Heuristical Real-time Shadows

Meaney, D.
CBT systems, Dublin

O'Sullivan, C.
Image Synthesis Group, Trinity College Dublin

1 Introduction

Computer-generated graphical scenes benefit greatly from the inclusion of accurately rendered shadows. Shadows contribute to the realism of a scene, and also provide important depth cues, providing information relating to the relative position of objects within a scene. Indeed, it has been demonstrated in [Wanger et al. 1992] that compared to other visual cues, such as elevation, texture, perspective and motion, shadows result in the greatest improvement in the accuracy of positional perception and size perception of objects in a scene.

Dynamic scenes, such as animations and real-time simulations, employ shadows to more clearly illustrate object interaction. However, shadow generation imposes a significant penalty in terms of the time required to render a scene, especially as the complexity of the scene and the number of polygons needed increase. Soft shadows, i.e. shadows generated by light sources of finite size rather than point light sources, further complicate and prolong the rendering process. The time needed to generate realistic scenes with shadows becomes an issue warranting serious consideration in the area of real-time scenes. A frame rate of 15 frames per second is normally required and therefore established rendering techniques such as ray tracing and radiosity, which handle shadowed scenes very effectively, become impractical.

This paper proposes that real-time scene generation benefits from the use of a heuristical approach to the determination of shadow areas. For example, the shape of a moving object is perceived less accurately by a viewer than the shape of a static object. So it is reasonable to say that less effort - and therefore less time - should be expended on realising the shadow of a moving object compared to that of a static object. The main objective is to suggest a number of heuristics that might be employed to facilitate real-time animation at acceptable frame rates. An OpenGL application was created to investigate some aspects of the proposed approach. This overview will break down as follows. Section 2 discusses the background of shadow generation, and outlines existing work done in this area. Section 3 introduces the heuristical techniques proposed and gives some background to established heuristical techniques used in other areas of computer graphics. Section 4 outlines the technical aspects of the application written to illustrate some of the techniques proposed in Section 3 and Section 5 presents and analyses results. Finally, Section 6 draws conclusions from the work and proposes some future work.

2 Techniques for Generating Shadows

Shadows are an important part of a scene generated by computer software, but in many applications are neglected because of resulting performance degradation, excessive memory usage or lack of generality [Heckbert and Herf, 1997]. In essence, a shadow is an area of reduced illumination, caused, for a given point, by the obstruction of a direct path between that point and a given light. The obstruction is referred to as the *shadow casting object*, or *occluder*. The inclusion of shadows in a computer generated scene can be approached in a number of ways. In addition to the various techniques described in the next section, a decision can be made as to whether to render shadows as *hard* or *soft*:

- *Hard Shadow:* A hard shadow is characterised by having a distinct edge and uniform shading across its area, typically caused by a point light source, i.e. a light source that has no area, just a position (figure 1).

- *Soft Shadow:* A soft shadow is characterised by having a less distinct edge, often blending from the colour of the shadow at its darkest to the colour of the plane the shadow is being projected onto. It is the effect of an *extended* light source, i.e. a light source of finite area, that causes this softening at the edge of the shadow (figure 2). The darkest and most solid area of such a soft shadow is called the *umbra.* The less distinct area at the edge is called the *penumbra*.

The generation of shadows in computer graphics has been addressed extensively, using a number of techniques:

- Projection shadows
- Shadow volumes
- Shadow maps
- Ray tracing
- Radiosity
- Discontinuity meshing

The following sections describe each of these techniques and assess each in terms of its relevance to real-time shadow generation.

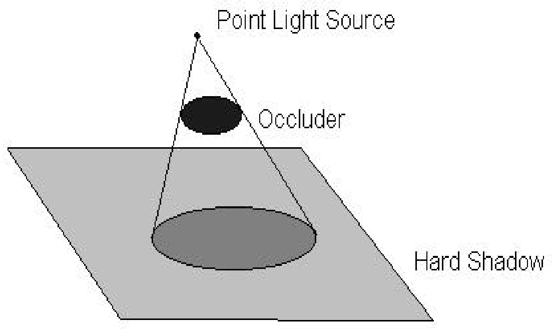

Figure 1 Creation of a hard shadow

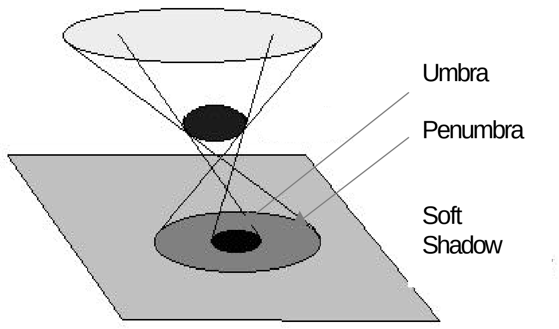

Figure 2 Creation of a soft shadow

2.1 Projection Shadows

This method uses a projection transformation to simply apply an orthographic or perspective projection matrix to the modelview transform, rendering the projected object in an appropriate shadow colour [Blinn 1998]. The mechanics of generating the shadows for a scene using OpenGL simply involve the determination of the correct projection matrix, multiplying this with the current modelview matrix, and rendering the entire scene.

