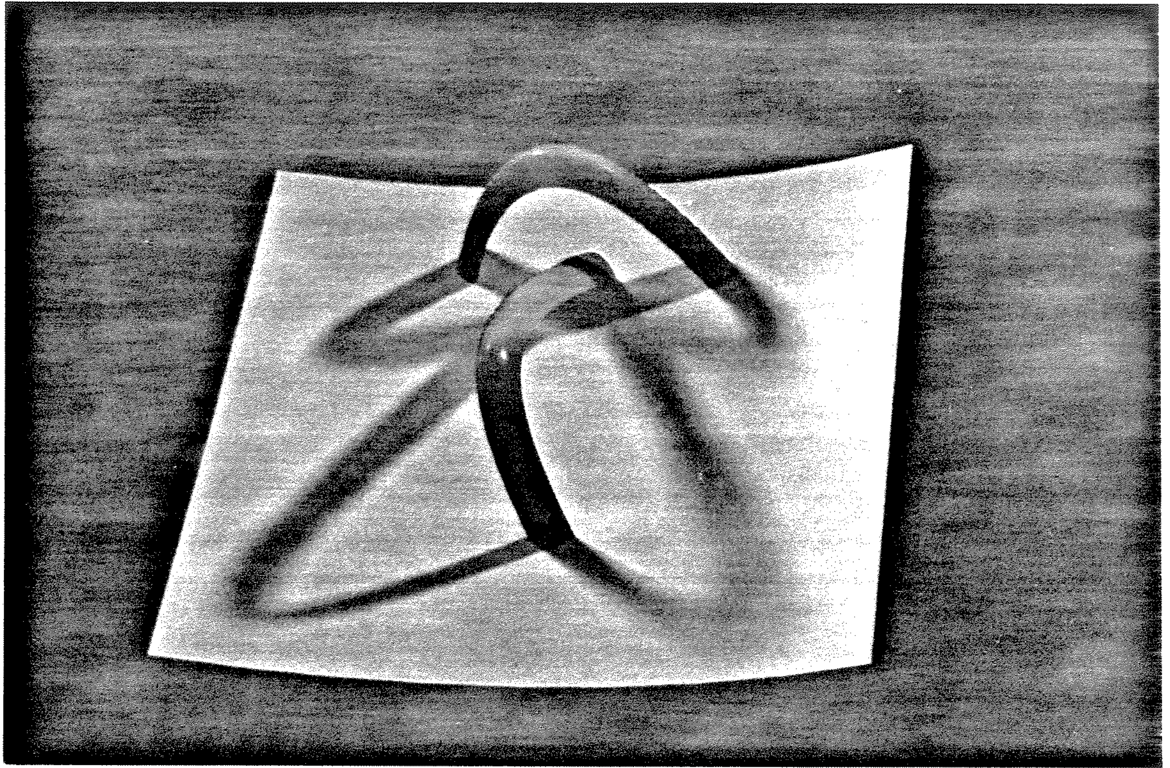

Figure 11

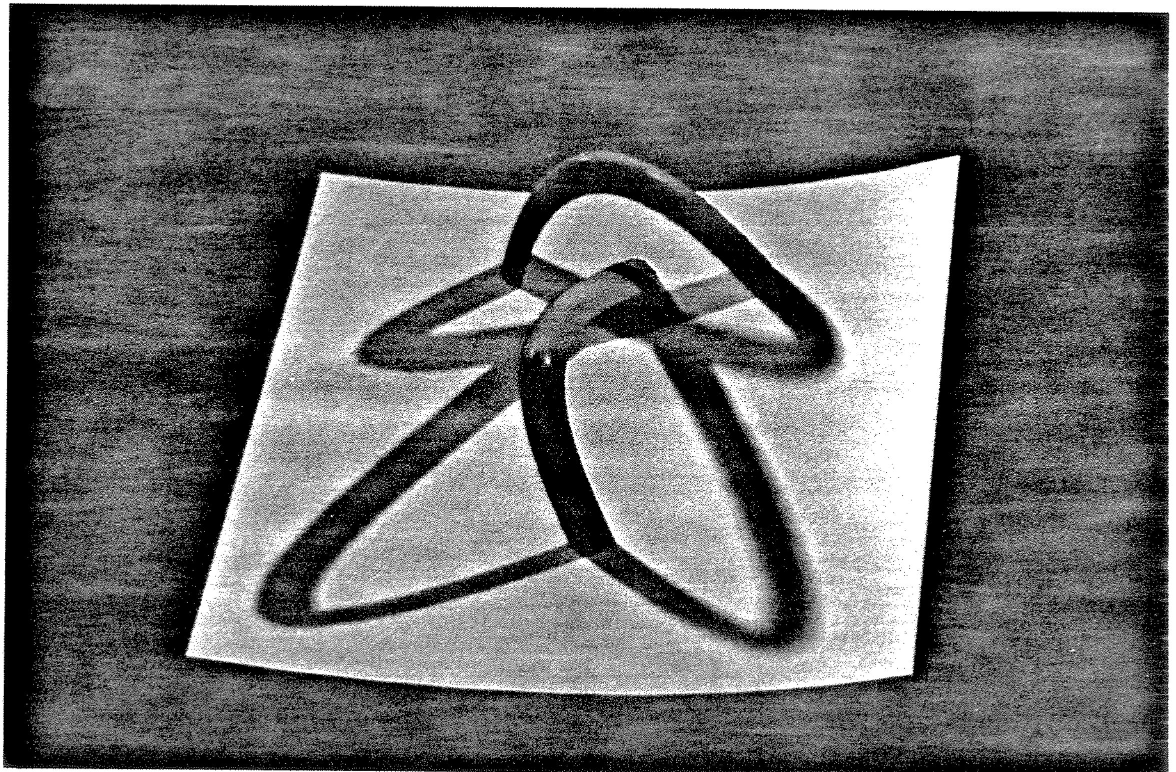

