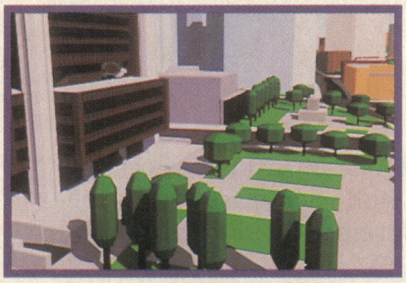
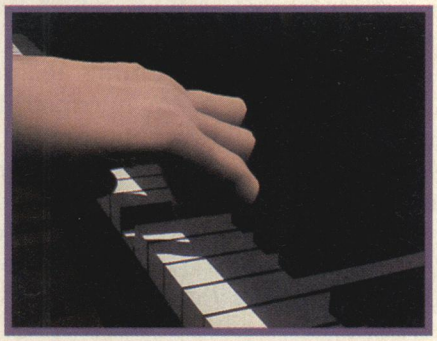
A General Version of Crow's Shadow Volumes

Philippe Bergeron, Digital Productions*





In 1977 Frank Crow introduced a new class of algorithm for the generation of shadows. His technique, based on the concept of shadow volumes, assumes a polygonal database and a constrained environment. For example, polyhedrons must be closed, and polygons must be planar. This article presents a new version of Crow's algorithm, developed at the Universite de Montreal, which attempts a less constrained environment.

The method has allowed the handling of both open and closed models and nonplanar polygons with the viewpoint anywhere, including any shadow volume. It does not, however, sacrifice the essential features of Crow's original version: penetration between polygons is allowed, and any number of light sources can be defined anywhere in 3D space, including the view volume and any shadow volume. The method has been used successfully in the film *Tony de Peltrie* and is easily incorporated into an existing scan-line, hidden-surface algorithm.

Shadows can be a problem in generating realistic images. They are visible from the viewpoint but invisible from the light source. Shadow casting vastly improves realism and gives useful information about relationships between objects, but the technique itself is extremely complex. It was for this reason that early implementations of shading algorithms eliminated the need for shadows, assuming that the light source was positioned at the observer's point of view. If more than one light source was used, the environment was generated as if it were a cloudy day with diffuse illumination.

These constraints were clearly unacceptable in some cases. Shadow casting was vital to applications that called for a realistic appearance, such as computer-animated sequences for flight simulation or for entertainment.

In 1977, Frank Crow documented existing shadow-generation methods and presented a technique for shadow casting that was based on shadow volumes.[1] He found that three classes of shadow generation methods were prevalent:

1. shadow computation during display
2. polygon shadow generation based on clipping transformation
3. shadow volumes

In 1978 Lance Williams introduced a fourth class:[2]

4. z-buffer shadow computation

and in 1980 Turner Whitted presented the fifth:[3]

5. generation of shadows using rays

*Digital Productions is a division of Omnibus Simulations. The work described in this article was done at Universite de Montreal in Canada.

0272-1716/86/0900-0017$01.00 © 1986 IEEE

Table 1. Classification of existing shadow algorithms.

Class No.	Description	Hidden-Surface Algorithm
1	Shadow computation during display	Scan-line, frame buffer
2	Polygon shadow generation with clipping transformation	Scan-line, frame buffer
3	Creation of shadow volumes	Scan-line, frame buffer
4	*z*-buffer shadow computation	Frame buffer
5	Shadows using rays	Ray tracing

Table 1 shows typical hidden-surface algorithms that can be associated with each class. Other specific conditions were addressed by Liu, Burton, and Campbell[4] and by Max.[5]

The technique described here, which we developed at the Universite de Montreal, is a general version of Crow's shadow-volume technique. It permits shadow casting in an unconstrained environment, while still allowing interpenetration of polygons and any number of light sources anywhere in 3D space—features of Crow's original version. It was used in the 35-mm, 3D animated short film *Tony de Peltrie*,[6] which was shown at the close of the SIGGRAPH 85 Film and Video Show.

Shadow-generation techniques

We will now look at the five classes of shadow-generation techniques in more detail. The general version of Crow's algorithm is categorized in Class 3.

Class 1

Shadow computation during display, the first method ever used, has been demonstrated independently by Appel[7] and Bouknight and Kelley.[8] Basically, shadow boundaries on polygons are detected by a scan-line algorithm while the image is displayed. The boundaries are formed by projecting potential shadow edges onto the polygon being scanned. Shadow edges are then projected onto the image plane. The color of a scan segment changes when it crosses the edge of a shadow.

Class 2

Polygon shadow generation involves two passes through an object-space hidden-surface algorithm. The first pass separates shadowed and unshadowed polygons. Polygons partially shadowed are divided by determining the hidden surfaces from the viewpoint of the light source. The colors of shadowed polygons are modified. The second pass processes the altered data from the normal viewpoint. The best known algorithm of this class is named Atherton-Weiler.[9]

Class 3

The third class of algorithm was introduced by Crow.[1] It consists of constructing shadow volumes, which can be defined as the invisible volume of space swept out by the shadow of an object. In the scan-line process the shadow polygons forming the volumes are treated as an invisible surface that, when pierced, causes a transition into or out of an object's shadow.

Shapiro Brotman and Badler developed a method for generating soft shadows by combining a *z*-buffer algorithm with shadow volumes.[10]

Class 4

The *z*-buffer shadow computation approach, introduced by Williams,[2] works with a *z*-buffer hidden-surface algorithm. A view from each light source is constructed, and the *z* values are stored. A view of the scene is then constructed from the observer's point of view. The *x*-*y*-*z* position of each point in the observer's view is then tested for visibility by transforming it to each light source. If the point is not visible to the light source, it is in shadow.

Class 5

The fifth class of shadow-generation techniques, presented by Whitted,[3] involves a ray-tracing hidden-surface algorithm. For each pixel a ray is created. The ray goes from the eye, passes through the pixel, and stops when a model is encountered. A new ray is directed toward the light source. If a new model is pierced before reaching the light source, the pixel is in shadow; otherwise it is not. This idea obviously extends to more than one light source, but it is extremely costly in terms of computation time.

Ray tracing is a very powerful shading technique that lends itself naturally to the casting of shadows (as well as to transparency and refraction). Any type of model can be used in ray tracing (polygons, patches, simple types, etc.).