One attribute of projection shadows that is very useful is the fact that each shadow is an entity that exists in the scene, rather than being simply an area of reduced illumination or a lighting effect. This means that each shadow can be manipulated just as any other object can. It also means that information relating to the shape and location of a shadow can be used in calculations relating to the scene. For example, in an application that employs collision detection techniques, shadows can often provide information that might simplify or speed up the process by which a decision is made as to whether two objects are in contact. The flexibility to manipulate shadows generated by projection made this an attractive technique for the work done in this project.

2.2 Shadow Volumes

Shadow volumes can be briefly described as the volume on the opposite side of an occluder from a light source that is blocked from a view of the light source by the occluder [Crow 1977]. Therefore any object in a scene that intersects this volume can be said to be in the shadow of the occluder, and by determining the exact intersection, the shadow boundary on that object can be drawn. Binary Space Partition (BSP) trees have been shown to efficiently and elegantly represent the total shadow volume formed by a set of polygons [Chin and Reiner 1989].

The shadow volume technique is an efficient algorithm for the computation of shadows, especially in scenes with relatively simple occluders. The algorithm can be easily extended to handle multiple light sources by repeating the second pass of the algorithm.

2.3 Shadow Maps

In essence, a shadow map is a depth map rendered from the point of view of the light source [Williams 1978]. Its advantage over the shadow volume technique is that its performance is not directly dependent on the complexity of the shadowing object. It is essentially a technique involving the transformation of distances from one viewpoint, that of the viewer, to distances from another viewpoint, that of the light source.

A shadow map is essentially a depth map that represents the distance of each pixel in a scene from the light source. On rendering the scene from the point of view of the camera, each pixel is analysed by transforming it back into the co-ordinates appropriate to the scene as viewed from the light source. Next the distance from the pixel to the light source is calculated and compared with the value stored in the shadow map for this pixel in the co-ordinate system representing the view from the light source. If this pixel has a larger value than that stored in the shadow map, it can be concluded that there is a shadow-casting object between the pixel and the light source, and so, returning to the scene as observed by the camera, the point should be drawn in shadow.

Shadow mapping provides a very elegant technique for rendering shadows. It offers the advantage of not being directly dependent on the complexity of any occluding objects in the scene, and does not require an analysis of the scene using binary space partition trees as with shadow volumes. However, a problem that can arise is aliasing artefacts that are introduced by the fact that the shadow map is point sampled. It is possible that the area coverage of a pixel in one co-ordinate system may be significantly different to that of the same area in the other co-ordinate system, thus introducing quantisation errors. Consider, for example, a point in a model that is close to the camera position, but distant from the light source. Upon rendering the scene, when this point is transformed into the light source co-ordinate system, the resolution of this area of the scene could be significantly reduced. This could be evident in the final rendering of the scene as the shadow boundary may be displayed in the same low resolution that is associated with the light source view of the scene. In general it can be said that there is an uneven mapping of resolution between the scene rendered form each viewpoint. Therefore precision is an issue that can be very important to this technique.

A version of this algorithm has been implemented in hardware, as described in [Segal et al. 1992]. The shadow map in this example was stored as a texture map, so that the texture co-ordinate transformation hardware could be employed to perform a comparison between the transformed *z* co-ordinate and the stored *z* co-ordinate stored in the shadow map. While this technique is useful in speeding up these calculations, it does present the drawback of requiring a very high precision in the texture mapping hardware, and it is also susceptible to creating a bottleneck in the texture mapping hardware when it is required for pure texture mapping functionality while the scene is being rendered.

Another problem is that it is difficult to use the shadow map technique to cast shadows from a light surrounded by objects. This is because the shadow map is created by rendering the entire scene from the light's point of view, and therefore it is not always possible to come up with a transform to do this, depending on the geometric relationship between the light and the objects in the scene.

In general, these issues can be overcome and this is a very useful technique. However, for the purposes of this project, and as with the shadow volume technique, the shadows are generated globally with no distinction made between individual shadows. Also, as the overall shape of individual shadows are not resultant from this technique, there is little scope for manipulating the shadows based on the perceived importance of each shadow in the scene.

