The Visual Computer

Computing caustic effects by backward beam tracing

Jung-Hong Chuang,
Shih-Ann Cheng

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

Caustic effects produced by the transport of light from specular surfaces to diffuse surfaces are a common type of optical effect that cannot be modeled by ray tracing. We propose a two-pass algorithm to model caustic effects efficiently and reliably. In the proposed method, information on transmitted light beams is collected in a tree structure, which is used to compute the intensity efficiently. The method does not require the polygonization of diffuse surfaces and can easily be combined with any rendering algorithm.

Key words: Caustic effects – Backward beam tracing – Ray tracing

Correspondence to: J-H Chuang

1 Introduction

Ray tracing is an effective technique for producing realistic images of 3D scenes (Appel 1968; Arvo and Kirk 1987; Kajiya 1982; Kajiya 1983; Glassner 1984; Hanrahan 1983; Mitchell and Hanrahan 1992; Wallace et al. 1987). In addition to providing shading effects, the ray tracing technique allows users to generate many other optical effects, such as reflection, refraction, and shadow. When a light ray fired from a light source hits a diffuse surface after intersecting specular surfaces, it emits equally in all directions. The traditional ray tracing cannot simulate light transport of this kind, hence the ray-tracing method fails to simulate the resulting optical effects. Tracing from light sources is one way of solving this problem. In backward ray tracing, the problem becomes trivial, because light rays can transmit through specular surfaces and propagate to the diffuse surface.

Methods using backward ray tracing and backward beam tracing have been proposed to simulate light transport from specular surfaces to diffuse surfaces (Arvo 1986; Chen et al. 1991; Heckbert and Hanrahan 1984; Shinya 1987; Sillion and Puech 1989; Wallace et al. 1987; Watt 1990), but the algorithms used in these methods are costly in terms of both computation time and memory space. For example, the major disadvantage of Watt's algorithm (Watt 1990) is that it traces a large number of beams in backward tracing, especially in areas of high curvature. Other disadvantages are that the beam tracing is restricted to polygonal environments and that the resolution of the polygonal mesh is very critical. Moreover, the computation of the beam-object intersection is complicated and is difficult to speed up.

We propose a two-pass algorithm to compute light transport from specular surfaces to diffuse surfaces. This algorithm attempts to find all paths from point light sources passing through specular surfaces to a point on a diffuse surface so that the intensity on the diffuse surface can be determined accurately. The proposed method comprises two steps: a preprocessing step and a rendering step. The preprocessing step establishes a tree structure for each point light source. This tree structure record information on the reflection and refraction of light from the point light source through polygons on specular surfaces. When computing the intensity of a point on a diffuse surface, we

The Visual Computer (1995) 11:156–166

must find all specular surfaces that contribute to the intensity of the point. Since the intensity computation is necessary only for visible points, the proposed algorithm minimizes the amount of computation in the preprocessing step and leaves most of the computational load to the second step. The tree structure is used to develop methods for discarding, without actually testing their visibility, specular polygons that have no effect on the visible point. As a result, much unnecessary computation can be avoided. The algorithm's structure allows it to be easily combined with any rendering algorithm.
Section 2 briefly describes the global illumination and caustic effects. In Sect. 3, the proposed two-pass algorithm for computing caustic effects is discussed. Section 4 presents some concluding remarks.

2 Global illumination and caustic effects

2.1 Global illumination

To produce a realistic image, the lighting interactions between objects with different reflectivity must be taken into account. This shading component is termed *global illumination.* Surfaces are traditionally classified as specular surfaces and diffuse surfaces, depending on their reflectivity (Wallace et al. 1987). Four types of light transport between two surfaces are possible:

1. Diffuse to diffuse
2. Diffuse to specular
3. Specular to diffuse
4. Specular to specular

Techniques such as ray tracing and radiosity are successful in dealing with global illumination in only a few types of light transport. Traditional ray tracing can model only types 2 and 4, because rays travel in a direction opposite to that in which light is propagated. Only light transfer via the reverse path is taken into account. Since the paths from a diffuse surface to other surfaces are not reversible, no secondary ray can be generated once a view ray hits a diffuse surface. The radiosity method can model light transport type 1 very well, but it performs poorly when handling directional reflection or refraction. Many algorithms combine ray tracing and the radiosity method to compute global illumination for all types of light transport (Campbell and Fussell 1990; Chen et al. 1991), with increased cost in terms of both time and space.

