1: Introduction

Shading and shadows are determined by the characteristics of the surfaces, the distance of the object with respect to the light source and the viewer, the orientation of the object, and the properties of the light sources. These aspects are essential to the realism of images. Most of the previous work has been involved in dealing with point and directional light sources. However, many light sources are not well approximated by point and directional lights. It becomes then important to render the effects of linear and area light sources. Tanaka and Takahashi [1] extended the illumination model for linear light sources of Poulin and Amanatides [2] for perfectly diffusing polygonal light sources (Lambertian cosine distribution). They implemented the Phong shading model and then transformed the light sources from Cartesian coordinate system to polar coordinate system to compute the specular reflection. This allowed to produce more realistic scenes as polygonal lights can more closely approximate real light sources.

In this report, section 2 reviews the previous work on illumination models, including point, directional, linear, and area light sources. The area light source shading model proposed by Tanaka and Takahashi [1] is studied and implemented in section 2.2. Six resulting images are presented. In their paper, no shadows were computed. Section 3 presents related algorithms on shadow determination. An intuitive way to approximate the shadowing effects of the linear or area light sources is to use many point light sources. Unfortunately, aliasing problems arise when too few point light sources are used, or the computation becomes too expensive when many point light sources are used. The perfect solution consists in clipping the light by the projections of all the objects between the light and the point to shade. The intensity is therefore calculated only for the visible parts of the light. However, this process is fairly expensive and depends on the scene complexity.

After describing other shadowing algorithms presented in section 3 a new algorithm based on shadow maps is proposed. The details are given in section 4. The advantages and disadvantages of our approach are compared to other shadowing algorithms, especially shadow map-based algorithms. Performance evaluation is discussed in section 4.5. Section 5 concludes and discusses extensions of our work.

2: Shading Algorithms

We usually simplify the light reflection off a surface into two components : diffuse and specular. The diffuse reflection at a certain point depends on the intensity of the light source, the diffuse reflection coefficient of the surface, and the angle between the direction of the light source to the point and the surface normal. The specular reflection, which can be observed on shiny surfaces as highlights, depends on the intensity of the light source, the specular reflection coefficient of the surface, and the angle between the direction of reflection and the viewpoint direction. Both of them must be modeled to create realistic images. (Figure 1) Phong [3], Blinn [4], and Cook and Torrance [5] proposed early diffuse and specular shading models in computer graphics. Many more sophisticated models have been presented since them.

2.1 Linear Light Sources

Verbeck and Greenberg [6] were among the first ones to treat extended (linear and polygonal) light sources. They simply distributed a number of points on the light source, replacing the behavior of the extended light by the sum of the contributions of every points. The results are prone to aliasing and can be computationally expensive. Nishita et al. [7] derived an analytical solution for the diffuse reflection of surfaces illuminated by linear light sources. In their system, for faster computation, each linear light source is classified into one of three categories with respect to the point to shade: parallel, perpendicular, or skewed. Two analytical solutions are derived for the first and second cases. For the skewed case, the integration of the diffuse reflection is done by transforming the coordinate system and integrated numerically. However, the specular reflection is only handled as a series of point light sources. Therefore, the same sampling problems as above arise here.

Poulin and Amanatides [2] chose Phong's model to implement specular reflection caused by linear light sources because of its relative accuracy with reality and the possibility to compute and integrate it analytically at a moderately low cost. They deal with specular reflection by transforming the light source to a coordinate system different from the one used for diffuse reflection and by approximating the resulting integral with Chebyshev polynomials.

2.2 Polygonal Light Sources

In [1], an analytic solution was presented to determine the diffuse and specular components of surface reflection when illuminated by an area light source. The solution assumes a Lambertian emitting polygonal light source and integrates Phong's reflection.

