SHADOW ALGORITHMS

Computer Science Seminar

Institute of Computer-Graphics
Vienna University of Technology

Gerald Matzka
9526002
e9526002@student.tuwien.ac.at

Introduction

Why Shadow ?

In former days, computer graphics was only known as flat, two-dimensional graphical representation. After three-dimensional principles were introduced into computer graphics, objects were often rendered without shadows and do not appear to be anchored in the environment. Shadows convey a large amount of information because they provide what is essentially a second view of an object.

On the left side the teapot appears to float above the plane, whereas on the right side it is anchored in the environment (see Figure 1 and Figure 2):

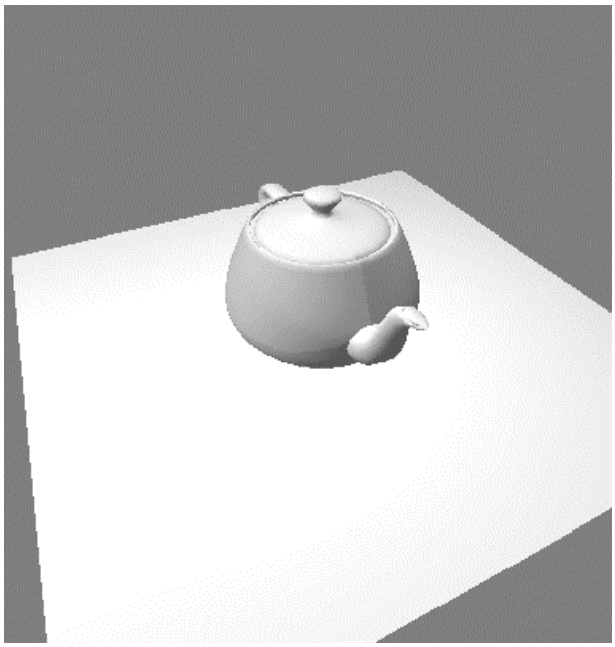

Figure 1 Scene without shadow

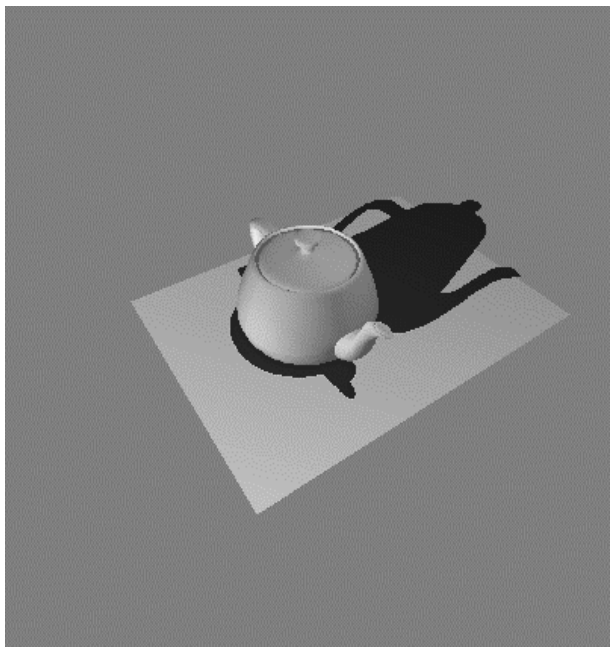

Figure 2 Scene with shadow

What is shadow ?

A shadow is defined as "an area that is only partially irradiated or illuminated due to blockage of light by an opaque object", or "the rough image cast by an object blocking rays of light" (see also [1]).

So in some short words, a shadow is a dark area caused by light being blocked by an object.

Anatomy of a Shadow

A point is in shadow relative to a given light source if rays from that light source cannot directly reach the point. Stated another way: shadowed points are points that cannot see the light (see also [2]).

Point light sources produce shadows with "hard" edges. Non-point light sources produce both umbra and penumbra shadows.

Umbra vs. Penumbra

The shadow behind an object lit by a light source (in contrast to a point light source) does not have sharp boundaries. This is caused by the fact that each point in the boundary area is only partially shadowed. The volume in full shadow is the **umbra**, the boundary area the **penumbra** (see also Figure 3).