For a polygonal database, the ray-tracing technique, like our general version of Crow's algorithm, allows any shape of model or polygon. However, displaying large polygonal databases with ray tracing takes too much time to be practical—which is why we developed a general shadow algorithm compatible with scan-line techniques.

The shadow volume approach

Crow describes the shadow volume approach this way:[1]

> The third class of shadow algorithm includes shadow volumes in the hidden-surface computation by adding their surfaces to the data. Assuming a polygonal object, the shadow surface is given by planes defined by contour edges and the light source position. Each such edge defines a polygon whose boundaries are the edge itself, the two lines defined by the light source position and the endpoints of the edge and the bounds of the field of view [see Figure 1]. The sense of the polygon must be maintained so that the near surface of a shadow volume (frontfacing polygons) may be distinguished from that far surface (backfacing polygons). Thus the polygons facing the light source plus the set of projected shadow polygons for an object define its shadow volume.
>
> Shadow polygons may be treated just like the rest of the data when applied to a scan-line, hidden-surface algorithm; only the shading for visible surfaces must be handled differently. Shadow polygons are themselves invisible, thus they do not count in the determination of visibility. However, the depth order of shadow surfaces and visible surfaces determines shadowing. A frontfacing shadow surface puts anything behind it in shadow while a backfacing shadow surface cancels the effect of a frontfacing one. For example, a post or column might cast a shadow surface consisting of a single polygon pair. Any surface lying between those two shadow polygons would be in shadow while surfaces lying in front of or behind both polygons would be shaded normally.

The algorithms in this class can be divided into two processes: the creation of shadow volumes (in object space), and the display of shadows in the scan-line process (in image space).

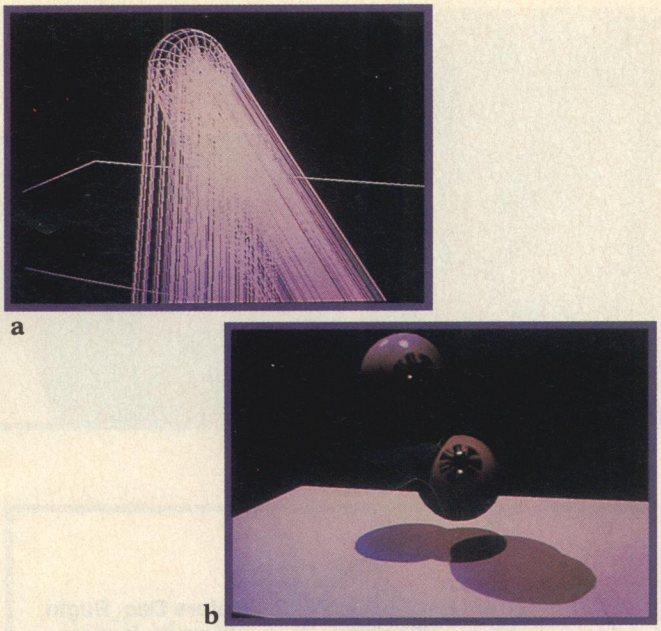


Figure 1. In (a) the normally invisible edges of the shadow polygons are projected from two spheres (the second sphere is obscured by the lines) lit by two light sources. In (b) you see the shaded images resulting from the models and shadow polygons of (a).

The environment

Let us describe the representation of models and the representation of lights in relation to the new version.

Representation of models

Most existing shadow algorithms based on a polygonal database assume a well-developed environment (convex polyhedrons or polygons, for example). Limitations vary from one algorithm to another. There is, however, one common restriction: Every algorithm (except ray tracing) assumes a planar polygonal database. If the original environment is not made of planar polygons, the user will have to subdivide each polygon into triangles before rendering.

Depending on the polygon, this process may be costly. It may also give results that are not artistically satisfying. Moreover, even if the original environment has been made planar to avoid subdivision, the user is forced to build models under severe and unnatural constraints. If the model is highly irregular, as in Figure 2, the polygons must be decomposed into triangular subsections to ensure planarity.

Our algorithm does not impose such limitations on the environment description. The data input consists of convex or concave, open or closed polyhedrons, with or without holes. Each polyhedron may be composed of convex or concave, planar or nonplanar polygons, with or without the holes.[11] Penetration between polygons is allowed. No subdivision is necessary at any stage during the process.

To use contour edges, the data structure provides links between adjacent polygons. An edge is shared by at most two polygons. A polygon observed from an outside view is defined clockwise.

Note that the shadow volume approach is incorporated into an existing scan-line algorithm: the Bouknight method.[12] Since this algorithm already handles the convex/concave, polygon/polyhedron cases, as well as penetration between polygons, we will not discuss those problems.

Representation of lights

Light can be modeled with either parallel or punctual light sources. Any number of sources can be defined, each of which may be anywhere in the

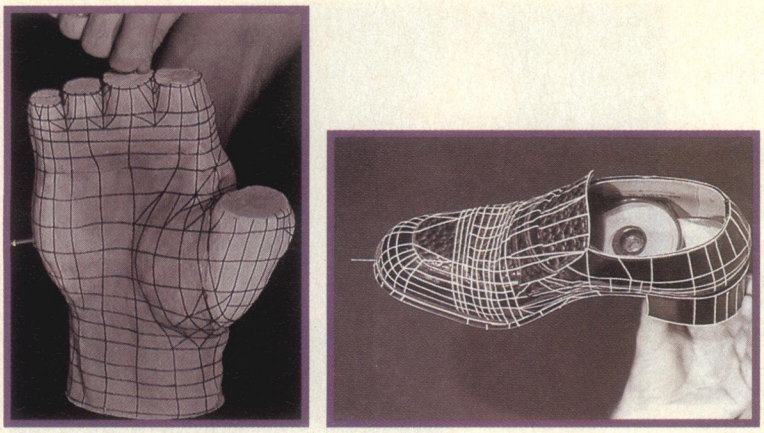


Figure 2. If the models are highly irregular, as these are, in the old version the polygons must be decomposed into triangular subsections to ensure planarity. In the new version this is not necessary.