2.2 Caustic effects

We are especially interested in light transport from specular surfaces to diffuse surfaces. In Fig. 1, the major difference between picture A and picture B is that the bright area D appearing on picture A is the result of the light reflected from a nearby mirror M. Such caustic effects cannot be handled by ray tracing. The term *caustic* comes from the field of optics. Caustics are lighting effects on a diffuse surface due to light reflecting from a curved specular surface or refracting through a specular object. An example of a caustic effect is sunlight refracted through a magnifying glass so that a hot spot forms behind the lens. To model such effects, several methods have been proposed, e.g., (Arvo 1986; Watt 1990). Backward ray tracing (Arvo 1986) is a two-pass algorithm. In the first phase it fires a large number of rays from the light source to the scene and constructs an illumination map for each object that intersects the light rays. An illumination map is a grid of data points that is pasted onto each object in the scene in much the same way as in texture mapping. When a light ray intersects an object, the energy carried by the light ray will be distributed to the nearby data points on the illumination map, and the illumination map will be updated when other objects are intersected by the reflected or refracted rays. Because an object may intersect the light rays more than once, the energy from the light rays accumulates on its illumination map. The second phase of backward ray tracing uses general rendering methods, such as a Z-buffer, scanline, or ray-tracing algorithm. The intensity of a visible point can be linearly interpolated by nearby data points on the illumination map. This approach is highly prone to aliasing problems because of the point sampling and the discretization of the illumination map. The illumination map must have a greater number of data points for a scene with a high frequency of change in the intensity. Computationally, this approach is generally extremely expensive.

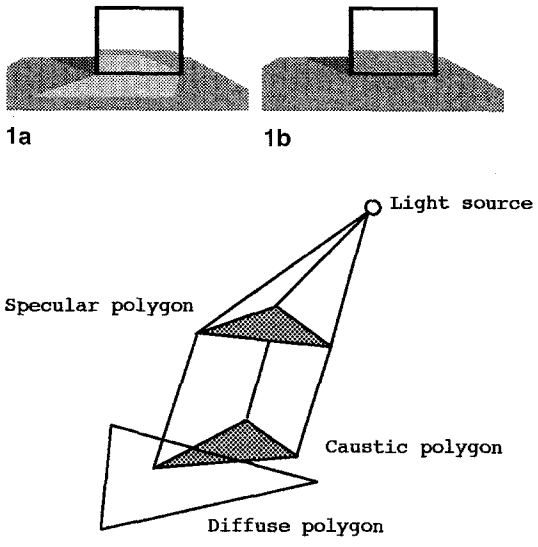


Fig. 1 a, b. Picture **a** with caustic effect; **b** without caustic effect

Fig. 2. Backward beam tracing and the caustic polygon

Another two-pass method that employs backward beam tracing is proposed in Watt (1990). Beam tracing is not a point sampling technique, and hence it resolves most of the aliasing problems. This method requires the specular and diffuse surfaces to be discretized into small polygons, referred to as specular polygons and diffuse polygons, respectively. Each specular polygon defines a light beam by firing rays from its vertices to the light source. For each specular polygon, the *transmitted light beam* is constructed by reflecting or refracting light rays with respect to the vertex normals. The transmitted light beam is swept through the entire scene to test for intersection of diffuse polygons. When a transmitted light beam intersects a diffuse polygon, it is projected onto the plane of the diffuse polygon. The projection forms a *caustic polygon*, as shown in Fig. 2. Because a diffuse polygon may intersect more than one beam, it may associated with more than one caustic polygon. The second phase is a view-dependent rendering process. The diffuse component of the intensity of a diffuse polygon is the sum of the intensities of the caustic polygons associated with it. The intensity of a caustic polygon is computed in a way that resembles the form factor of the radiosity method. The time complexity of Watt's method is proportional to the product of n_s, n_d, and n_l, where n_s and n_d are the number of specular polygons and diffuse polygons, respectively, and n_l is the number of point light sources. As stated in Watt (1990) and Watt and Watt (1992), the specular polygons must be much smaller when modeling the caustic effect than when modeling the surface itself. Hence the method for computing caustic effects is generally computationally expensive when the specular surfaces and diffuse surfaces are geometrically complex. Moreover, when Watt's method is incorporated with ray tracing, problems arise in ray tracing the polygonal diffuse surfaces (Snyder and Barr 1987).

3 The computation of caustic effects

The method in Watt (1990) actually traces beams backward and computes all caustic polygons associated with each diffuse polygon. As already stated, the resolution of the specular polygons must be very high for the accurate computation of the caustic effect. This implies that the computational cost depends on the resolution of the polygonization and may be extremely high. The algorithm proposed here is similar in structure to Watt's method, but it does not actually trace the beam backward. In the first phase of the algorithm, all the information on the transmitted light beams is recorded in a tree structure for efficient searching in the second phase. The intensity computation in the second phase can be incorporated into any rendering algorithm. When a point on a diffuse surface is rendered, the diffuse component of the point's intensity is the accumulated contribution of all transmitted light beams that intersect the point. The tree structure constructed in the first phase facilitates the search for transmitted light beams that intersect a given visible point. The light beam transmitted from each point light source is enclosed by a cone and the cones are merged recursively into a tree structure, called a *cone tree*. When a point is rendered, the point is tested recursively against each cone tree in a top-down fashion. If the point is outside the cone of the current node then the point cannot be inside

its descendant cones. If the point is inside the cone, the test continues until the enclosing leaf nodes are reached. The algorithm then checks whether the point is inside the transmitted light beam corresponding to each of the leaf cones. All transmitted light beams, from all point light sources, that intersect the point contribute their intensities to the point. With the cone tree structure, the time complexity for determining the caustic effect on a diffuse point is proportional to the sum of the height of the cone trees. Hence the proposed algorithm can efficiently compute caustic effects for scenes involving complex specular surfaces where polygonizations of very high resolution are generally necessary. In the following subsections, we describe how the cone tree for each point light source is constructed.