The model in Nishita and Nakamae [8] is used to integrate the diffuse term. For the specular term, the equation in Phong's model is used because the specular reflectivity is rotationally symmetrical to the mirror reflection vector of the viewer's direction (Vector Vr). To simplify the specular reflection equation, the Cartesian coordinate system is transformed to the polar system and Vector Vr is chosen as the new Z axis. A unit sphere whose origin is the point to shade and of unit radius is constructed. Then the polygonal light source is projected to this sphere and triangulated by great circles to determine the area of integration. The coordinate system is rotated around the Z axis to obtain the values of angles for integrating areas in the polar coordinate system. Finally, the integrand is approximated by using Chebyshev polynomials which are integrated analytically. If regular samples are taken on the curve, Chebyshev approximation is assured to return the best (min-max) polynomial for a given polynomial degree.

This approach produces both diffuse and specular reflections and it can generate images that are more photorealistic than previous methods. Figures 2, 3, 4, and 5 are images generated by this method. Figure 2 (a) and (b) shows an office scene illuminated by an area light source from different view points. Highlight positions changed because they depends on the angle between the light source and the view point. Figure 3 and figure 4 show how the sharpness of specularly reflected light is controlled by the surface roughness. With a higher coefficient, the surface becomes smoother (less rough) and therefore the shape of the pentagonal light and star-shaped light can be observed from the image. Figure 5 shows a scene with three objects. The surface roughness and view angle are changed to demonstrate the result. These experimental results show that specular reflection is essential for photorealistic rendering.

3: Shadowing Algorithms

All the images presented in Section 2.2 (Figure 2, and 5) have no shadows and all objects seems to float in the air. Shadows are essential in rendering realistic images. Furthermore, shadows provide effective information indicating the positional relationships, shapes, and surface characteristics of objects. This can provide the observers with a more accurate comprehension in complex spatial environments. Moreover, the approximate location, intensity, shape, and size of light sources can be obtained from the correct rendered scenes with shadows. For point and directional light sources, shadows are simply determined by the binary visibility of the light. A point occluded by opaque objects is in umbra; otherwise it is completely illuminated. Hard shadow algorithms deal with the determination of in-umbra or in-light regions. For linear and area light sources, full occlusion from the light still produces umbra regions, but partial occlusion from the light creates penumbra regions. Soft shadow algorithms must calculate the fraction of opaque occlusion, not just the binary decision as for hard shadow algorithms. As we will see in the next sections, determining this fraction requires much more computationally expensive algorithms.

3.1 Hard Shadows

Ray tracing can be used for the surface visibility calculation. To calculate shadows in this context, an additional ray is shot from the intersection point to the light source. If the shadow ray intersects any object along the way, the intersection point is in shadow. The main advantage of ray tracing is that it can handle any ray type, whether primary, reflected, and refracted rays in an identical manner. Ray tracing can be extended recursively for global illumination computation.

Crow [9] introduced the shadow volume algorithm to generate umbrae. A shadow volume is defined by the light source and an object, and is bounded by a set of invisible shadow polygons. Any point within a shadow volume is in shadow of the light source. To determine if a point is in shadow, we count the number of front-facing (+1) and back-facing (-1) polygons between the eye and the point. A result greater than zero means the point is in shadow. The counter for the eye itself must also be computed, but only once per image.

Area subdivision shadow detection [10][11] is based on a two-pass hidden surface algorithm. The first pass computes an image from the view point of the light source. All surfaces are then divided into illuminated and in shadow. The second pass applies visibility determination from the eye. The results are combined to determine the pieces of each visible part of a polygon and the scene is scan-converted. The storage complexity depends on the spatial complexity of the scene with respect to the light sources. A robust clipping algorithm is required for dealing with concave polygons and polygons with holes.

The last hard shadow algorithm discussed here is the shadow map method [12]. It will be adapted later for our approach for linear light shadowing calculation. It uses image -precision calculations and supports primitives other than polygons. While a Z-buffer depth map stores the distances to the viewer for determining the surface visibility problem, the shadow map stores the distances with respect to the light for determining if a point is visible from the light source. Once computed, determining if a point is in shadow or not requires to identify the shadow map pixel it projects to and compare the stored distance to the distance of this point to the light. It can however have aliasing problems due to discretized map cells and the orientation of shadow map.