2.4 Shadow Volume and Shadow Map Hybrid

One very interesting shadow generation technique that has been demonstrated recently in [McCool 1998] is a hybrid of the shadow volume technique and the shadow map technique. This technique, Shadow Volume Reconstruction, is based on the observation that once a shadow map has been generated, the information in the shadow map (i.e. the depth, or *z* information for the scene as observed from the light source), combined with the *x*, *y* co-ordinates for each pixel as observed from the light source actually define a volume. That volume, when transformed to the appropriate co-ordinates for rendering from the point of view of the camera, is a shadow volume.

The generation of this shadow volume is more efficient than that generated by the pure shadow volume technique described by Crow. This is because the volume generated in the pure shadow volume technique is composed of many polygons which are often overlapping. The shadow volume reconstruction technique generates shadow volumes that are minimal in size; they extend to the shadowed surface from the occluding object, but no further. This technique also offers advantages over the shadow mapping technique by eliminating the requirement - in the hardware implementation – for the use of texture maps. Therefore the scene can contain textured polygons without the need for an extra rendering pass.

However, this technique retains some of the less desirable attributes of the two existing methods. It inherits the quantisation problems outlined above. Also, the most basic variant of this hybrid suffers the disadvantage of Crow's algorithm, i.e. the potentially large number of shadow polygons generated. The report, however, describes edge detection techniques that were employed to simplify the shadow volume such that only inefficiencies are removed.

2.5 Non-Real-Time Approaches to Shadow Generation

Non-realtime techniques are important to the area of shadow generation in computer graphics. However, these techniques are computationally intensive and could not be considered for use in a real-time application. Ray tracing algorithms determine the visibility of surfaces by tracing imaginary rays of light from the viewer's eye to the objects in the scene [Whitted 1980]. Ray traced scenes can be extremely realistic, particularly for shiny objects (specular lighting effects), but they require considerable computation time to generate. A ray tracing process will inherently reveal areas of the scene that lie in shadow. However, paying special attention to areas that represent shadow boundaries can enhance the overall effect.

Radiosity algorithms determine the direct and indirect illumination for diffuse surfaces in an environment and produce a view-independent solution. They do this by subdividing a surface into small elemental surface patches and, assuming they are small enough, approximating their intensity distribution over the surface using a constant value that depends on the surface and the direction of the emission.

3 Heuristical Techniques

A characteristic of graphical applications is the amount of error that can be tolerated in the final result. Unlike many applications that might be more analytical in nature, the product of a graphics-rendering system can ultimately be judged for quality by the viewer of the graphic. It is common for graphics to be inaccurate in many respects while still conveying an effective representation of the scene.

Some very well-established techniques in computer graphics rely on this characteristic. For example, texture mapping allows scenes of great complexity to be realised in quite a simple way. For example, such objects as buildings, which might require a large number of polygons, can be represented by a single cube with its sides texture mapped with the graphic of an existing building. In fact, many substances that are extremely difficult to render, such as bodies of water, vegetation, and hair can be represented efficiently by this method.

This paper approaches the use of heuristics with several objectives. High-level knowledge of the system is used to decide how important a shadow is, and therefore how much effort should be expended in drawing it. However, at a lower level, for a shadow that is deemed suitable for rendering with superior accuracy, a way of doing this more efficiently than otherwise will be described. A shadow that is deemed to be important to the scene is said to have a high priority, and a shadow that is considered to have relatively ittle impact on the overall scene is said to have a low priority.

The ideas presented here require a high level of flexibility in terms of manipulating the shape, dimension, position, and colour of each shadow. Therefore it is important to be able to identify individual shadows – even to the point that their vertices are known, thus allowing manipulation of the shadow objects to be performed. For example, a rough estimate of the overall length and width of a shadow allows a very low-priority shadow to be approximated simply by rendering an elliptical object of these dimensions.

The goal of a heuristical approach is to find a good balance between the richness of the shadows and the time it takes to display them, in order to maintain a constant frame rate. To this end, there are three types of shadow defined for the system:

- *detailed* shadows
- *correct* shadows
- *vague* shadows

3.1 Detailed Shadows

A relatively large effort is expended in creating detailed shadows. These shadows should be rendered in a way that will enhance their accuracy and realism. This includes drawing the shadow to the correct dimensions and position and also softening the edges of the shadow, i.e. creating an umbra and penumbra. Much work has been done on the creation of soft shadows, but most of these techniques are suitable only for static scenes. To achieve soft shadows in real-time requires speed-up techniques. One such technique would be to perform an analysis on the extent – or area – of the light source and the relative distance of the occluder from the light source and the shadow projection plane at the area the shadow will appear. This would give an indication of the relative sizes of the umbra and penumbra.