3.1 Cone tree construction

3.1.1 Constructing cones that enclose transmitted light beams

Enclosing the transmitted light beam with a cone reduces the number of beam-point intersection tests, since points outside the enclosing cone cannot be inside the beam, and thus only points inside the cone need to be tested against the transmitted light beam. The cone that encloses a transmitted light beam is determined according to the transmitted rays. It is difficult to determine the enclosing cone, since transmitted rays generally do not intersect at one point. Two methods are presented here: one constructs the cone in the unit sphere and the other uses two overlaid cones.

The single cone approach. Normalized ray vectors are mapped to a unit sphere and the cone that encloses the normalized ray vectors is found, as depicted in Fig. 3. With such a cone, the problem of locating the apex of the enclosing cone does not exist and the point-cone intersection can be tested easily. The axis of the cone is the unit vector *Axis* on the unit sphere such that the maximum angle between *Axis* and the normalized ray vectors is minimized. This is equivalent to

$$\max_{Axis} \min_{i} (Axis \cdot Ray_i) \tag{1}$$

where Ray_i is the ith normalized ray vector. The problem can be rephrased as that of determining the smallest circle on the sphere that encloses the normalized ray vectors on the sphere. Computing a solution to this problem is a difficult task. Instead of computing it, we derive a box that has