3.2 Soft Shadows

Soft shadow algorithms must determine the visible part of the light from the point to shade. For linear light sources, the light clipping algorithm must construct a triangle with the two end-points of the light and the point where illumination is being calculated. It then must test each object to check whether it intersects the triangle. If it does, the coordinates of the intersection points are projected on the linear light. The occluded part of the light is removed. Although this algorithm yields exact light segments, the cost can be significant as many objects have to be tested with respect to the plane supporting the light triangle. Nishita et al. [7] preprocess the scene. They project the contours of objects as planes from both end-points of the light. This way, two shadow polygons are generated. Then, their algorithm computes the corresponding convex hull. If a point is inside the convex hull, light clipping determines the visible parts of the light by projecting only the polygons associated with this convex hull. Its cost is high when the scene is complex and when the objects are rugged, because each object involves generating convex hulls of every other object.

Space subdivisions can reduce the number of objects subject to light clipping. Poulin and Amanatides [2] used 3-D scan conversion of voxels. They also proposed a linear light buffer scheme where a cylindrical buffer is oriented along the linear light. The 3-D space is split into sectors (Figure 6). In both cases, a list of object candidates to occlusion is stored within each spatial element. Tanaka and Takahashi [13] used a shadowing algorithm for linear light sources based on cylindrical buffer. Besides sectors, the band which is a subspace bounded by a pair of flaps rotating in the plan with the linear light source is used to subdivide the space again. (Figure 7) Sectors and bands together segment 3-D space into many subspaces called sections. Each cell of the buffer stores a list

of objects that lie within or intersect the section mapped to the cell. To compute the shadow of a point, we must first find its corresponding cell. All the objects stored in that cell are candidate objects to occlude the point. Five bounding-volumes formulas are applied to those objects to reduce the number of candidates. To optimize for polygons, they are subdivided into trapezia and triangles cut by a cylindrical scan-conversion algorithm before they are stored in the buffer. (Figure 8) This speeds up the computation of light clipping. The overall improvement makes the technique achieve over 10 times faster than the linear light buffer algorithm. This method can also be adapted to handle curved objects.

4: A New Approach for Computing the Shadows from Linear Light Sources

4.1 Review

In this section, we present techniques based on shadow maps to compute shadows of extended light sources. Shadow map has always been a powerful rendering technique for computing shadows from point and directional light sources. A shadow map is like an image of the scene, but viewed from the light source instead of from the viewer. The depth of the first visible point is stored at each pixel of the shadow map. Whether a point is illuminated or not is decided by comparing its distance to the light source with the value in the same direction kept in the shadow map pixel. If the value in the shadow map is smaller, there is at least one object lying between the point and the light source, therefore this point is in shadow. If the two values are equal, the point is then on the first object which the light ray meets in this direction, thus it is illuminated. Shadow map is an image-precision algorithm, and it can generate shadows for any object that can be scan-converted, including curved surfaces. Moreover, the computation time is linear in the number of pixels in the image produced. Memory required for shadow maps is known and fixed. Construction of shadow maps is similar to visibility determination in an image without the extra computation required by shadings, reflections, and refractions. Access to shadow maps is constant for any 3-D point. However, it is prone to the same aliasing problems as image-precision algorithms and memory required by each shadow map can be large if fine resolution is needed to reduce aliasing.

Shapiro and Badler [14] use many point light sources distributed randomly or evenly to approximate linear light sources. In their algorithm, a sufficient number of points

should be generated so that the addition of one more point to the set of point sources will not significantly affect the shading. The amount of light reaching any point in the penumbra region corresponds to the addition of the contributions of all visible point lights. Certain restrictions exist on the choice of objects that can be successfully rendered. Objects must be a union of convex, closed polyhedral pieces. Polygon faces must be consistently ordered. This algorithm requires considerable amount of time and space.