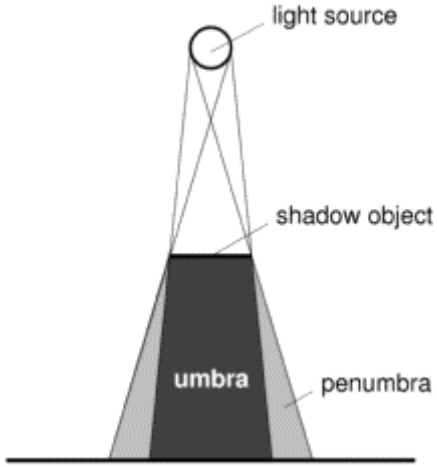

Figure 3 Light source producing umbra and penumbra

The figure is actually inaccurate, since the penumbra does not have a constant shadow density, but rather a gradient ranging from shadow to light (see also [4]).

Shadow Volume

For each shadow-generating polygon facing the light source, shadow planes (see [8] and [8]) are formed by each edge of the polygon. Each shadow polygon is a semi-infinite quadrilateral with two finite vertices corresponding to each pair of edge endpoints and two infinite vertices. The infinite vertices are placed at the limit of rays emanating from the light source and passing through each finite vertex. The volume of space enclosed by the shadowing object and the shadow planes is the shadow volume (see also Figure 4). The shadow volume itself is infinite but mostly clipped to the viewing area.

Those objects or parts of objects inside this shadow volume are in shadow, whereas those outside are lit.

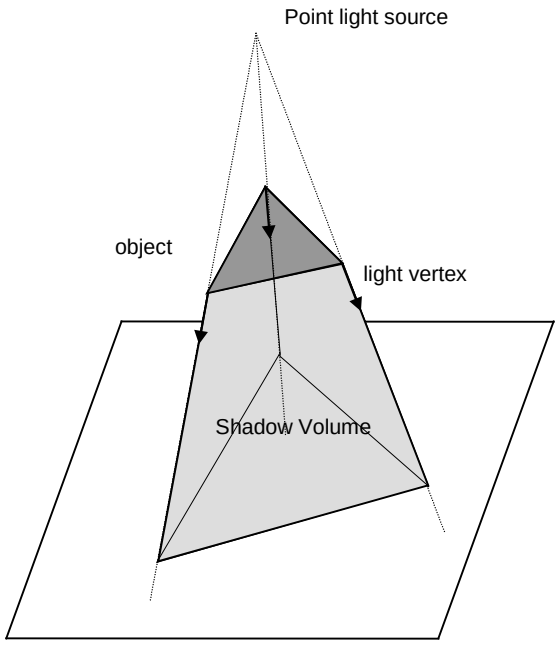

Figure 4 Definition of Shadow Volume

Shadow Algorithms

Existing shadow algorithms can be divided into four broad categories: ray-casting, projection, shadow volumes, and shadow maps. Of these, only the last three are (currently) applicable to real-time rendering. There are also several hybrid algorithms that combine features from different categories.

In this paper the basics of the most common used and applicable algorithms are described.

1. Fake Shadow

One of the simplest way to create a shadow is to add below the object a polygon which represents the shadow. The position and shape of this polygon is not exactly calculated, rather it is a rough approximation of where the shadow of an object might be. A common approximation is to estimate the position of the shadow polygon by using the center of the object and the position of the light source. See Figure 5 for a representation (see also [1] and [5]).

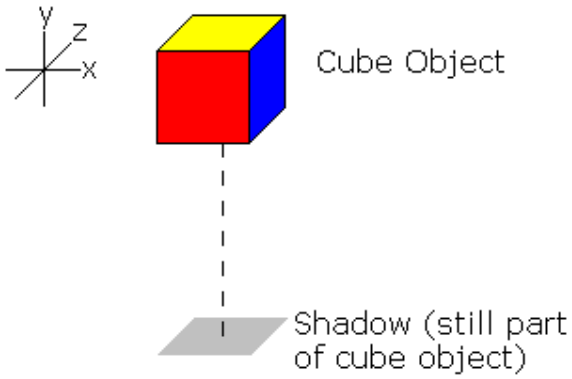

Figure 5 Fake Shadow

This is a very simple way of doing shadow. Unfortunately, it is very limited because the ground must be flat, the shadow is estimated by a polygon which looks like the object that generates the shadow. In addition the object can only be rotated around the y-axis and it does not shadow anything other than the floor.