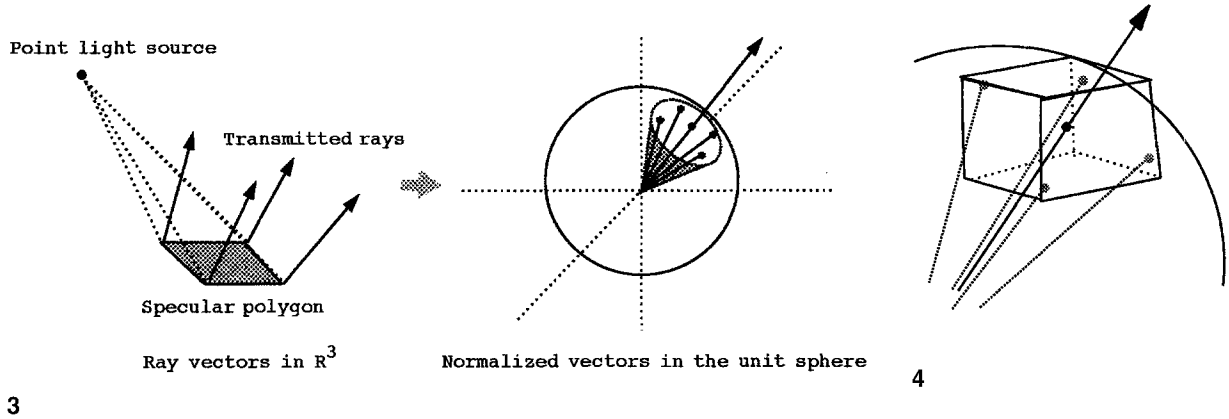


Fig. 3. Mapping normalized ray vectors to a unit sphere

Fig. 4. Approximating the axis of the enclosing cone

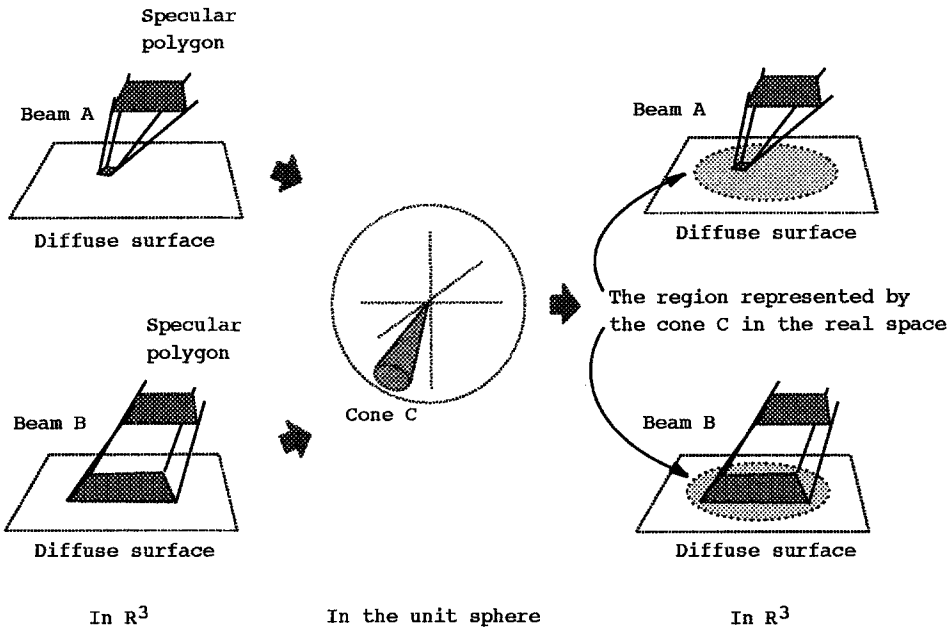


Fig. 5. Enclosing cones for convergent and divergent transmitted light beams

faces parallel to the coordinate axes and bounds the endpoints of the normalized ray vectors. We consider the center S of the bounding box as the center of the desired circle (Fig. 4). Then the axis of the enclosing cone is the vector from the sphere center to the point S, and the spread angle is computed accordingly. With the enclosing cone defined in the unit sphere, merging the cones and computing the cone-point interaction is easy. However, in Fig. 5, suppose the convergent light beam A and divergent light beam B have the same enclosing cone C in the unit sphere, and the enclosing cones of A and B represented by C in R^3 are identical, provided the two specular polygons are the same. In this case, computing the cone-point intersection for beam A would be much less efficient than that for B.

The double cone approach. With two overlaid cones, we can bound the transmitted light beam in R^3. The overlaid cones have the same axis and spread angle, but different apexes, as depicted in Fig. 6. The direction of the cone axis and the spread angle are computed using the single-cone approach. To locate the apex for each of the overlaid cones, we first derive the narrowest cross section of the beam, called the *neck of the beam*, and then locate the apex according to the spread angle (Fig. 6). The neck of the beam is located iteratively by sweeping a plane along the axis. The plane is parameterized by the variable t, with $t = 0$ for the plane passing one of the polygon vertices. Using the minimum spanning circle algorithm described in Toussaint (1985), we find the minimum spanning circle on each plane that encloses all ray-plane intersection points. The neck of the beam occurs on the plane with the smallest minimum spanning circle. The iteration begins with three initial plane positions (t_1, Dia_1), (t_2, Dia_2), and (t_3, Dia_3), where $t_1 < t_2 < t_3$ and $Dia_1 > Dia_2$ and $Dia_3 > Dia_2$, as shown in Fig. 7. With (t_1, Dia_1) and (t_3, Dia_3), the parameter for the new plane is linearly interpolated by

$$t_{new} = \frac{Dia_1 * t_3 + Dia_3 * t_1}{Dia_1 + Dia_3} \tag{2}$$

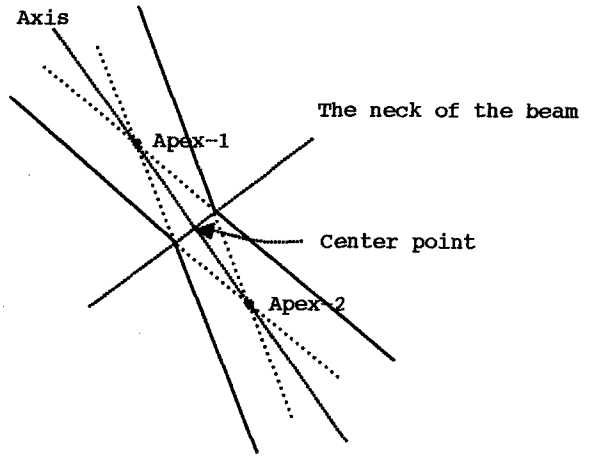


Fig. 6. Locating the apexes of the overlaid cones

On the new plane the minimum spanning circle is found with radius Dia_{new}. For the next iteration, we select two neighboring planes of the new plane as follows. If $t_{new} > t_2$ and $Dia_{new} < Dia_2$, we use the planes in the order of (t_2, Dia_2), (t_{new}, Dia_{new}), (t_3, Dia_3). If $t_{new} < t_2$ and $Dia_{new} > Dia_2$, we use (t_1, Dia_1), (t_2, Dia_2), (t_{new}, Dia_{new}). Having found the neck of the beam, as depicted in Fig. 8, we can compute the apexes of the overlaid cones by

$$Apex = C \pm D * \frac{r}{\tan(\alpha)} \tag{3}$$

where C and r are the center and the radius of the minimum spanning circle, α is the spread angle, and D is the direction of the cone axis.

3.1.2 Constructing the cone tree

Having constructed an enclosing cone or overlaid cones for each transmitted light beam, we organize the cones into a tree structure to ensure an efficient search. An internal node corresponds to a cone that encloses all its descendant cones. For example, Fig. 9 depicts a cone tree representing four cones enclosed by a parent cone. Collecting the descendant cones is straightforward if the specular surface is polygonized with a regular surface subdivision. For example, the cone tree can be a quadtree if the polygonization of the specular surface is obtained by standard domain subdivision. The problem becomes nontrivial, however, when the specular surface is itself an arbitrary polygon set.