Chen and Williams [15] apply their view interpolation technique to shadow maps in order to generate soft shadows for linear light sources. A shadow map is computed first for each of the two end-points of the light source using a conventional rendering method. The shadow map for an in-between point on the linear source is interpolated from the two end-points shadow maps using a morphing method. Then the standard shadow map algorithm mentioned before is used to compute the visibility for in-between light positions. The process is repeated to generate more in-between points until a desired interval is reached. The resulting shadow images are combined to create the soft shadows for the linear light source.

Hole problem arises when an object is invisible from the two end-points, but becomes visible from an in-between point. The shadows of such objects are missed because the only available information can be derived only from the two shadow map. One way to solve this problem is to use multiple computed (not interpolated) shadow maps to reduce the lost of information, but at an increased space and time cost. Max and Ohsaki [16] present a method for reconstructing an image from precomputed Z-buffer views. They store multiple Z levels at each pixel, allowing hidden objects to be reconstructed from even a single view. This solves in part most of the hole problems.

Like the Z-buffer visible-surface algorithm, the shadow map algorithm is prone to aliasing; therefore, percentage closer filtering [17] is used to antialias the shadows for each image and to create soft shadow boundaries that resemble penumbrae. This is based on the fact that for an extended light source, a point to shade should project onto a region of the shadow map rather than a single pixel. The technique also reverses the order of the filtering and comparison steps in texture map filtering but applied to the shadow map. The Z-values of the shadow map across the entire region are first compared against the depth of the surface being rendered. This simple transformation converts the depth map under the region into a binary map, which is then filtered to give the proportion of the region in shadow. The resulting shadows have soft antialised edges. However, this algorithm does not compute exact penumbrae. An example is given in Figure 9. Consider a scene with either Object 1 or Object 2. In both cases, they have the same projection areas in the shadow map but result in the same penumbrae boundaries. In fact, the penumbrae should also depend on the relative distances between objects and light source which percentage closer filtering does not consider.

4.2 Implementation Theory

Our approach to generate shadows for linear light sources is based on the shadow map technique. A multiple-level shadow map is constructed for each end of the linear light. Object information is derived by comparing the two end-point shadow maps and is then stored in polygonal shadow map. During rendering, the visible light segments for a point to shade are calculated from the objects information. Integrating over all visible light segments gives the light intensity at the point which is reflected towards the viewer.

First, a given resolution shadow map is associated with each end-point (i.e. *end 0* and *end 1*) of the light. The shadow maps are aligned with the linear light source. They are numbered as *shadow map 0* and *shadow map 1*. (Figure 10) Instead of using a standard 2-D floating point array as the data structure of the shadow map, it is augmented by a link list in each cell. Each cell stores the visibility information for a direction from its end of the light. When a ray is shot from the end-point of the light source, all intersection points with objects are kept in depth order in the corresponding cell. (Figure 11) The reason to use multiple-level shadow maps is to keep all information of objects which are hidden behind other objects and thus reduce the hole problem.

Because the two shadow maps have identical resolutions and orientations, two rows with same index refer to the same shadow map scan-line. Therefore, they also refer to the same 3-D plane. The shadow maps are aligned with the linear light source; each row in the shadow maps is also aligned with the light source. If a cell has n elements and the next cell in the same row has $n+1$ elements, this means the ray from the light source through the next cell direction has at least one more intersection point with some object. The next cell is defined as an *entering critical point* because it is the first intersecting cell with some object. If a cell has $n+1$ elements and the next cell in the same row has n

elements, this means the ray from the light source through the cell direction does not intersect with some object any more. The cell is defined as a *leaving critical point* because it is the last intersecting cell with some object. Scan-lines go toward the same direction as the shadow map column index increases. The *polygonal shadow map* is formed by a 1-D array which contains as many elements as the number of rows in the shadow maps. Each row contains a link list of all entering and leaving critical points. (Figure 12)