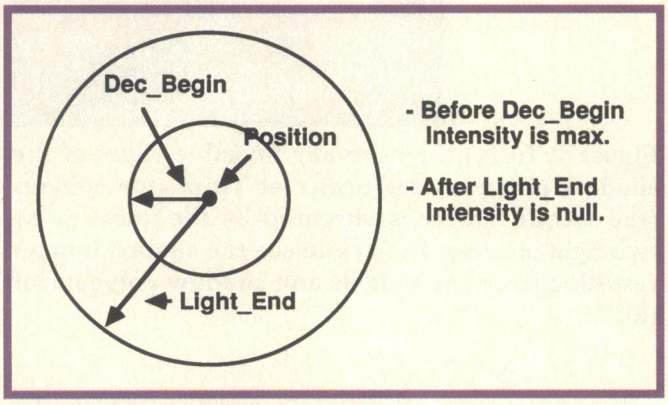


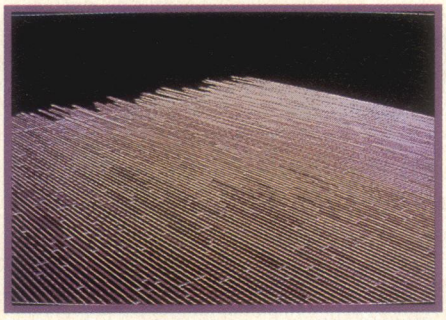
Figure 3. Description of a light.

environment space, including the view volume, and any shadow volume. A light is defined by five of what Cook calls *appearance parameters*[13] (see Figure 3):

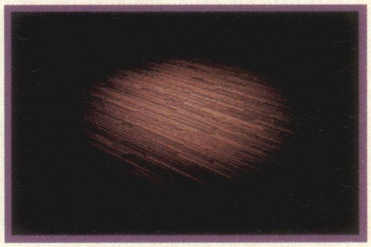
1. Position: An x-y-z point representing its position
2. Intensity: An RGB color representing its intensity
3. Dec_Begin: A real value representing the distance from Position where Intensity starts to decrease
4. Light_End: A real value representing the distance from Position where Intensity is null
5. Decrease: Exponent describing the decrease between Dec_Begin and Light_End (linear, exponential, etc.)

We suppose that the lights are not directional; that is, they distribute energy equally in all directions. The sphere defined by center Position and radius Light_End is referred to as the light's *sphere of influence*. Figure 4 is an example, a wooden floor illuminated by a limited light source.

The data structure of a light has more fields used for internal computations. These fields (see Figure 5) will be explained in more detail later.



a



b

Figure 4. In (a) you see a line drawing of a wooden floor. In (b) the floor is shaded, lit by a light source with a limited sphere of influence (there is no antialiasing).

The algorithm

Our general version of Crow's algorithm incorporates four basic methods:

1. a method for handling both topologically open and closed models by altering the depth count
2. a method for handling nonplanar polygons
3. a method for determining eye-point's initial shadow depth (when the eye is in shadow)
4. a limited extent for shadow polygons

By no means, however, does this technique overcome the shading anomalies in concave polygons or rotating models.[14]

Open models

To create the shadow volume, we take advantage of contour edges. An edge is a contour edge when *either* of two conditions is true:

1. It lies at the extremes of the surface for open polyhedrons.
2. It separates front-facing and back-facing polygons where the surface curves behind itself (the silhouette).

In fact, an edge is theoretically a contour edge only when it is visible. We will not consider this condition, since it is computationally expensive to find such information strictly in object space (where the contour edges are determined).

A light's *depth count* is a positive integer that represents the current depth value when shadow polygons are pierced. A front-facing polygon adds +1 to the depth count, while a back-facing shadow polygon adds -1, canceling the effect of a front-facing one. The shadow surface depth count going out of a model's shadow volume must be the same as when it was first pierced (for a single light source). This problem does not occur for closed polyhedrons, since they generate an equal number of front-facing and back-facing shadow polygons relative to the viewpoint.

Open polyhedrons, however, may not. Figure 6 is an example. Here the light source, *L*, is located near the reader's point of view, and the open polyhedron is generating three shadow polygons (SPs). For clarity we intentionally drop the four SPs generated by horizontal edges.

The depth counts before the first and after the last piercing are not the same. One solution is to introduce a new field in the shadow polygon's structure, called Nb_C. This value will be added to the current depth count each time a shadow polygon is pierced. If a shadow polygon is generated from an edge shared between two polygons (case 2 of contour edge), Nb_C will be +2 (-2 if back-facing). If the edge is not shared (case 1 of contour edge), Nb_C will be initialized to 1 (-1 if back-facing).

In our example Nb_C (SP_1) = +2, Nb_C (Sp_2) = -1, and Nb_C (SP_3) = -1. In fact the depth count will always be exactly twice that of Crow's if all polyhedrons are closed.

Nonplanar polygons

From the observer's point of view a nonplanar polygon may look perfectly normal, as in Figure 7a, but viewed from a particular light source, its edges may intersect and look like Figure 7b.

A polygon is "*L*-twisted" if its edges intersect when viewed from light *L*. A nonplanar polygon may be *L*-twisted relative to a light source *L* but normal to some other light source.

The following straightforward method is used to decide if a polygon is twisted relative to *L*:

```
(* For the user. *)
Position  : Vector;
Intensity : Color;
Dec_Begin: Real;
Light_End : Real;
Decrease  : Integer;

(* For internal use. *)
Depth_Count       : Integer;
Depth_Count_Start : Integer;
Next              : Light;
```

Figure 5. Data structure of a light.

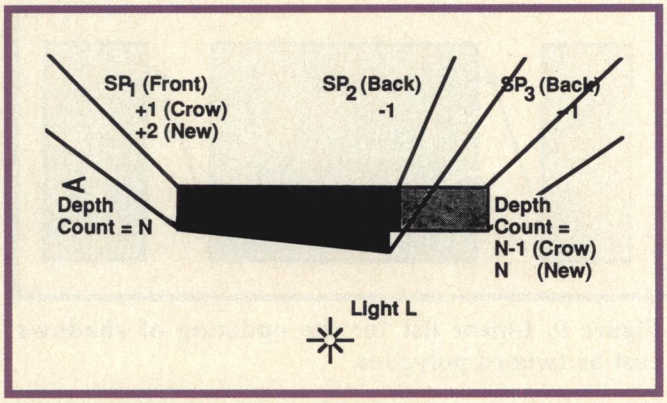


Figure 6. Open polyhedrons are a problem with Crow's original algorithm.