A method similar to the one for constructing the cones is used to merge cones in the unit sphere. The normalized ray vectors are replaced here by the normalized direction vectors of the cone axis. The spread angle of the parent cone is obtained by

$$\max_i [Angle(Axis_{parent}, Axis_i) + SpreadAng_i] \tag{4}$$

where $SpreadAng_i$ is the spread angle of the descendant cone i. (Fig. 10).

For overlaid cones, the axis direction and the spread angle of the parent cone are also determined in a manner similar to that used to construct the overlaid cones. To locate the apexes, iterations similar to those used to locate the neck of the beam are applied. Since the ray is replaced by the cone, the sweeping plane intersects a cone in an ellipse. The major axis r_a and minor axis r_b of an ellipse can be found as follows:

$$r_a = \frac{z_0 \sin\delta \cos\delta}{\cos^2\delta - \sin^2\alpha}, \tag{5}$$

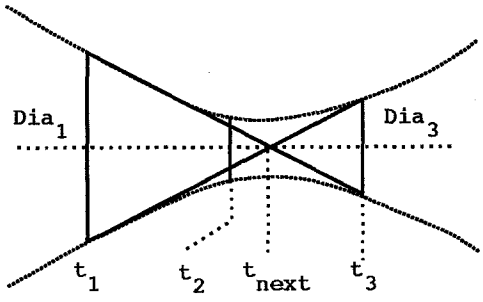
$$r_b = \frac{z_0 \sin\delta}{\sqrt{\cos^2\delta - \sin^2\alpha}} \tag{6}$$

where, as shown in Fig. 11, z_0 is the distance from the cone apex of the child cones to the plane, α is the angle between the axes of the parent cone and the child cone, and δ is the spread angle of the child cone. The direction of the major axis of the ellipse is given by

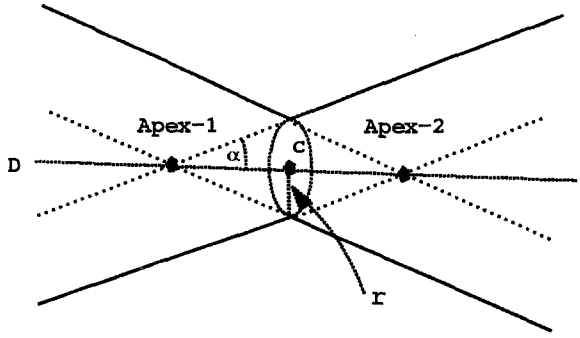
$$MajorAxis = Axis_{child}/\cos\alpha - Axis_{parent} \tag{7}$$

and the direction of the minor axis of the ellipse can be obtained by

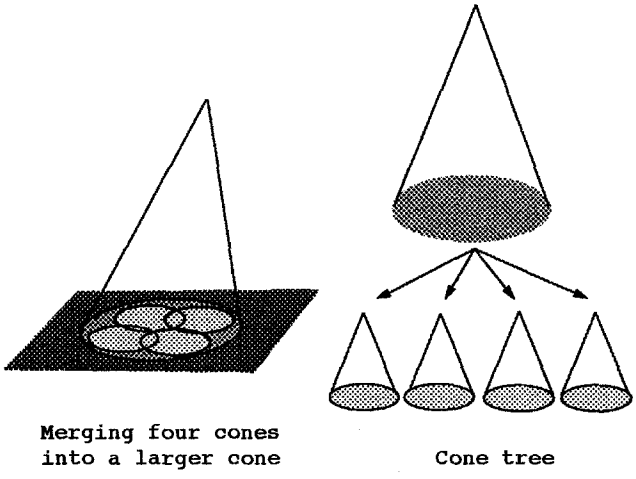
$$MinorAxis = MajorAxis \times Axis_{parent} \tag{8}$$