Since polygonal shadow maps are derived from shadow maps, rows with the same indices also refer to the same scan-line position. To compute the correct visibility for the light, we create the following construction. Draw two parallel lines perpendicularly to the light. These two lines have the same resolution in columns than the scan-lines they represent . Draw the depth value of each column along the vertical axis. This creates a 3-D coordinate system with two planar discontinuous curves representing the depth changes along the same scan-line in the two shadow maps. (Figure 13). For each point appearing in both shadow maps, we can connect its two 3-D positions in our construction. This forms a discontinuous ruled surface. The discontinuities occur at the critical points; their segments are called *critical lines*. For a position *P* in the scene, we must find its positions in shadow map 0 and shadow map 1 and connect the two positions. If *P*'s line intersects any critical lines, this means the visibility of the light source as seen from point *P* changes at those intersection points.

An example to calculate the point *Pa* on the light where this visibility changes is given in Figure 14. Let (*i, a0*) be the position of the entering critical point A in shadow map 0 and (*i, a1*) be the corresponding position of critical point A in shadow map 1. (*i, p0*) denotes the position of point P in shadow map 0 and (*i, p1*) denotes its position in shadow map 1. The 2-D parallel projection along the depth of Figure 13 bottom is redrawn at Figure 14 bottom with a 90 degree rotation. Critical line (*i, a0*)-(*i, a1*) and line

(*i, p0*)-(*i, p1*) intersect at (*i, a'*). If there were a shadow map at *Pa*, then the corresponding position for entering critical point A would be at (*i, a'*). Pa can be derived by similar triangle as a ration such that :

$$\text{Pa} = \text{end } 0 + \text{Vector}(\text{end } 0, \text{end } 1) \times \left(\frac{\text{a0} - \text{p0}}{(\text{a0} - \text{p0}) + (\text{p1} - \text{a1})} \right)$$

The first visible light segment is from *end 0* to *Pa*. The second intersection *Pb* with critical line B can be derived with the same method. *Pb* to *end 1* is the second visible light segment from point *P*. To compute the proper shading, we then simply integrate the two visible segments to obtain diffuse and specular intensity for point *P*.

4.3 An Example

Another example is now given to illustrate in more detailed the algorithm. (Figure 15) A linear light source, a triangle (blocker), and a plane on which the shadow will be cast form the 3-D scene. First, we construct *shadow map 0* for *end 0* and *shadow map 1* for *end 1*. We then find all critical points for the two polygonal shadow maps. Row *i* has an entering critical point at (*i, j0*) and a leaving critical point at (*i, k0*) in shadow map 0, and a entering critical point at (*i, j1*) and a leaving critical point at (*i, k1*) in shadow map 1.

To simplify the situation, representative points *a, b, c, d,* and *e* located on the plane are aligned with the light source. Their respective positions are at (*i, a0*), (*i, b0*), (*i, c0*), (*i, d0*) and (*i, e0*) in shadow map 0 and (*i, a1*), (*i, b1*), (*i, c1*), (*i, d1*) and (*i, e1*) in shadow map 1.

We compute their corresponding positions in the two shadow maps and connect each pair of points (i.e. *a0-a1*, *b0-b1*, *c0-c1*, *d0-d1*, and *e0-e1*) in the polygonal shadow

map. (Figure 16) Let us look at the segment with respect to the critical lines, (*a0-a1*) does not intersect any critical line as it is located before the entering critical line (*j0-j1*). Therefore, point a is visible from both end-points of the light. (*b0-b1*) intersects the entering critical line (*j0-j1*) at point *jb*. It means there exists a position *R* on the light source for which point *b* is mapped to (*i, jb*) in *R*'s shadow map and for which point *b* is on the entering boundary of the shadow. Therefore, the light segment from *end 0* to *R* is visible at point b (penumbra). *R* is calculated as :

$$\text{Position R} = \text{end } 0 + \text{Vector(end0, end1)} \times \left(\frac{\text{b0} - \text{j0}}{(\text{b0} - \text{j0}) + (\text{j1} - \text{b1})} \right)$$