2. Vertex Projection

This method is still very simple, but it is considerably better than the last one. In this method, each polygon of an object is projected onto a flat ground (see also [1] and [2]).

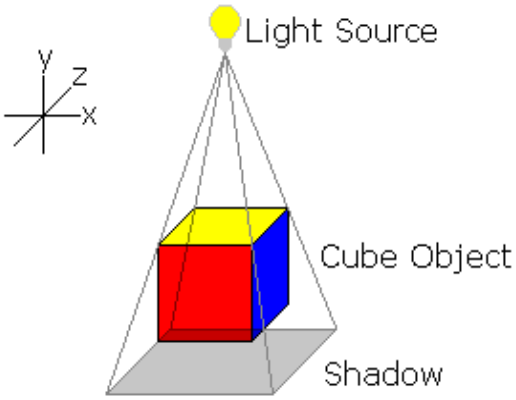

Figure 6 Vertex Projection

The calculations for this method can be summarized in the following matrix:

$$M_{shadow} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -x_I/z_I & -y_I/z_I & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Light position (x_I, y_I, z_i)

Now, given the co-ordinates of any polygon vertex **P**, the shadow vertex **S** can be calculated by multiplying like below:

$$S = P * M_{shadow}$$

This algorithm is still simple, only works if the ground is flat and it does not shadow other objects. But in contrast to the Fake Shadow algorithm, it performs an accurate calculation of the shadow polygon and has no restrictions concerning rotations and translations of an object.

3. Shadow Z-Buffer

This method follows directly from the idea that points in shadow are "hidden" from light. In other words, shadows are "hidden surfaces" from the point of view of a light (see also [1] and [2]).

If we pretend that the light point is the center of projection (camera point), the scene can be rendered from the point of view of the light using a Z-Buffer to compute surfaces visible to the light. The Z-Buffer resulting from this will record all of the points that are closest to the light. Any point that has a "farther" Z value at a given pixel is invisible to the light and hence is in shadow.

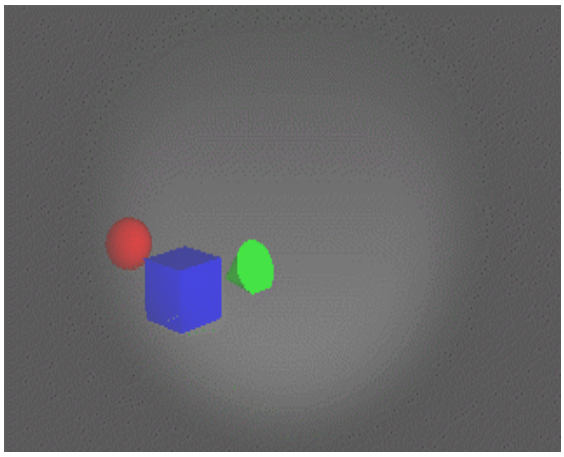

Figure 7 Light view of scene

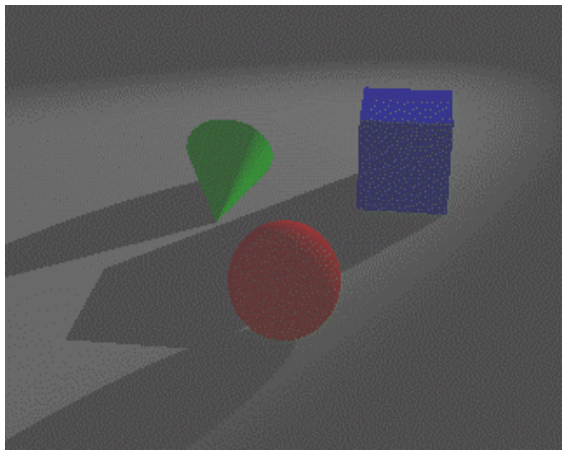

Figure 8 Camera view of scene

The Z-Buffer method involves looking at the object from the point of view of each light in the scene and computing a Z-Buffer of the object as seen by each light. After this preprocessing is performed, the object is rendered from the "true" camera position. For every pixel visible, the object point is transformed into the view of the light to determine whether that point was visible to the light by comparing the transformed Z value of the point with the corresponding Z value in the Z-Buffer. If it was not, then that point is in shadow.