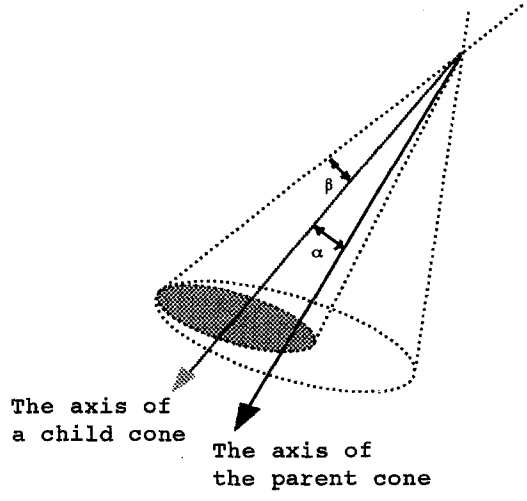
7



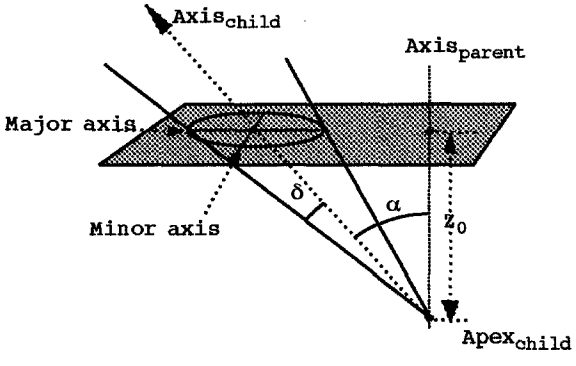
8



9



10



11

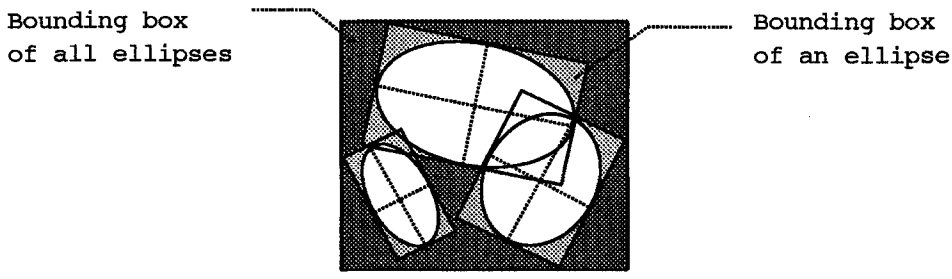
Fig. 7. Locating the new plane using linear interpolation

Fig. 8. Computing the apexes of the overlaid cones

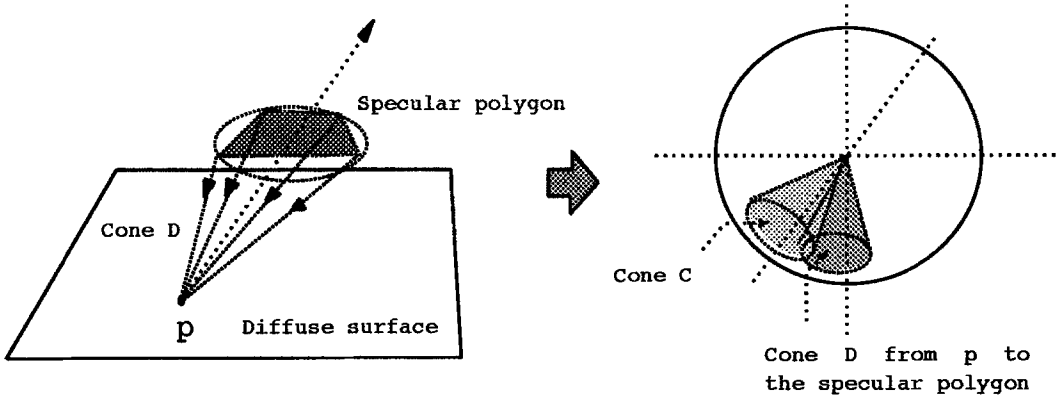
Fig. 9. Merging four cones into a larger one

Fig. 10. The spread angle of the parent cone

Fig. 11. Computing the ellipse of the plane-cone intersection



12



13

Fig. 12. The bounding box approach for ellipses

Fig. 13. Cone point intersection test for cones in the unit sphere

Both the *MajorAxis* and *MinorAxis* are normalized vectors. The center of the ellipse is represented by

$$Center = Apex_{child} + Axis_{child} * z_0 + MajorAxis * dy \tag{9}$$

where

$$dy = \frac{z_0 * \sin\alpha * \cos\alpha}{\cos^2\delta - \sin^2\alpha}. \tag{10}$$

It is, in general, difficult to find a circle that encloses all the ellipses. We approximate the computation by bounding each ellipse with a box with edges parallel to the major and minor axes of the ellipse. Then a box with edges parallel to the coordinate axes is found to enclose all the bounding boxes, as shown in Fig. 12. The enclosing circle is centered at the center of the bounding box and has a radius equal to half the length of the diagonal of the box.

We have described two ways of representing enclosing cones for the transmitted light beams and explained how the cones are merged. The single cone in the unit sphere can be constructed easily. However, the cone contains the beam only loosely, as shown in Fig. 5. Although the overlaid cones can enclose a beam more tightly, the rays of the beam are sometimes almost parallel. This case often occurs for leaf cones on the cone tree, and it makes the computation of apexes a difficult problem. In the implementation that follows, a cone tree that uses both types of enclosing cones is constructed. When the spread angle of the

enclosing cone is less than a prescribed value, say ten degrees, the single cone in the unit sphere is constructed. For cases in which the spread angle is larger than this value, the overlaid cones are computed.

3.2 Intensity computation

To render a point on a diffuse surface, we consider the diffuse component of the point's intensity as the total contribution of all transmitted light beams intersecting the point. The point is tested recursively against each cone tree, constructed for each point light source, in a top-down fashion. If the point is outside the cone of the current node then the point cannot be inside the descendant cones. If the point is inside the cone of the current node, the testing continues until the enclosing leaf nodes are reached. Each enclosing leaf cone is then checked to see if the point is inside the transmitted light beam enclosed by that cone.