(*c0-c1*) has no intersection point with the two critical lines as it is located after the entering critical line (*j0-j1*) and before the leaving critical line (*k0-k1*).Therefore, point *c* is completely in shadow (umbra). The segment (*d0-d1*) intersects the leaving critical line (*k0-k1*) at point *kd*. This means there is a position *Q* on the light source for which point *d* is mapped to (*i, kd*) in *Q*'s shadow map and point *d* is on the leaving boundary of the shadow. Therefore, the light segment from *Q* to *end 1* is visible to point *d* (penumbra). And *Q* is obtained by :

$$\text{Position Q} = \text{end } 0 + \text{Vector(end0, end1)} \times \left(\frac{\text{d0} - \text{k0}}{(\text{d0} - \text{k0}) + (\text{k1} - \text{d1})} \right)$$

(e0-e1) does not intersect any critical line so it is illuminated by the entire linear light.

4.4 Algorithm

The algorithm to compute the shadows is described as follows:

Step 1. Let *s* be a 3-D point in the scene. Find the position of *s* in *shadow map 0*, denote it by (*i, s0*). Similarly, find the position of *s* in *shadow map 1*, denote it by (*i, s1*). Shadow maps 0 and 1 include all objects here, that is they contain a list of intersection points for each pixel. If any of *i, s0,* and *s1* exceeds the range of the shadow maps, the illumination must be computed with another technique such as shooting random rays towards the linear light.

Step 2. Calculate the number *n0* of critical points with index values smaller than *s0* in row *i* of polygonal shadow map 0. Calculate the number *n1* of critical points with index values smaller than *s1* in row *i* of polygonal shadow map 1. (*n0-n1*) corresponds to the number of intersection points between (*s0-s1*) and all critical lines in row *i* of the polygonal shadow maps.

Step 3. If the line (*s0-s1*) intersects the critical line (*i, j0*)-(*i, j1*) at *js*, the light position where *s* is mapped to (*i, js*) is computed as:

$$\text{end } 0 + \text{Vector}(\text{end0},\ \text{end1}) \times \left(\frac{\text{s0} - \text{j0}}{(\text{s0} - \text{j0}) + (\text{j1} - \text{s1})} \right)$$

Step 4. Pair all light positions found in Step 3. The visible segments of the light are the segments after a leaving critical point and/or before an entering critical point.

Step 5. Integrate all the visible light segments to obtain the diffuse and specular intensity.

As the scene becomes more complicated, critical points in the two shadow maps might have different orders. At this moment, care must be taken to properly pair the light positions found in Step 3. Figure 17 shows such an example.

4.5 Evaluation

The resolution of the shadow maps influences the resulting image and the performance very much. In this experiment, an SGI Indigo 2 is used. (The details are 1 150 MHz IP22 Processor, FPU: MIPS R4010, CPU: MIPS R4400, Main memory size: 64 Mbytes) For a 256*256 (pixels) image, when the resolution of the shadow map is less than 200*200, the aliasing problems are obvious. Each element in the shadow maps has three fields: one double value to keep the distance to the light end, three double values to record the intersection point coordinate, and a pointer to point to the next element. These add up to 40 bytes. For a 500*500 resolution shadow map, if each cell has more than one element, it requires at least 10 Mbytes. A linear light source has two such shadow maps. Therefore, this technique requires a lot of memory when a high resolution is used.

The time in seconds to construct the various shadow maps with different resolutions are shown in Table 1. Two 256*256 sample images are used here. (Figure 18) The obvious advantage is the rendering time is independent from the scene complexity. Figure 18(b) shows a multiple levels scene. The light source is not aligned with the objects in Figure 19(a). The planes are slant in Figure (b). Figure 19(c) is a case where the order of some critical points are crossed as explained in Figure 17. Figure 19(d) presents another example.