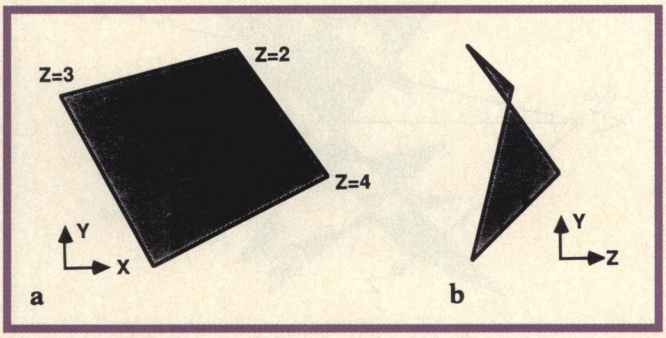


Figure 7. In (a) a nonplanar polygon may look perfectly normal, but when it is viewed from a certain angle (b), it is twisted.

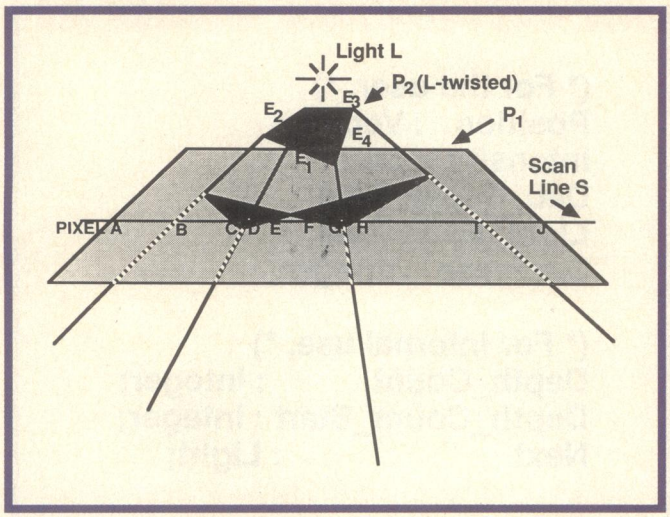


Figure 8. P_1 is the floor and P_2 is an L-twisted polygon floating in the air.

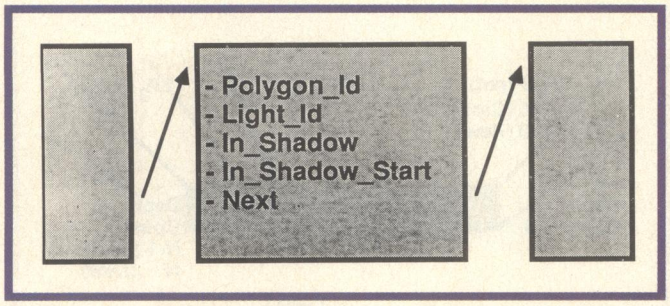


Figure 9. Linear list for the updating of shadows cast by twisted polygons.

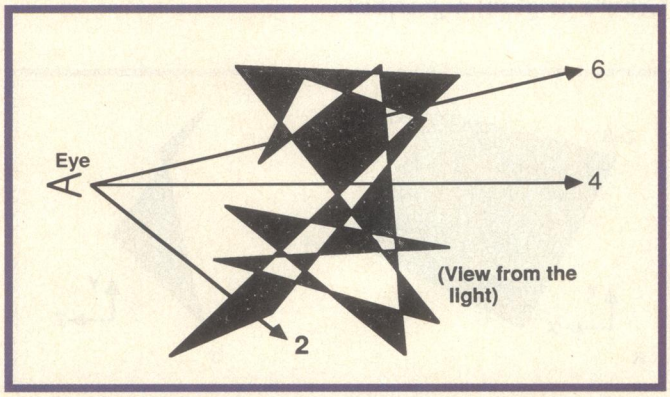


Figure 10. Reader is located at the light source position. The total number of pierced polygons to pass through the volume is always even.

1. A 2D projection of the polygon relative to L is computed.
2. Each pair of nonadjacent edges is compared to find if they intersect. This simple 2D computation has proven efficient, since most of the polygons have three edges (no comparison), four edges (two comparisons at most), or five edges (five comparisons at most).

Imagine a scene with two quadrilateral polygons P_1 and P_2. Let P_1 be the "floor" and P_2 a nonplanar polygon floating in the air. The edges of P_2 are denoted by E_1 to E_4. The light source L is up in the air, and P_2 is L-twisted. Figure 8 illustrates.

According to the definition of Case 1 of contour edges (described in earlier in the "Open models" subsection), each edge of P_2 is a contour edge. Thus, P_2 generates four shadow polygons, SP_1 to SP_4. Because the sense of the shadow polygons is the same as that of their parent edge, we can say that SP_1 and SP_3 are opposites relative to the viewpoint. That is, if SP_1 is back-facing the viewpoint, then SP_2 is front-facing, SP_3 is front-facing, and SP_4 is back-facing.

During the display of scan line S, no problem occurs before reaching pixel F, since from A to C and E to F there are no shadows when P_1 is first encountered (through the z-search), and from C to E a front-facing polygon is pierced (SP_2 before pixel D, SP_3 after), causing a transition into shadows.

Once pixel F is reached, however, and the z-list is sorted again (SP_1 is now in front of SP_3), we enter the back-facing polygon SP_1 first, introducing an error. This situation occurs only with shadow polygons generated from twisted polygons. It can be corrected by keeping a shadow flag In_Shadow for each distinct (L,P) pair, where polygon P is L-twisted. The implication is that front-facing/back-facing information Nb_C is useless for shadow polygons generated from an L-twisted polygon. The penetration of a shadow polygon simply changes the flag and updates the depth count value L.Depth_Count by ± 1, since the shadow polygons are generated from open edges.

Assume that In_Shadow is false for L-twisted P_2, and L.Depth_Count = 0. Between pixels F and G, SP_1 is first pierced, setting In_Shadow to true, and adding 1 to L.Depth_Count (–1 if In_Shadow were changing from true to false). The identifications of the light source and the parent polygon are kept in the shadow polygon structure. We can concatenate the two IDs to form a key and access the flag in an independent data structure. Our independent data structure is a linear list accessed sequentially (Figure 9). (The In_Shadow_Start flag used for the eye's initial shadow depth will be explained later.)