3.2.1 Testing cone-point and beam-point intersections

For a single cone C represented on the unit sphere, to test whether a point p is contained in C, we construct a cone D on the unit sphere that encloses the normalized rays fired from the vertices of the specular polygon to p (Fig. 13). If C and D overlap, we conclude that p is inside C; otherwise, we conclude that p is not inside C. Note that cone D indicates the range of the viewing direction from every point on the specular polygon to p.

For the overlaid cones, the cone-point intersection is tested in R^3. Prior to the test, we identify whether the point is on the same side of the specular polygon as the light beam. If it is not, no cone-point intersection test is necessary. For the test, two rays are fired from the point to the two apexes of the overlaid cones. If any one of the angles between the rays and the axis is smaller than the spread angle, the point is inside the overlaid cones; otherwise, the point is outside the cones.

Once the diffuse point is found to be inside the enclosing cone, the point is checked to see if it is inside the corresponding transmitted light beam.

3.2.2 Computing intensity

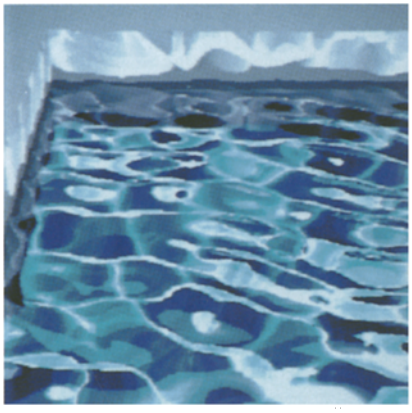
The intensity with which a light beam is propagated to a point can be computed using a technique similar to Watt's algorithm (Watt 1990), except that the normal of a testing point is used to form a projection plane on which the corresponding caustic polygon is constructed accordingly. The intensity at the caustic polygon, $I_{project}$, is computed as

$$I_{project} = E/Area_{project} = I * (N \cdot L) * Area_{specular}/Area_{project} \quad (11)$$

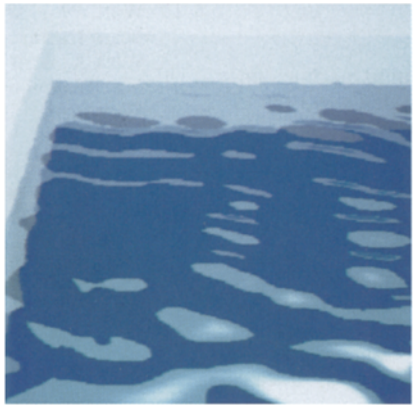
where $E = I * (N \cdot L) * Area_{specular}$ is the fraction of energy arriving on the specular polygon, I is the intensity of light incident on the specular polygon, N is the surface normal of the specular polygon, L is the direction of the light, $Area_{specular}$ is the area of the specular polygon, and $Area_{project}$ is the area of the caustic polygon. With Eq. 11, the caustic component for each pixel will be the same as that obtained by Watt's method. Since, in the proposed algorithm, the caustic effect is computed on a point-by-point basis, it is possible to improve the accuracy of the computation of the caustic effect by first locating points on the specular surfaces at which the light rays are transmitted or reflected to the diffuse point under consideration, and then accurately determining the energy transfer according to the local geometry of the specular points.

3.3 Implementation and examples

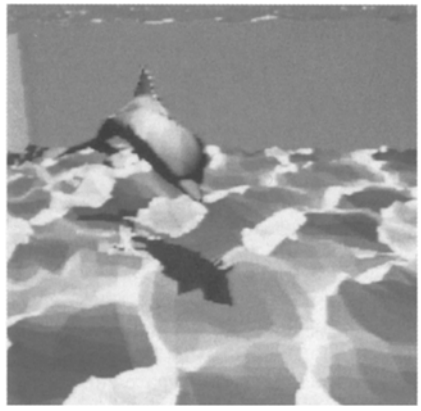
The proposed method was implemented on an IRIS 4D/25G, and several examples were tested. For the examples shown, only one point light source is considered. The cone trees are quadtrees with the same hierarchy as the subdivision of the specular surfaces. Figures 14 and 15 show two pools, one with caustic effects and the other without. The pools are rendered by ray tracing with a resolution of 100×100 pixels in height and width, while the water is approximated by 7575 specular polygons. The number of cone-point intersection tests for Fig. 14 was 1 293 973 and the number of beam-point intersection tests was



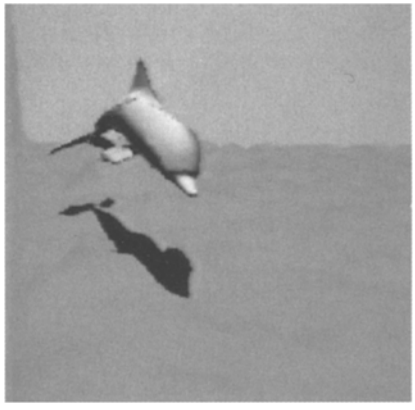
14



15



16



17

Fig. 14. A pool of water – with caustic effects

Fig. 15. A pool of water – without caustic effects

Fig. 16. A dolphin under the water – with caustic effects

Fig. 17. A dolphin under the water – without caustic effects

68 648, and the beam-point intersection test succeeded 12 960 times. Figures 16 and 17 show images of a dolphin under the water, one with caustic effects and the other without.