3.2 Correct Shadows

Correct shadows are drawn to the shape and dimensions that are exact for the occluder. The role that correct shadows play in this system is to render shadows that have some visual significance in the scene in a time-efficient manner. This is achieved by disregarding such shadow features as the penumbra. These shadows are essentially *hard* shadows. However, a further optimisation is realised in the analysis of the occluder to shadow transformation, and the convex hull operation that is performed on the shadow vertices.

3.3 Vague Shadows

A shadow being judged as having a low priority in a scene will have the least amount of time spent rendering it. As the lowest-priority shadows in the system, the emphasis is on quick rendering as opposed to an accurate representation of the shadow. Therefore these shadows will be rendered as simple circular or elliptical flat objects. To maintain some degree of accuracy, the dimensions of the shape will reflect the true dimensions of the shadow as it should appear. This should present no significant overhead, as the shadow dimensions are stored in the object representing the shadow.
Ideally these shadows would not be required in a scene. Their importance lies in the completion of a scene when time is very limited. When it is necessary to use them, as few as possible should be introduced into the scene.

3.4 Criteria for Use of Heuristics

Much research has been conducted in the area of visual perception in computer graphics. This research is largely directed towards such applications in computer graphics as realistic image synthesis, scientific and information visualisation, and time-critical rendering for virtual environments. Rendering a scene in computer graphics is largely concerned with providing enough information on a 2D plane, i.e. a page or computer screen, to an observer to allow the observer to interpret it correctly three dimensionally. This means that the relative position of objects, their orientation, and their speed and direction if they are in motion, are important visual cues that are not inherent in two dimensions. However, there must be enough additional information in these two dimensions to derive three dimensional scenes unambiguously.

Pictorial cues were first employed by painters wishing to create the effect of depth and correct spatial relationships in pictures. In [Wanger et al. 1992] these are broken down into the following, and the relative importance of each is assessed:

- **Perspective** – the foreshortening effect caused by increased distance from the viewpoint, thus establishing a size-to-distance relationship, which provides clear indications of the distance of an object when the size of that object is known
- **Texture** – a graphical indication of the nature of the substance of a surface, which is also an important visual cue for the determination of the spatial orientation and relief of flat and curved surfaces
- **Shading and shadow** – areas of varying illumination caused by the amount of light falling on different areas of a scene, which provide information on the shape and location of an object
- **Motion** – the relative displacement of objects in consecutive frames, due either to the movement of the objects or the viewpoint, can indicate distance between those objects
-

3.5 Relative Importance of a Shadow to a Scene

A number of techniques intended to speed up the determination and rendering of shadows are dependent on the following factors:

- Distance of the shadow from the viewpoint
- Motion of the shadow
- Location of the shadow in the viewer's peripheral vision
- Extent – or area – of the light source
- Relative distance of the occluder from the light source and the shadow projection plane

These are expanded upon in the sections below. Each aspect of a shadow as described here can be used to determine the importance of that shadow to the accuracy and realism of the scene. From the consideration of each of these criteria, a list of shadow objects may be drawn up and prioritised in order of decreasing importance to the scene. This prioritised list can then be used to select the method of rendering – detailed, correct, or vague – for each shadow. This decision should be based on the available time and the desired frame rate, taking into account the expense of each technique in terms of time.

3.5.1 Distance of Shadow from Viewpoint

Firstly the distance of the shadow from the viewpoint can be taken into account. In the context of this thesis this is only relevant if the scene is rendered using perspective. This is because the use of orthographic projection does not allow an observer to distinguish the distance of an object based on its size – all objects are drawn to their correct dimensions regardless of their position in the scene.

The foreshortening effect on objects that are distant in a scene rendered using perspective projection means that the accuracy with which they are drawn is necessarily reduced. This is a result of the same screen resolution being applied to objects positioned in both the foreground and the background of the scene. In [Funkhouser et al. 1992] this idea is applied using spatial subdivision and visibility analysis to perform a real-time architectural walk-through. Taking account of the distance of an object from the viewpoint will be extended to shadows in this discussion. As the shadow of an object moves toward the background of a scene, its detail will become increasingly obscured and therefore less time should be spent rendering it. As such, it would be a candidate for occupying a position of lower priority in the overall list of shadows in a scene.