The first image

resolution	shadow map 0	shadow map 1	polygonal s. m.	rendering
100	1.18	1.24	<0.0.1	18.00
200	4.98	4.96	0.03	17.26
300	11.14	11.33	0.06	18.52
400	19.99	21.16	0.12	19.11
500	31.58	33.82	0.17	19.23

The second image

resolution	shadow map 0	shadow map 1	polygonal s. m.	rendering
100	1.28	1.34	0.01	17.93
200	5.47	5.59	0.04	18.09
300	12.39	12.54	0.07	17.24
400	22.18	23.14	0.12	18.23
500	34.84	37.08	0.19	19.45

Table 1. The time costs for two sample images (in seconds)

5: Discussion

There is a tradeoff between time, space, and accuracy. The computational time and required memory for constructing the shadow maps and polygonal shadow maps are proportional to their resolutions. Aliasing problems arise when the shadow maps have insufficient resolutions. However, after the shadow maps have been constructed, the rendering time of shadows is linear in the resolution of the image and independent from the scene complexity. Moreover, the rendering time is less than the light buffer algorithm. Table 2 shows that the rendering time with a polygonal shadow map algorithm is about 68% of the rendering time of the light buffer algorithm. The resolution of the light buffer has 36 sections.

Image Size	Polygonal shadow map	Linear light buffer
100*100	2.6	3.06
200*200	9.95	11.7
300*300	22.48	26.4
400*400	40.18	49.39
500*500	68.45	77.72
1000*1000	248.87	292.31

Table 2: The rendering time of two algorithms (The second image of Figure 17)

The shadow map memory requirement can be reduced greatly by storing only the distance value in each element. The intersection point coordinate could be recomputed on demand from the distance and shadow map index vectors. Besides, the distance value could be saved as integer like for Z-buffer. This can reduce the size from currently 40 bytes per shadow map element down to 4 bytes.

There are some problems resulting from the resolution and sampling problem. Although two shadow maps have the same resolution, two scan-lines may still have differences. Because all critical points are found by sampling so the positional difference may cause some edges or corners of objects to appear in one shadow map but not in the other. This brings up problems in matching information when constructing the polygonal shadow map. Moreover, determining the best resolution for the shadow map is still based on trial-and-error experiment. The quality resulting with a given resolution can only be known after the rendering. Worst, even though a given resolution would give good results, another higher resolution may show other artifacts.

Resolution and sampling problem also affect how close two intersection points can be considered as identical. A larger error range should be allowed at a lower resolution. There are still some other limitations. With this technique, non-polygonal objects are allowed but curved surfaces cannot be rendered. When a surface with large curvature is rendered, rays with the same row indices from the light ends hardly have the same intersection points. If there are no common critical points, the polygonal shadow map cannot be derived. (Figure 20) Therefore, this technique does not apply. The possible solution to render curved surfaces is to combine view interpolation technique [18]. When a very long linear light source is used, the two end shadow maps may not see all the same objects. In this case, our technique does not apply.

Comparing to view interpolation, this technique uses a multiple-level shadow map data structure. This avoids the problems with holes, but consume more space and time. Rays from the light source will stop only when they do not intersect any more objects. All intersection points along the way are recorded in shadow maps. No object is therefore missed. This technique solves the hole problem. An example is shown in Figure 21.

Scan-line algorithm [18] [19] [20] [21] offers an attractive alternative to improve this technique. We can construct each row in the two shadow maps using the scan-line algorithm so p critical points can be derived accurately because scan-lines are not discretized along the line direction. This will greatly reduce the sampling problem. Furthermore, the same rows in two shadow maps should be built simultaneously to obtain the polygonal shadow map. When enough information is kept in the polygonal shadow map, shadow maps are not needed any more. This will reduce the memory requirement tremendously.

Overall speaking, this polygonal shadow map technique has a potential to generate accurate soft shadows efficiently. Especially, in some applications such as the background of the animation where the shadow maps only need to be computed once before rendering. Only the moving objects in a 3-D scene need to be recomputed in each shadow map (assuming fixed light sources). However, the conventional method to build the shadow maps should be improved to get correct and more accurate information. Scan-line algorithm can be expected to be the key of improvement.