When calculating the hidden surfaces from the point of view of each light source, it is only necessary to calculate the depth information and not to perform full lighting calculations for these polygons, because the "light's eye views" will not normally be seen by the user. This permits faster rendering when pre-calculating the shadow Z-Buffers, because this is just an intermediate step for the Z-Buffer shadow algorithm.

But the shadow Z-Buffer algorithm has two serious problems relating to how the pre-computed Z-Buffers are sampled. Consider a point that is visible to both the eye and a light. When transforming the point's world coordinates to shadow coordinates the point will (ideally) project in the shadow Z-Buffer to the same spot that this point projected to when viewed from the light. But it is still possible, that the algorithm may decide that the point is in shadow: Due to inaccuracies in the projection calculations, the point may project to a spot in the shadow Z-Buffer that has a slightly "nearer" Z value, this is known as self-shadowing or because the X and Y values are inaccurate and therefore mistakenly compared with on of its neighbors in the Z-Buffer.

The solution to the problem of points "shadowing themselves" is to cheat a little when transforming the point into shadow coordinates to see whether it is obscured by anything. This is done by adding a small fudge factor so that points project in front of themselves and thus do not shadow themselves.

The solution to the problem of comparing with the wrong Z-Buffer values is to perform "Area Sampling" of the Z-Buffer around the projected point, rather than just "Point Sampling". Area Sampling is typically done by calculating the Z values of more than one point of the area, where the appropriate point is located. Afterwards the average or the median of these Z values is used for Z-Buffer shading (see also [12]).

This is a very efficient algorithm which produces accurate shadows with very little effort, although it still has several downsides. Shadows can appear blocky depending on the resolution of the light Z-Buffer. An obvious downside is that the scene takes longer to render due to the fact that the scene is drawn at least twice.

4. Shadow Volumes

The basic idea behind the Shadow-Volume algorithm is to use a shadow count which is counting the number of shadow volumes that contain a point of a surface. This is done by incrementing the shadow count by one whenever there exists a shadow front-facing polygon crossing in front of the nearest visible object and decrementing the shadow count by one whenever there exists a shadow back-facing polygon. If the final shadow count is zero, then the visible object does not lie in shadow (see also [3] and [12]).

The algorithm depends on three important components:
- a means for generating the silhouette of an object,
- a method for drawing the shadow volume polygons, and
- a technique for rendering the actual shadow.

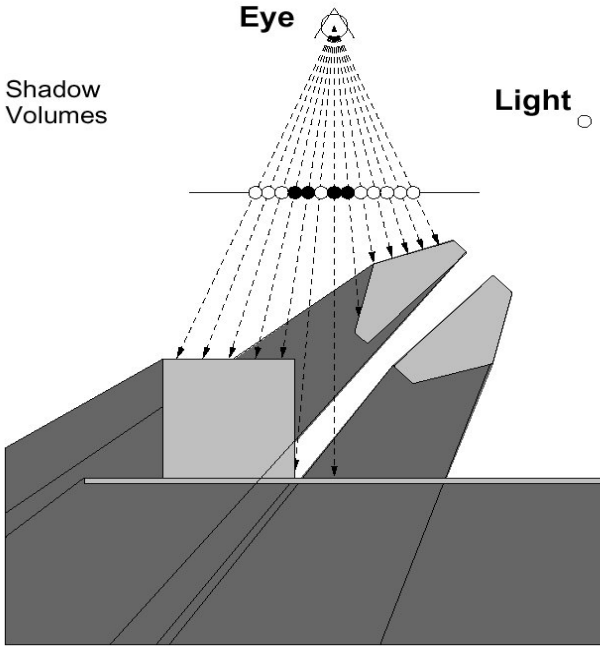

Figure 9 Shadow Volume(s)

The entire scene is initially rendered without any consideration for shadowing but with full color and depth information.