This method works for any type of polygon. It even works for n-sided polygons, which can be exceedingly L-twisted. As long as the polygon is closed, the total number of piercings after passing

through the entire shadow volume is always even (Figure 10).

The shadows will clearly be consistent with the shape of the polygon. A highly nonplanar octagon, for example, will cast a shadow as weird as itself. Our experience with *Tony de Peltrie* has shown us that, even though our method is a general algorithm, the polygons should seldom be defined with more than six edges, if the model is to be animated and twisted.

Twisted polygons in a polyhedron

Imagine a situation in which edge E_1 is shared between two polygons P_1 and P_2, the latter being L-twisted (see Figure 11). Is E_1 a contour edge?

The answer is not clear because we cannot determine if P_2 is front-facing the light source. We do know, however, that P_1 is front-facing the light. If we assume that P_2 is front-facing (or back-facing) the light, then E_1 (or E_2) is not a contour edge. The correct shadow is not cast in either situation, as Figure 12 shows.

The solution to this problem is to treat each L-twisted polygon independently by constructing its own shadow volume (as seen earlier), and to process the rest of the model (now open) independently. Thus, if an edge is shared between two polygons P_1 and P_2, and at least one of them is L-twisted, it is actually two edges—Case 1 of contour edges (both edges not shared).

The eye in shadow

A special case is when the eye is in shadow. If the frontmost shadow polygon is back-facing, we can conclude that the eye is in shadow. Unfortunately the reciprocal is not true. We cannot conclude that the eye is not in shadow if the first polygon is front-facing (see Figure 13).

The eye-point-in-shadow procedure presented in this section is general and complete. In fact, the eye in shadow can be detected before the scan-line process. The first step is to close the shadow volume, which is made up of three types of polygons:

1. the model's polygons (first end of the volume)
2. shadow polygons generated from the light source (sides of the volume)
3. shadow polygons created by a scaled and translated capping model projected to the extremes of the volume (second end of the volume)

It is the closing of the shadow volume by the original model and the projected model's polygons that ensures the completeness and the generality of this solution to the eye-point-in-shadow problem.

As we leave the eye, we examine each polygon (whatever the type) pierced by a ray from the eye through an arbitrary pixel, say pixel 1 of scan line 1. Along the way, we keep track of each L.Depth_Count, and of the shadow flags of each distinct (L,P) pair, where P is L-twisted. After the process we

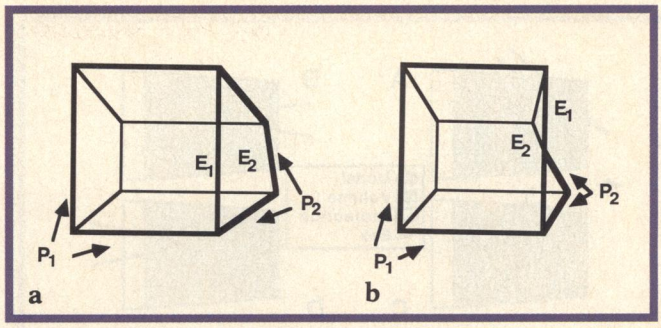


Figure 11. Two views of the same model: (a) from the eye, and (b) from the light L. P_2 is L-twisted.

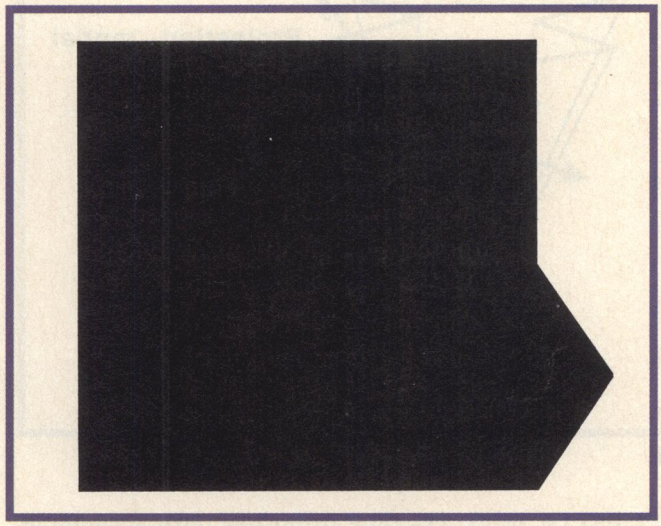


Figure 12. Exact shadow cast by the model.

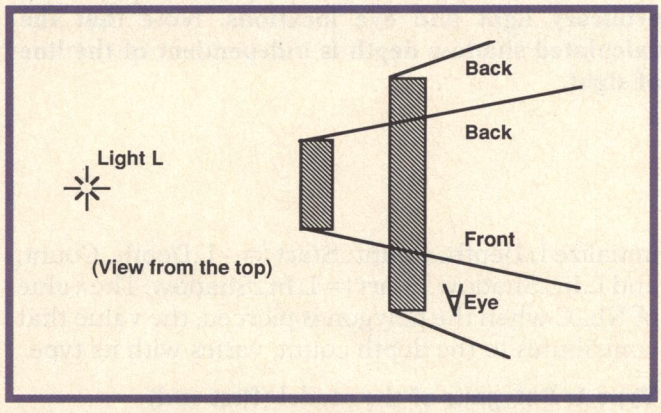
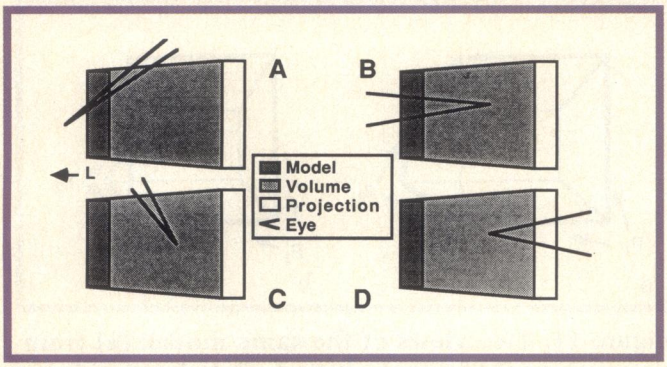
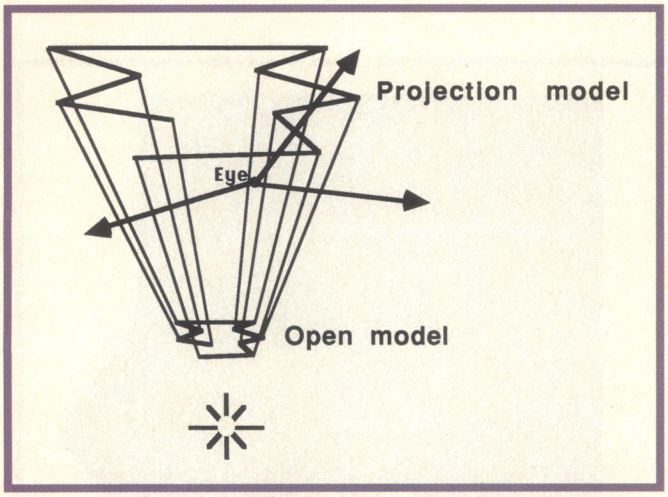