3.5.2 Motion of Shadow

In a scene, objects that are in motion present the problem that they must be fully re-rendered for each scene. However, the details of moving objects are not as easily discerned by people as are the details of static objects. This is an observation that is commonly employed in computer graphics. For example, in the area of collision detection, much effort is expended in determining whether two objects are in contact. However, a commonly used shortcut relies on the fact that objects in a collision detection system will always be in motion and therefore the exact moment of contact between two objects will never be discerned accurately by an observer. Therefore a tolerance of proximity can be utilised [Hubbard 1995].

The same can be applied to shadows that are in motion. By taking into account the velocity of a shadow upon rendering a scene, a decision can be made to lower the priority of that shadow and render it as either a correct or vague shadow if it is moving rapidly.

3.5.3 Shadow Located in Viewer's Peripheral Vision / Direction of Gaze

When deciding to prioritise the shadows in a scene, or indeed any other object in general, some assumptions may be made as to the importance of the object based upon its location in the scene. One such assumption is the fact that objects that lie in the centre foreground will be the main focus of the scene. Therefore all other objects may be considered to be of less importance in the scene.

This technique is rather qualitative – much more so than the others discussed here. The more that is known about the context of the scene and the details of the elements that compose the scene, the more effective this approach will be. While this technique can be employed effectively in certain applications, it is not a suitable candidate for inclusion in a system that attempts to generalise to any 3D scene. Alternatively, an eye-tracking device could be used to locate the viewers point of fixation, and prioritise objects based on their distance from this point, as in [O'Sullivan 1999].

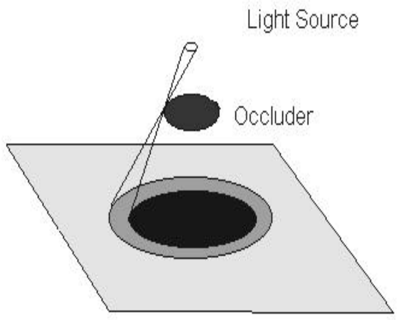

Figure 3. Soft shadow generation (occluder more distant from light source)

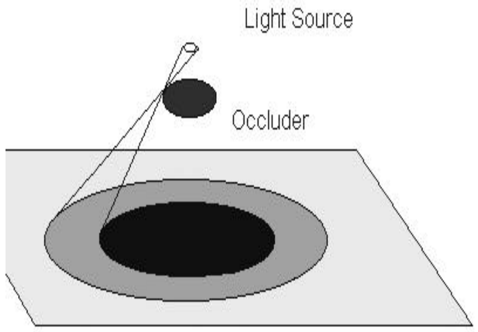

Figure 4. Soft shadow generation (occluder less distant from light source)

3.5.4 Extent of Light Source

The area occupied by the light source can have a significant impact on the nature of the shadow cast by an occluder. Point light sources will always cast hard shadows – shadows with no penumbra. A light source that occupies a finite area will result in a softer edge to shadows associated with it, and as the area increases, the degree of softness will increase for a given shadow. Computer graphic systems usually simulate this by rendering the scene with multiple point light sources distributed across the area of the light source.

The alternative to performing this costly computation is to draw up some heuristics relating to the size of the light source. One approach would be to devise a table relating the size of a light source to the extent of penumbra observed in a corresponding shadow. Then an approximation of the soft shadow could be rendered using a technique such as jittering (see Section 4.4.3).

3.5.5 Relative Distance of Occluder from Light Source and Shadow Projection Plane

Finally, one other attribute of a graphical scene employing shadows is the location of the occluder relative to the light source and the projection plane of the shadow. This information can be used to quickly deduce an approximation of the degree of softness of a shadow. It relies on the observation that the softness of a shadow is related to the relative distance of the occluder from the light source and the shadow projection plane; the closer the occluder is to the light source, the softer the shadow will be. This is a result of the geometry of the path of the ray travelling from the light source to the shadow projection plane (see figures 3 and 4).

4 RTShadow

RTShadow (Real Time Shadow) is a Microsoft Windows application, employing OpenGL and the Microsoft Foundation Classes (MFC), which was written to demonstrate some aspects of the generation of shadows in this thesis. It is based on the ideas for generating shadows in a number of different ways, increasing in accuracy and definition as the shadow is determined to be more prominent within a scene. It employs three types of shadow, as described earlier, namely *detailed*, *correct*, and *vague* shadows.

The starting point for the application was a sample program from [Fosner 1996], which provides an MFC framework application enhanced with an additional class, COpenGLView.