The silhouette of an object can be determined by finding the boundaries between adjacent front-facing and back-facing polygons. Once the silhouette of an object has been found, the shadow volume can be created and projected. This is done by adding the light vertex to the silhouette vertexes of an object and forming new polygons which are known as shadow planes. These shadow planes are open and infinite and all together form the shadow volume. Normally the shadow volumes must be clipped to the viewing area, but it must be ensured, that the volumes remains closed. This mechanism is known as capping and is important for the Shadow Volume, otherwise the generated shadow would be incorrect.

The shadow volumes are actually rendered twice into a separate buffer, invisibly both times. An appropriate implementation for this buffer is the stencil buffer. The stencil buffer is similar to other buffers, except stencil pixels do not represent colors or depths, but have application-specific meanings. The stencil buffer is not directly visible like i.e. the color buffer, but the bits in the stencil planes affect the drawing commands, through the stencil function and the stencil operations. The stencil function controls whether a fragment is discarded or not by the stencil test and the stencil operation determines how the stencil planes are updated as a result of that test (see also [9] and [11]).

First, the front-facing polygons of the shadow volumes are rendered into the scene, incrementing the stencil buffer value for every pixel that passes a normal Z depth test. Next, the back-facing polygons are rendered and the stencil buffer is decremented for every pixel that again passes a normal Z depth test (see also [7] and [9]).

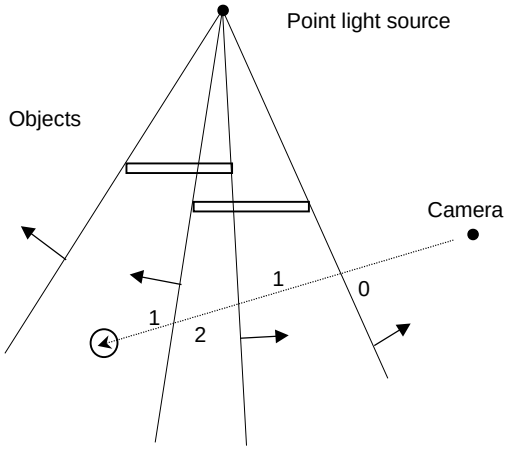

Figure 10 Creation of shadow volume using stencil buffer

As a result of these operations, the stencil buffer holds a positive value for all pixels that lie within a shadow volume. The shadow itself is rendered by drawing a single black or a semitransparent black polygon over the entire scene to darken all pixels in shadow using the bits in the stencil buffer. After this, the shadows of the first light appear in the scene as intended and the entire process can be repeated for the next light in the scene.

The benefit of this algorithm is the greatly improved realism and the hardware acceleration of the stencil buffer implemented into modern graphic cards. On the other hand, the programmer must deal with fill-rate limitations, because the shadow volumes must be rendered twice for each light source and one polygon must be drawn over the entire scene to render the shadow. Also, this algorithm produces sharp shadows and for the creation of the shadow volume some scene-management concerns must be considered.

5. Shadow Volume Reconstruction

The Shadow Volume algorithm and the Shadow Z-Buffer algorithm can be combined into a single algorithm. This will result in a new algorithm, the so-called Shadow Volume Reconstruction (see also [1] and [12]).

The basic algorithm is to create a Z-Buffer from the point of view of the light (just like in the Shadow Z-Buffer algorithm). After that, an edge filter is run on this light Z-Buffer, which produces the silhouette image of the objects. These silhouette images are then used to create shadow volumes and to render the shadow accordingly just as in the Shadow Volume algorithm.

By using these silhouette images, the shadow volumes can be reconstructed. This is done by transforming the silhouette image in the Z-Buffer back into world space, project it through the same viewing transformation as the rest of the scene and rasterize it by one of the two possible ways:

- generate the parts of a mesh that touch at least one silhouette edge (see Figure 11), or
- create a lookup table to determine what the mesh should look like. The lookup table would simply contain the 16 possible combinations of four adjacent pixels (see Figure 12).