Figure 13. We cannot conclude that the eye is not in shadow if the first polygon is front-facing.



a



b

Figure 14. In (a) are four situations with associated L.Depth_Count_Start values (clockwise from top left) 0 = -(+1+1-2); 2 = -(-1-1); 2 = -(-2); and 2 = -(-1-1). In (b) is a more complex model with arbitrary light and eye locations. Note that the calculated shadow depth is independent of the line of sight.

initialize L.Depth_Count_Start := - L.Depth_Count, and L.In_Shadow_Start := L.In_Shadow. The value of Nb_C when the polygon is pierced, the value that contributes to the depth count, varies with its type.

Type 1: Polygons of the model (first end)

```
if polygon facing the light
  if polygon facing the eye
    Nb_C := 1
  else
    Nb_C := -1
else
  if polygon facing the eye
    Nb_C := -1
  else
    Nb_C := 1
```

Type 2: Shadow polygons forming the sides. These shadow polygons are generated from the light source, so Nb_C is independent of their orientation to the light.

```
If polygon generated from edge of type 1 (not
shared)
  if polygon facing the eye
    Nb_C := 1
  else
    Nb_C := -1
else
  if polygon facing the eye
    Nb_C := 2
  else
    Nb_C := -2
```

Type 3: Polygons of the projected model (second end). The Nb_C value of a projected model polygon varies with its orientation, but it is the inverse of its physical correspondent (Type 1):

```
if polygon facing the light
  if polygon facing the eye
    Nb_C := -1
  else
    Nb_C := 1
else
  if polygon facing the eye
    Nb_C := 1
  else
    Nb_C := -1
```

Figure 14a shows four situations with their associated Depth_Count_Start values. Figure 14b shows a more complex model with arbitrary light and eye locations.

Closing the shadow volume permits us to handle every eye-point-in-shadow situation. The added polygons do not dramatically increase the memory in image space, since most of these polygons will not be scan-converted. Moreover, when the user knows that the eye is not in shadow, he may use an option that considerably reduces the number of shadow polygons in object space: In other words, Type 3's are useless. In a convex model, the volume can be closed without using the back-facing (relative to *L*) polygons in Types 1 and 3.

Sphere of influence of a light

The intensity of a light is null past its sphere of influence. That is, the colors of models outside a sphere of influence are not altered by the intensity of the light, and naturally the light never casts shadows on such models. Thus, to optimize memory allocation, we are truncating shadow volumes outside the sphere. Of course, truncating the shadow volume is of little importance if the scene is enclosed in the light's sphere of influence. Rendering a city

with shadows at night, for example, would be extremely costly without such an optimization. Figure 15 shows an example with models M_1 and M_2, and light L.

The figure shows that M_2 is not altered by L, so the shadow volume of M_1 need not include M_2. Of course M_2 could be included in the shadow volume of M_1 and vice versa for some other light sources.

Theoretically the shadow volume should be curved—a costly process for polygons. One way to overcome this cost is to create a simple shadow volume that includes the theoretical shadow volume (see Figure 16).

Here the key to the problem is to ensure that the shadow projection of the closest point of model M to L (not necessarily a vertex) is lying on the sphere, or further. The projection formula is as follows:

For each vertex V of M

$$\text{Projection} := V + \left(\frac{\text{L.Light\_End}}{\text{SD}} - 1\right) * (V - \text{L.Position})$$

where SD = smallest distance between M and L.Position

It is expensive to compute the exact SD value for a nonplanar polygonal model. By experience, we have found that approximating SD by calculating the smallest distance between the model's bounding box and L.Position is valuable.

Displaying shadows in image space

The display of shadows is processed by the Bouknight scan-line algorithm.[12] Basically the Bouknight approach consists of a y bucket sort on the edges. A list of active edges, based on scan-line coherence, is kept in memory. For each scan line, an x sort (usually a simple bubble or insertion sort) is run on the active edge list. A z-sorted list of active polygons is updated while the scan line is being processed. The first polygon is the visible one. The list does not have to be updated at each pixel if we capitalize on point-to-point coherence.

Shadow polygons may be treated just like physical polygons through the three sorts. The z list is searched to find the visible polygon (the first non-shadow polygon). Each time a shadow polygon is pierced, the shadow state of its parent light is updated. Nb_C is added to the current light depth count. If the parent polygon P is L-twisted, the flag is changed and if the shadow has been exited, a -1 is added to the depth count of L; otherwise, a $+1$ is added. When the visible polygon is displayed, the shading model takes into account only those lights having a null depth count.

The priority list of shadow polygons is searched only when the visible polygon changes. If a shadow polygon is inserted (or deleted) behind the visible polygon in the z list, no update is done. If the insertion (or deletion) is in front of the visible polygon, only the parent light depth count of that polygon needs to be updated.