4.1 Determining Shadow Polygons

The shadows generated in RTShadow are based on objects within the application, each object representing one shadow. Each shadow object contains a list of vertices that represent the outline of a two-dimensional figure on the ***y = 0*** plane, and can be drawn as a polygon. These vertices can be used in a number of ways, depending on the type of shadow that is to be generated for a given shadow object. *Vague* shadows are based on a subset of the shadow vertex points, *correct* shadows use all the vertices, and *detailed* shadows use more advanced techniques to create a realistic shadow representation. These shadow types and the details of how they are created are expanded on in subsequent sections.

Projection shadows, as outlined in section 2.1, are used in RTShadow. Implementing the Shadow Projection Matrix in RTShadows could easily have been done in RTShadow by multiplying such a matrix onto the current OpenGL ModelView stack. Rendering the object after doing this would have resulted in what would essentially be a *correct* shadow, as defined in this application. However, it was desirable to have as much flexibility as possible in rendering and manipulating the different types of shadow. For this reason it was necessary to obtain the actual vertices of the shadow figure. This turns out to be much more difficult than it appears. Because simply re-rendering the occluding object with the shadow projection matrix on the ModelView stack creates the shadow, the vertices of the shadow do not need to be explicitly known. Indeed, OpenGL does not provide a way of returning the co-ordinates of the transformed vertices. Therefore a slightly different approach was taken.

The shadows in RTShadow make use of the shadow projection matrix, but instead of multiplying it onto the ModelView stack, each vertex in the occluding object is explicitly multiplied by the matrix to determine its corresponding vertex on the shadow. The application maintains a list of all the objects in the scene, and each of these objects contains a full list of the co-ordinates of the vertices in that object. At this point the performance of the application can be considerably enhanced by acting on the observation that in a shadow only the vertices that define the outline of the shadow need to be considered. In RTShadow, a Package-Wrapping convex hull algorithm has been implemented to handle this [Sedgewick 1992].

4.2 Vague Shadows

Vague shadows are drawn simply as quadrilateral shapes in the scene. These shadow types are included specifically to allow some shadows in the system to be drawn with the very least effort and impact on performance. Any shadows chosen to be drawn as vague shadows will be very distant in a scene or fast moving or both. The intention is ultimately that these shadows would only be used if the system really does not have enough time between frames to render fully the scene with correct and detailed shadows, and so they should be rendered as fast as possible.

Early on in the design of the application, the intention was to use circular or elliptical shapes to represent vague shadows. However, because of the number of vertices contained in these shapes, rendering them would mean that in many cases rendering simple correct shadows would be more efficient.

4.3 Correct Shadows

The drawing of correct or hard shadows is relatively straightforward. They are drawn as black polygons on the grey surface representing the floor of the application. The vertices for a single hard shadow polygon are stored as the *culled* vertices of the entire occluding object, projected onto the floor (**y = 0**) plane. The culling is achieved as described above using a convex hull algorithm.

Firstly, the colour for the shadows is set to black. In many respects, OpenGL behaves as a state machine, applying a setting to all relevant operations until that setting is either cleared or changed. This colour setting will apply to all vertices drawn through OpenGL, in any buffer, until another colour setting is specified.

As all the vertices marking the outline of the projected occluder are stored for each frame, shadows of arbitrary complexity can be rendered quickly.

4.4 Soft Shadows

RTShadow uses the OpenGL accumulation buffer to simulate soft shadows. The accumulation buffer holds RGBA colour data just like the colour buffers do in RGBA mode. It is not drawn into directly, rather the accumulation buffer is filled in rectangular blocks from data stored in a colour buffer. It allows a number of images to be accumulated into a final composite image, which is then transferred back to a colour buffer to allow the image to be displayed. Effects that are commonly created by use of the accumulation buffer are scene antialiasing, motion blur, and simulation of photographic depth of field.

4.4.1 Using the OpenGL Accumulation Buffer

Before a drawing operation is performed in a colour buffer or the accumulation buffer, there are a number of tests that are applied to the buffer before any information can be written to it, and the outcome of these tests determines the results of writing data to the buffer. The tests that are applied to these buffers are

- Scissor test
- Alpha test
- Stencil test
- Depth test
- Blending
- Dithering
- Logical operations

The *Scissor test* is employed in this project when creating soft shadows and is described in detail in the section *Enhancing the Performance of the Accumulation Buffer* below. *Alpha* testing is based on the fourth value in RGBA representation of vertices; it allows the drawing of a fragment to be accepted or rejected, based on a comparison with a reference alpha value. The *Stencil* test is performed in conjunction with a stencil buffer. The stencil buffer contains values with a one-to-one correspondence with the pixels in a scene. By a comparison technique, the values contained in the stencil buffer determine whether or not to draw pixels in a colour buffer or the accumulation buffer.