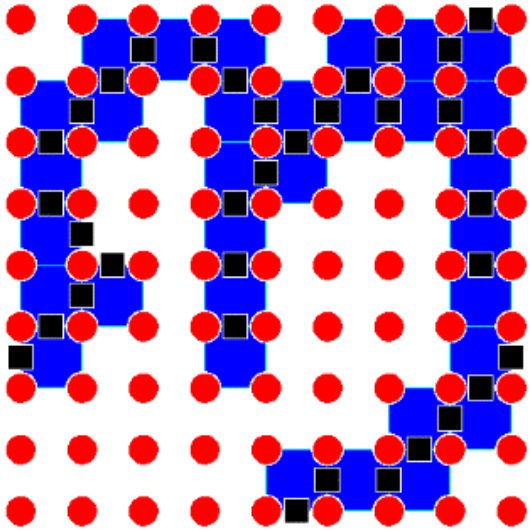

Figure 11 Generating the Mesh I

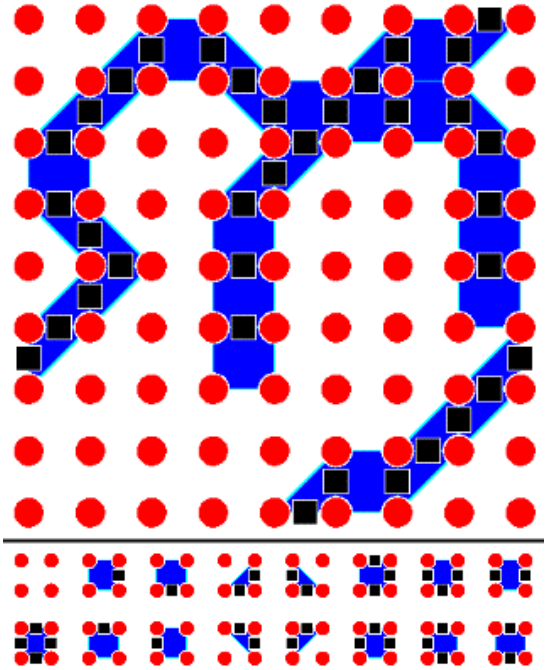

Figure 12 Generating the Mesh II

The black darkened points, in the figures above, are produced by the edge filter and form the silhouette image of an object.

The created shadow volumes can then be rendered to the stencil buffer like in the normal Shadow Volume algorithm. Finally the shadow is drawn to the entire scene by drawing a single black or a semitransparent black polygon over it to darken all pixels in shadow using the bits in the stencil buffer.

This algorithm is a compromise; it does not have the speed of the Shadow Z-Buffer method, because the additional steps of applying the edge filter and the reconstruction of the shadow volume is needed. However, it looks smoother but not as smooth as the Shadow Volume method, because the edge detect filter produces inaccuracies and the shadow volume reconstruction creates many polygons. It is an in-between, and it was intended to show that Shadow Volumes and Shadow Z-Buffers are similar and have different strengths that can be combined in a hybrid.

6. Static Shadow

There is another category of shadow algorithms, so-called Static Shadow. This method is actual not a true real-time Shadowing algorithm, but is also mentioned because of completeness.

The idea is that if some object or characters in the scene are not moving around, it makes sense to use static pre-calculated shadow maps. I.e. if you are changing your material settings only or just changing the camera position without animating the object, changing the position of the light and the object where the shadow is cast on, the static shadow approach can be chosen. Therefore the position of the shadow, generated by these object need not be calculated during runtime and therefore render much faster.

The static shadow of these objects are mostly represented by light-maps which can be rendered as textures on normal polygons into the entire scene.

Use Cases

Shadows in games and in so-called 3D engines are mostly implemented in order to give the viewer a better and more realistic representation of the three dimensional scene.

In most cases and most of the implementations it is more useful to have a fast and real-time computation of shadow in the scene, instead of an exactly calculated shadow of all the objects. In contrast, ray-tracers are

intended to reproduce an exact representation of the scene. As always, machine and graphic card power is limited, so in games and 3D engines, calculating shadow is a trade off between speed and shadow quality.

The 3D engines in popular games usually have some kind of implementation of a shadow algorithm. Nevertheless it is very hard to find useful information about which algorithm is used by which game. The only example found was from the game called 'Outcast' (see also [6]).