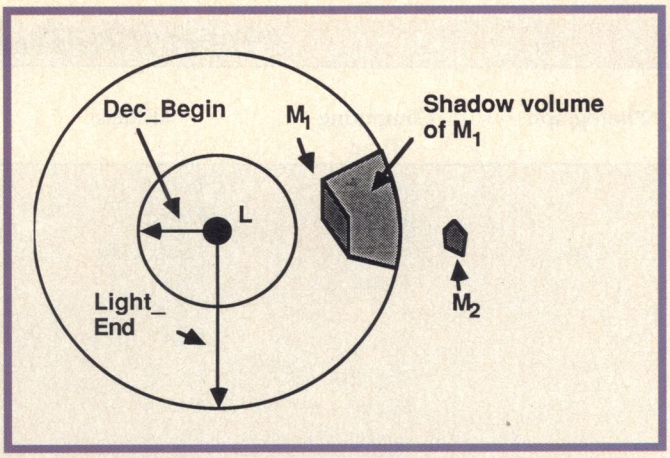


Figure 15. Model M_1 is inside the light's sphere of influence. M_2 is outside the sphere.

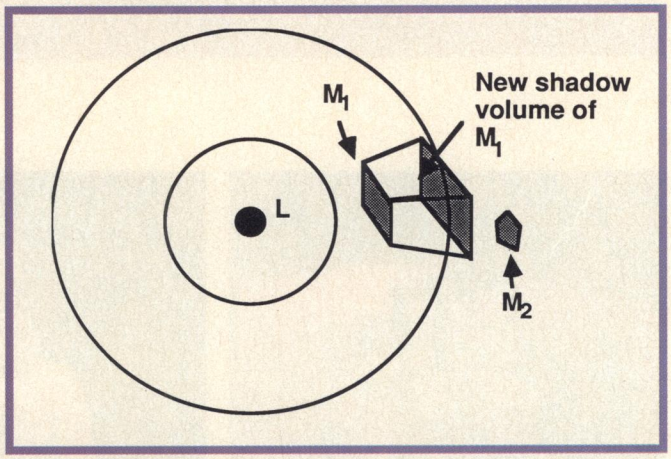


Figure 16. A simpler shadow volume that does not create any shadow artifacts.

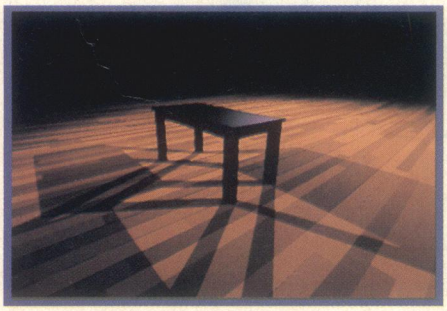
Experimentation

A version of this algorithm has been implemented and tested in Pascal and included in the DADS graphics package (Digital Animation Display Software) designed by Pierre Lachapelle and developed and programmed by Lachapelle and myself. It runs under NOS/BE on a CDC Cyber 855. Although general-purpose, DADS is used mainly for shaded images for art and entertainment.

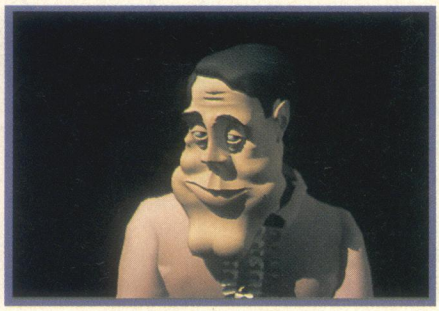
Table 2 lists some experiments using the new shadow algorithm. Corresponding images are

Table 2. Description of each photograph.

Photograph	Computing Time	Models	Planar Polygons	No. Scan-Converted Polygons	No. Scan-Converted Shadow Polygons
17a	6 min.	Bench	Yes	34	246
	22 sec.	Floor	Yes	663	—
17b	6 min.	Flesh/Hair	No	2343	1054
	11 sec.	Eyes	No	679	—
		Teeth	No	24	—
		Shirt	No	1165	—
17c	8 min.	Floor	Yes	14	—
	17 sec.	Shoes	No	796	993
		Socks	No	287	256
		Pants	No	44	622
17d	21 min.	Buildings	Yes	2715	8153
	2 sec.	Sidewalks (1)	Yes	377	—
		Sidewalks (2)	Yes	1520	1589
		Ground	No	4242	—

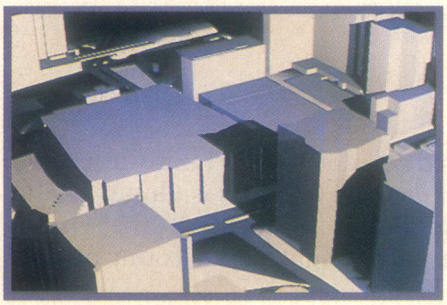


a



b

c



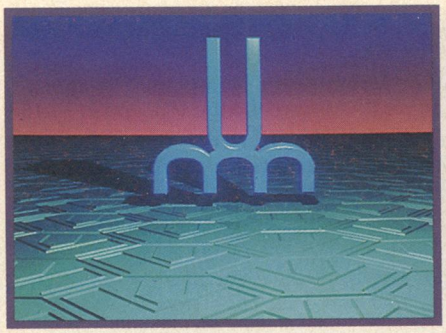
d

Figure 17. In (a) the bench was done by Philippe Bergeron, Pierre Lachapelle, and Pierre Robidoux. In (b) *Tony de Peltrie* was done by Lachapelle, Bergeron, Robidoux, and Daniel Langlois. In (c) the shoes were created by Philippe Bergeron. In (d) downtown Montreal was created by Bernard Guenette.

shown in Figure 17. Note that because we do not know in advance if the eye is in shadow, the algorithm closes the shadow volume (see earlier section "The eye in shadow").

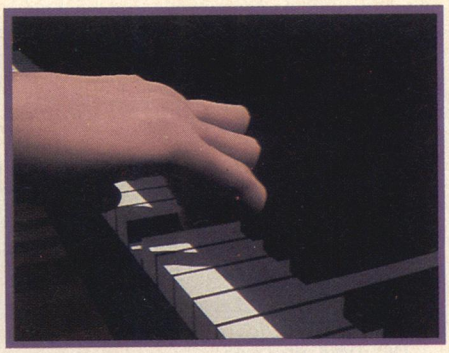
Computation times are from implementation on a Cyber 855. The dashes under "Nb of Scan-Converted Shadow Polygons" indicate that the model does not generate shadows. Computing resolution of each image is 2048 × 1536. Because of the contrast between the models' colors and the absence of gamma correction, some pictures may look as though they were were computed at the 512 × 384 screen resolution, but they were not.