The depth buffer is used in the *Depth* test. Like the stencil buffer, the depth buffer maintains a one-to-one pixel correspondence with the colour and accumulation buffers. The values stored for each pixel in the depth buffer represent the distance between the viewpoint and the object occupying that pixel. The depth test updates the depth buffer by replacing values for objects that are discovered to be closer to the viewpoint than the existing value for a given pixel.

Blending, d*ithering*, and l*ogical operations* are not tests as such, they are image-processing techniques that can be applied in the colour or accumulation buffers. Instead of simply overwriting existing values in a buffer, blending allows the new values of a fragment to be combined with the existing values. Dithering is a hardware-dependent operation that can improve the colour resolution of an image with a small number of colour bitplanes. There are a number of logical operations that OpenGL allows to be performed between the pixels existing in a buffer and incoming pixels representing a fragment of an object in a scene. Such operations as AND, OR, NAND, NOR, XOR, INVERT, COPY, and CLEAR, amongst others, are available.

4.4.2 Generating Soft Shadows Using the Accumulation Buffer

Early scenes incorporating shadows simplified the light source so that it could be considered to be a point light source. The result is shadows with edges as sharply defined as the object creating the shadow . This takes from the realism of the scene and was considered an identifying feature of early computer-generated graphical scenes.
Soft shadows are a result of light sources that are of finite size, resulting in a shadow is composed of two parts – the *umbra* and the *penumbra.* Soft shadows can be derived from any of the shadow-generation techniques described in this report. The most common way of doing this is to approximate the extended light source by multiple point light sources distributed across its surface area, essentially quantising it.

A huge cost is incurred by generating accurate soft shadows in this way. Not only does the scene have to be rendered multiple times, but particularly for techniques such as projection shadows and shadow volumes the shadows themselves must be recalculated for each rendering. The more times the scene can be rendered, i.e. the more point light sources are used to represent the extended light source, the better the quality of the shadows generated; artefacts such as step effects are less apparent. This, of course, slows the process down further, although it can be very effective for scenes that don't require real-time rendering rates.

The method used by RTShadow to generate soft shadows is a variation on this approach. It is possible to speed up the rendering process by taking some shortcuts, resulting in soft shadows that, while they are not perfectly accurate, are nevertheless effective. This approach still requires the shadow to be rendered multiple times to the accumulation buffer, but the position of the shadow is varied on each iteration by a rule of thumb, rather than by carefully calculating the correct position. The method employed to produce the soft shadow is called *jittering.*

4.4.3 Jittering

Jittering is a method of effectively vibrating the image, and of accumulating the images as they move by relatively small amounts about a centre point. In the context of soft shadow generation, it is the initially calculated shadow that is jittered, and the overall image accumulated. To increase the area of the shadow that appears as the penumbra, the degree of movement about the centre can be increased.

Jittering is an iterative process; the image is written to the accumulation buffer, transformed by a relatively small amount, then rewritten to the accumulation buffer, and so on. The distance by which the image is transformed on each iteration is important. While it might seem natural to choose positions that are equally spaced, this is not always the case. It is considered better to use a uniform or normalised distribution, clustering toward the centre of the image.

4.4.4 Rendering the Shadow

The vertices describing the outline of the shadow will have been determined at the time the rendering of the shadow is done. Within the RTShadow project, soft shadows are created in the method **RTShadow::softShadow()**, but this function is supported by a number of others. While the other shadow types in this application are drawn individually as they are encountered in the global list of associated occluder objects, the use of the accumulation buffer necessitates a slightly different approach for soft shadows.

5 Performance

This section outlines some issues relating to the performance of the RTShadow application and makes observations on the relevance of these issues to the area of real-time shadow generation.

5.1 Comparing Detailed, Correct and Vague Shadows

Figures 5, 6, and 7 show detailed, correct, and vague shadows respectively, as RTShadow renders them. In each case the shadows are cast by two cubes, created by RTSOccluderCube objects. The example of the detailed shadows required moving the positions of the cubes to allow the shadows generated to overlap somewhat. The position of the cubes for the correct and vague shadows are identical.