Figure 12

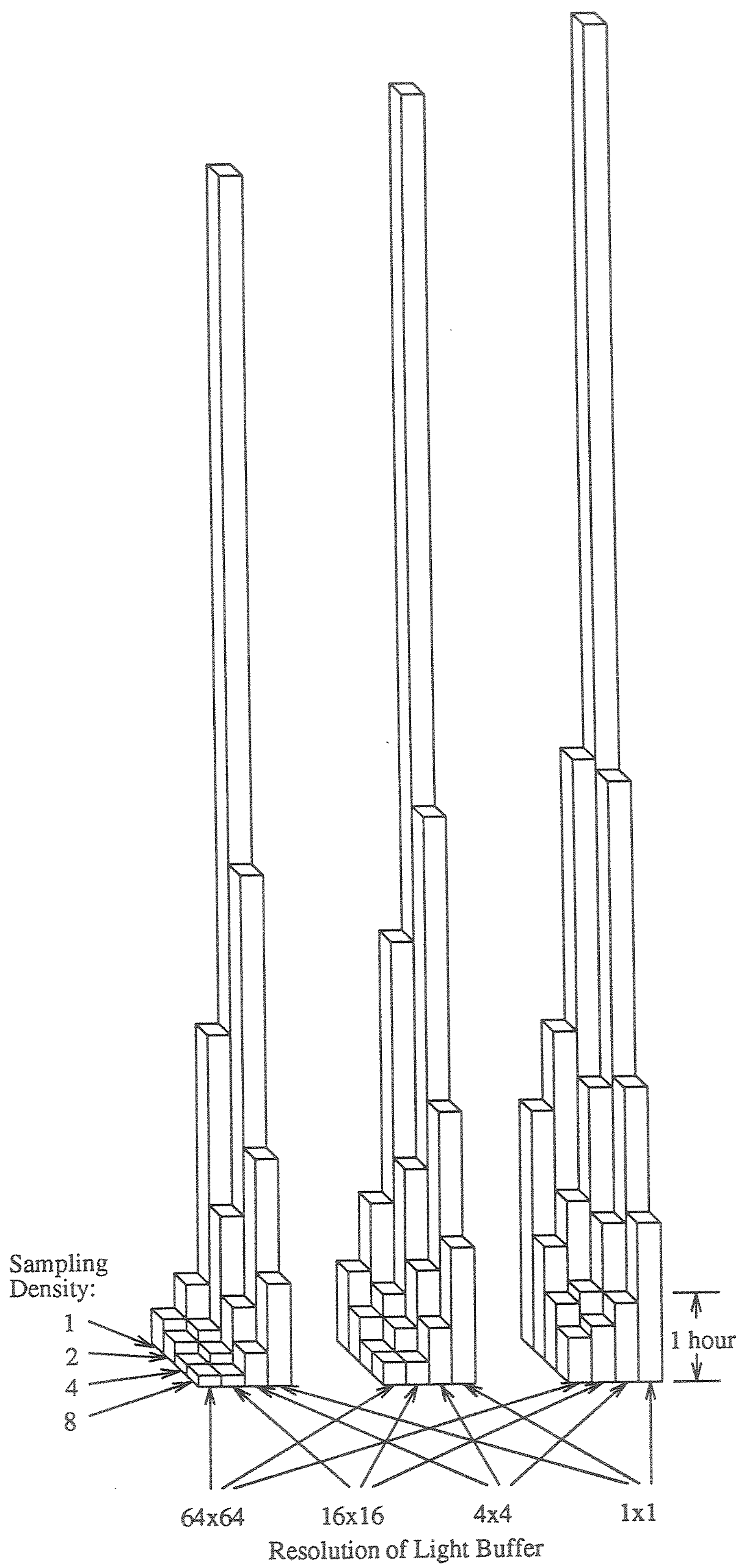

Figure 13: Timing Results, under different conditions