References

[1] Toshimitsu Tanaka and Tokiichiro Takahashi, "Shading with area light sources," In W. Purgathofer, editor, Eurographics 1991, North-Holland, Sept. 1991, pages 235-246.

[2] Pierre Poulin and John Amanatides, "Shading and shadowing with linear light sources," In C. E. Vandoni and D. A. Duce, editors, Eurographics 1990, North-Holland, Sept. 1990, pages 377-386.

[3] B. Phong, "Illumination for computer generated pictures," Communications of the ACM, Vol. 18, No. 6, June 1975, pages 311-317.

[4] J. Blinn, "Models of light reflection for computer synthesized pictures," SIGGRAPH 1977, Vol. 11, No. 2, July 1977, pages 192-198.

[5] R. Cook and K. Torrance, "A reflectance model for computer graphics," SIGGRAPH 1981, Vol. 15, No. 3, Aug. 1981, pages 307-316.

[6] C. Verbeck and D. Greenberg, " A comprehensive light source description for computer graphics," IEEE Computer Graphics and Applications Vol. 4, No. 7, 1984, pages 66-75.

[7] Tomoyuki Nishita, Isao Okamura, and Eihachiro Nakamae, "Shading models for point and linear sources," ACM Transaction on Graphics, Vol. 4, No. 2, Apr. 1985, pages 124-146.

[8] T. Nishita and E. Nakamae, “Half-tone representation of 3-D objects illuminated by area sources of polyhedron sources,” Proceedings IEEE Computer Software and Application Conference, 1983, pages 237-242.

[9] F. C. Crow, "Shadow algorithms for computer graphics," SIGGRAPH 1977, Vol. 11, No. 2, July 1977, pages 242-247.

[10] Tomoyuki Nishita and Eihachiro Nakamae, “An algorithm for half-tone representation of three-dimensional objects,” Information Processing in Japan, Vol. 14, 1974, pages 93-99.

[11] P. Atherton, K. Weiler, and D. Greenberg, “Polygon shadow generation,” SIGGRAPH 1978, Vol. 12, No. 3, Aug. 1978, pages 275-281.

[12] L. Williams, “Casting curved shadows on curved surfaces,” SIGGRAPH 1978, Vol. 12, No. 3, Aug. 1978, pages 270-274.

[13] Toshimitsu Tanaka and Tokiichiro Takahashi, "Fast shadowing algorithm for linear light sources," Eurographics 1995, Vol. 14, No. 3, 1995, pages 205-216.

[14] Lynne Shapiro Brotman and Norman I. Badler, “Generating soft shadows with a depth buffer algorithm,” IEEE Computer Graphics and Application, Vol. 4, No. 10, 1984, pages 5-38.

[15] Shenchang Eric Chen and Lance Williams, “View interpolation for images synthesis," SIGGRAPH 1993, Vol. 24, No. 4, Aug. 1993, pages 279-288.

[16] Nelson Max and Keiichi Ohsaki, “Rendering trees from precomputed Z-buffer views,”

[17] William T. Reeves, David H. Salesin, and Robert L. Cook, "Rendering antialiased shadows with depth maps," SIGGRAPH 1987, Vol. 19, No. 3, July 1987, pages 283-291.

[18] C. Wylie, G. W. Romney, D. C. Evans, and A. C. Erdahl, “Half-tone perspective Drawings by computer,” FJCC 67, Thompson books, Washington, DC, 1967, pages 49-58.

[19] W. J. Bouknight, “A procedure for generation of three-dimensional half-tone computer graphics presentations,” CACM, Vol. 13, No. 9, Sept. 1970, pages 527-236.

[20] W. J. Bouknight and K. C. Kelly, “A procedure for producing half-tone computer graphics presentations with shadows and movable light sources,” SJCC, AFIPS Press, Montvale, NJ, 1970, pages 1-10.

[21] G. S. Watkins, “ A real time visible surface algorithm,” Ph. D. Thesis, Technical report UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, University of Utah, Salt Lake City, UT, June 1970.