4 Conclusion

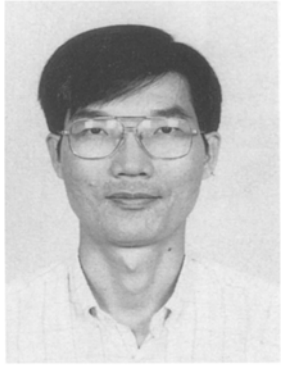
Caustic effects are, in general, difficult to compute, because in the rendering process the light transport from specular surfaces to diffuse surfaces must be simulated accurately and efficiently. We have proposed a two-pass algorithm that computes caustic effects by backward beam tracing. The algorithm has several advantages. Backward beam tracing for computing caustic effects requires a time complexity linear to the number of specular and diffuse polygons. In the proposed algorithm, the backward beams are not actually traced; instead, a tree structure is constructed from the enclosing cones of transmitted light beams. With the cone tree, the intensity computation can be speeded up by significantly reducing the number of beam-point intersection tests. Moreover, the proposed algorithm does not require the polygonal approximation of diffuse surfaces. Consequently, problems that arise in the ray tracing of polygonal scenes can be avoided (Snyder and Barr 1987).

Our future work will include deriving an efficient collection algorithm to collect sets of descendant cones for a specular surface that is itself an arbitrary polygon set. In addition, we are investigating ways of extending the present method to compute caustic effects that involve multiple specular reflections and transmissions.

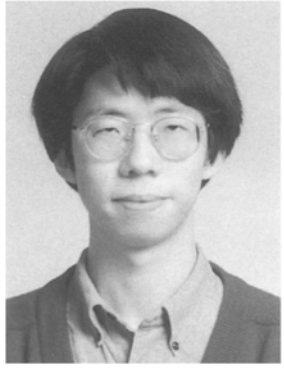
Acknowledgements. This work was supported by the National Science Council of the Republic of China under grant NSC 82-0408-E-009-428.

References

1. Appel A (1968) Some techniques for shading machine renderings of solids. Proceedings of American Federation of Information Processing Societies (AFIPS) JSCC, pp 37–45
2. Arvo J (1986) Backward ray tracing. SIGGRAPH Course Notes, No. 12
3. Arvo J, Kirk D (1987) Fast ray tracing by ray classification. Comput Graph 21:55–64
4. Campbell AT, Fussell DS (1990) Adaptive mesh generation for global diffuse illumination. Comput Graph 24:155–164
5. Chen SE, Rushmeier HE, Miller G, Turner D (1991) A progressive multi-pass method for global illumination. Comput Graph 25:165–174
6. Glassner AS (1984) Space subdivision for fast ray tracing. IEEE Comput Graph Appl 4:15–22
7. Hanrahan P (1983) Ray tracing algebraic surfaces. Comput Graph 17:83–90
8. Heckbert PS, Hanrahan P (1984) Beam tracing polygonal objects. Comput Graph 18:119–127
9. Kajiya JT (1982) Ray tracing parametric patches. Comput Graph 16:245–254
10. Kajiya JT (1983) New techniques for ray tracing procedurally defined objects. Comput Graph 17:91–102
11. Mitchell D, Hanrahan P (1992) Illumination from curved reflectors. Computer Graph 26:283–291
12. Shinya M, Takahashi T, Naito S (1987) Principles and applications of pencil tracing. Comput Graph 21:45–54
13. Sillion F, Puech C (1989) A general two-pass method intergrating specular and diffuse reflection. Comput Graph 23:335–344
14. Snyder JM, Barr AH (1987) Ray tracing complex models containing surface tessellations. Comput Graph 21:119–128
15. Toussaint G (1985) Computational Geometry. North-Holland, Amsterdam
16. Wallace JR, Cohen MF, Greenberg DP (1987) A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. Comput Graph 21:311–320
17. Watt M (1990) Light-water interaction using backward beam tracing. Comput Graph 24:377–385
18. Watt A, Watt M (1992) Advanced Animation and Rendering Techniques, Theory and Practice. Addison-Wesley, Wokingham UK



JUNG-HONG CHUANG is an Associate Professor of Computer Science and Information Engineering at National Chiao Tung University, Taiwan, ROC. His research interests include geometric and solid modeling, computer graphics, and visualization. Dr. Chuang received his BS degree in applied mathematics from National Chiao Tung University, Taiwan, in 1978, and MS and PhD degrees in Computer science from Purdue University in 1987 and 1990, respectively.



SHIH-ANN CHENG is currently an Associate Engineer at the Institute for Information Industry, Taiwan, ROC. His research interests include computer graphics, database, and computer network. Mr. Chen received his BS and MS degrees in Computer Science and Information Engineering from National Chiao Tung University in 1990 and 1992, respectively.