Figure 18 presents some additional results.

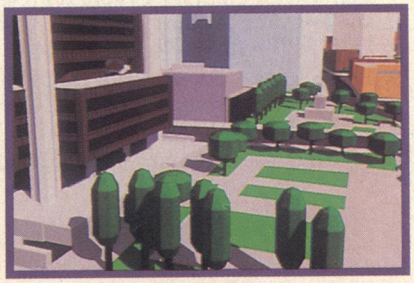


a

b



c



d

Figure 18. In (a) the logo of Universite de Montreal was created by Pierre Lachapelle. In (b) *Tony de Peltrie* playing the piano was done by Lachapelle, Philippe Bergeron, Pierre Robidoux, and Daniel Langlois. In (c) Tony's hand was created by Lachapelle, Bergeron, Robidoux, and Langlois. In (d) downtown Montreal was created by Bernard Guenette.

Conclusion

A general version of Crow's shadow algorithm based on the creation of projected shadow volumes has been presented. It can easily be incorporated into an existing scan-line hidden-surface algorithm of the Bouknight type. The input consists of a polygonal database. There are no limitations imposed on the environment.

Even if the data structure of a shadow polygon is much smaller than that of a physical polygon (no normal, no color), the bottleneck of the algorithm is memory. We have found that a Cyber 855 (very limited for main memory) is not an ideal environment for implementing the algorithm. We believe our method will work best in a virtual memory environment.

The algorithm leaves considerable room for improvement. In particular, more work is needed to develop shadow casting techniques in scan-line hidden-surface algorithms involving nonpolygonal models (fractals,[15] patches,[16] particle systems,[17] among others). Also, research should be focused on casting shadows from a computer-generated object onto a complex, live-action environment. ■

Acknowledgments

I am extremely grateful to Pierre Lachapelle for his useful suggestions and helpful discussion at every stage of the research. His advice made it possible to implement and test the algorithm in the DADS graphics package, developed at the Centre de Calcul of the Universite de Montreal. I also thank Gilles Brassard for his useful contribution to this article, and Claude Crepeau for helping us print the graphics. Several reviewers also provided valuable input. Finally, several people have provided us with computer-generated pictures that illustrate the new version of the algorithm. I am particularly grateful to Bernard Guenette, whose representation of downtown Montreal was part of his master's thesis.

References

1. F. Crow, "Shadow Algorithms for Computer Graphics," *Computer Graphics*, (Proc. SIGGRAPH 77) Vol. 11, No. 3, July 1977, pp.242-248.
2. L. Williams, "Casting Curved Shadows on Curved Surfaces," *Computer Graphics*, (Proc. SIGGRAPH 78), Vol. 12, No. 3, July 1978, pp.270-274.

3. T. Whitted, "An Improved Illumination Model for Shaded Display," *Comm. ACM*, Vol. 23, No. 6, June 1980, pp.343-349.
4. M.C. Liu, R. Burton, and D. Campbell, "A Shadow Algorithm for Hyperspace," *Computer Graphics World*, July 1984, pp.51-59.
5. N. Max, "AtomLLL with Transparency and Shadows," *Proc. 16th Hawaii Int'l Conf. Systems Sciences*, Vol. 2, 1983, pp. 407-418.
6. P. Lachapelle, P. Bergeron, P. Robidoux, and D. Langlois, *Tony de Peltrie*, 1985 (35-mm film).
7. A. Appel, "Some Techniques for Shading Machine Renderings of Solids," *Proc. SJCC 1968*, Thompson Books, Washington, DC, 1968, pp.37-45.
8. W.J. Bouknight and K. Kelley, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *AFIPS Conf. Proc.*, Vol. 36, 1970, AFIPS Press, Reston, Va., pp.1-10.
9. P. Atherton, K. Weiler, and D. Greenberg, "Polygon Shadow Generation," *Computer Graphics* (Proc. SIGGRAPH 78), Vol. 12, No. 3, July 1978, pp.275-281.
10. L. Shapiro Brotman and N. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp.5-12.
11. P. Bergeron, "Shadow Volumes for Non-Planar Polygons," *Proc. Graphics Interface 85*, Montreal, May 1985, pp.417-418.
12. W.J. Bouknight, "A Procedure for the Generation of 3D Half-Tone Computer Graphics Presentations," *Comm. ACM*, Vol. 13, No. 9, Sept. 1970, pp.527-536.
13. R. Cook, "Shade Trees," *Computer Graphics*, (Proc. SIGGRAPH 84), Vol. 18, No. 3, July 1984, pp.223-231.
14. W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, Hightstown, N.J., 1979, p.402.
15. L. Carpenter, "Computer Rendering of Fractal Curves and Surfaces," *Computer Graphics*, (Proc. SIGGRAPH 80), Vol. 14, No. 3, July 1980, p.109.
16. J. Lane, L. Carpenter, T. Whitted, and J. Blinn, "Scan-Line Methods for Displaying Parametrically Defined Surfaces," *Comm. ACM*, Vol. 23, No. 1, Jan. 1980, pp.23-24.
17. W. Reeves, "Particle System—A Technique for Modeling a Class of Fuzzy Objects," *Computer Graphics* (Proc. SIGGRAPH 83), Vol. 17, No. 3, July 1983, pp.359-376.

Philippe Bergeron is a research technical director at Digital Productions in Los Angeles, a division of Omnibus Simulations. From 1980 to 1981 he worked part time as a graphics programmer at the National Film Board of Canada. In 1981 he received the Best Technical Contribution Award at Eurographics 81 for a paper he coauthored. From 1982 to 1985 he codirected two storytelling computer-generated animated shorts: *Vol de Reve/Dream Flight* and *Tony de Peltrie* (about a has-been piano player), which closed the SIGGRAPH 85 Film and Video Show. The latter, winner of several international awards, has been hailed as a breakthrough by *Time* magazine. His interests include the production of storytelling, computer-generated films featuring believable actors that transmit emotions and feelings.

Bergeron received a BSc and an MSc in computer science from the Universite de Montreal. He is a member of ACM.

His address is 6140 Canterbury Dr. #312, Culver City, CA 90230.