Outcast:

In Outcast, shadows of background and static objects are pre-computed. However for dynamic objects and characters, shadows are rendered in real-time. The 3D engine of Outcast deals with voxels and polygonal representation, which made it impossible to use standard shadowing algorithms that relied exclusively on polygons.

Problem 1: How to know if a given pixel is in the shadow?

The first thing to determine is whether a given screen point is in the shadow of a given dynamic object or character mesh. Therefore the mesh is transformed from the point of view of the light and from it a shadow buffer is created. This shadow buffer gives the distance from this light (similarly to the Z-Buffer). For a given screen point, a back-projection using the screen Z-Buffer to retrieve a 3D point in the camera space must be performed. After successive transformations to convert from camera space to light space and by comparing the z value (in light space) with the corresponding z value of the shadow buffer, it can be found out whether a point is into the shadow or not. So this is a kind of software Z-Buffer shadow algorithm which cannot be accelerated by modern graphic cards.

Problem 2: Quickly find shadow outline

It is not realistic to check all the pixel of the screen. The originality of the method used in Outcast is to process only the outline of the shadow. Therefore it is necessary to first find one point which is in the shadow. Once such a point is found, each line is scanned to find the first point of the outline of the shadow. After this a shadow boundary detection algorithm which will back-project and check only the outline of the shadow is executed.

This method dramatically minimizes the number of pixels processed and therefore it is very efficient. See also [6] and Figure 13.

Figure 13 Outcast snapshot

Conclusion

Shadows are a very important spatial cue. They help determine the relative positions of objects, particularly depth order and the height of objects above the ground plane. Since a shadow is basically a projection of the scene from an alternative viewpoint, cast shadows can also elucidate the shape of an object and the positions of light sources. Generating shadows is a classic computer graphics problem, and considerable research has been devoted to it.

Existing real-time shadow algorithms can be divided into three broad categories: projection, shadow volumes, and shadow maps. There are several hybrid algorithms that combine features from different categories to eliminate restrictions or limitations.

Each algorithm described in this paper has it advantages and disadvantages. But as stated before nowadays it is a must for games producers to include shadow generation in their engines and of course one of the described shadow algorithms is used.

References

[1] **Michael Skinner**. "Shadows". http://www.gamedev.net/reference/articles/article1300.asp

[2] **Chris Bentley**. "Two Shadow Rendering Algorithms". http://www.cs.wpi.deu/~matt/courses/cs563/talks/shadow/shadow.html

[3] **Jason Bestimt and Bryant Freitag**. "Real-Time Shadow Casting Using Shadow Volumes". Intel Corporation 1999

[4] **Georg Mischler**. "Umbra - Penumbra (Terms of physics/lightning)". http://www.schorsch.com/kbase/glossary/penumbra.html

[5] **J. Blinn**. "Me and my (fake) Shadow". IEEE CG&A, Vol. 8, No. 1, 1988, pp. 82-86

[6] **Unknown Author**. "Shadow Effects in Outcast". http://www.appeal.be/products/page1/Outcast_GDC/outcast_gdc_10.htm

[7] **Andrew Woo, Pierre Poulin and Alain Fournier**. "A Survey of Shadow Algorithms". - IEEE CG&A 10(6):13-32, 1990

[8] **Franklin C. Crow**. "Shadow Algorithms for Computer Graphics". Computer Graphics (Porc. SIGGRAPH), Vol. 11, No. 3, Aug. 1977, pp. 242-248

[9] **Mel Slater**. "A Comparison of three Shadow Algorithms." The Visual Computer 9:25-38, 1992

[10] **Mark J. Kilgard**. "Improving Shadows and Reflections via the Stencil Buffer". nVIDIA Corporation - Advanced OpenGL Development

[11] **Sim Dietrich**. "Using the Stencil Buffer". nVIDIA Corporation

[12] **Hsien Ching Kelvin Sung**. "Area Sampling Machine". University of Illinois 1992

[13] **Michael D. McCool**. "Shadow Volume Reconstruction from Depth Maps". ATM Transactions on Graphics, 2000