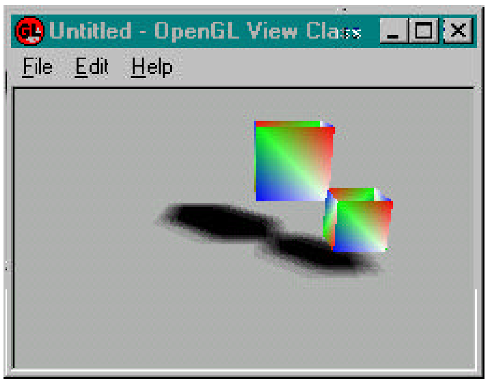

Figure 5 Overlapping detailed shadows in RTShadow

The detailed shadows shown in figure 5 demonstrate the soft shadow effect created by jittering a number of hard shadows and superimposing them on each other. The number of shadows used in the jittering process for this example was eight. It is also worth noting the way that the two shadows blend into each other. To achieve this it was necessary to render both shadows simultaneously in the accumulation buffer.

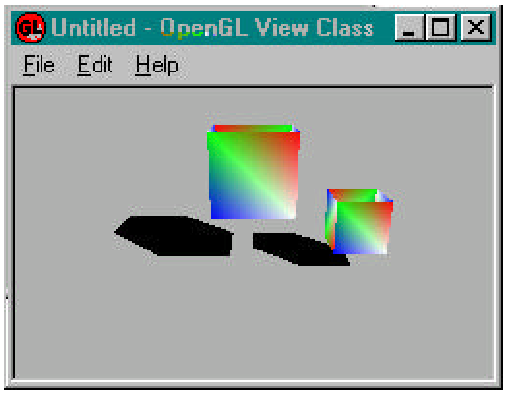

Figure 6 Correct shadows in RTShadow

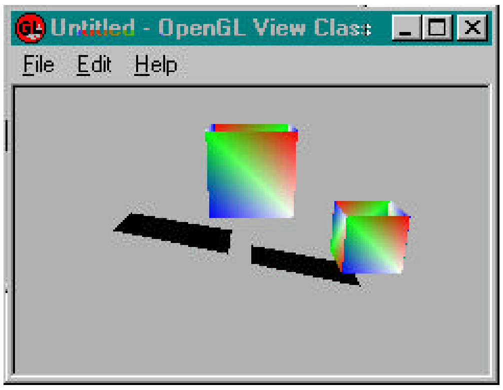

Figure 7 Vague shadows in RTShadow

The correct shadows shown in figure 6 demonstrate the fall off in visual quality by rendering hard shadows. However, the result is still accurate and provides a reasonable visual representation of the shadows for the cubes. The advantage of the correct shadows is a dramatic improvement in rendering time.

The vague shadows shown in figure 7 are obviously a method of compromise. However, the intention is to use these in situations where they would be very much more to the background and where their importance to the scene would be reduced due to other objects with either detailed or correct shadows.

5.2 The Performance of RTShadow

This section outlines the performance in time of RTShadow for detailed, correct, and vague shadows. It further breaks down the demonstration of detailed shadows by considering the generation of these shadows by writing to the full accumulation buffer and by clipping the area used in the accumulation buffer to the area occupied by the shadows. The timings given below were based on rendering the detailed shadows with and without the OpenGL Scissor test enabled, rendering the correct shadows once, and rendering the vague shadows three times, with between two and ten occluders in the scene.

5.2.1 Detailed Shadows – No Clipping

At 4.817 seconds to draw detailed shadows for two objects, this approach cannot be considered. Assuming that for real-time rendering a frame rate of 15 frames per second is adequate, that translates to a rendering time of 0.067 seconds. The performance of RTShadow in this specific case is obviously not to be considered a candidate for real-time applications. The use of the accumulation buffer is the cause of the poor performance here. However, it is worth noting that the accumulation buffer used to perform this test was implemented in software on the test machine. Much better results could be expected with hardware acceleration for the accumulation buffer functionality.

5.2.2 Detailed Shadows – Clipped

In order to improve the performance of the accumulation buffer, it is possible to clip the area of the buffer used to the area that needs the functionality provided The time taken to perform the accumulation buffer operations was 0.832 seconds, to generate the same shadows that took 4.817 seconds in the previous example. Although this is obviously a useful technique for speeding up the accumulation buffer's performance, it is probably best suited to honing the performance of an already fast application. The results show that this is not enough to take this application into the realm of real-time shadow generation.

5.2.3 Correct Shadows

The scene for correct shadows is set up exactly as with the examples for the detailed shadows. At 38.626 milliseconds to render the scene, it is apparent that correct shadows provide a much faster solution than detailed shadows, the obvious reason being the lack of dependence on the use of the accumulation buffer. This is demonstrated by the fact that the most time-consuming function in the example of correct shadow generation is **RecalculateShadow()**, which determines the vertices of the shadow and is not directly involved in the rendering process. There are two important operations performed in this function: the matrix multiplication by the shadow projection matrix, and the convex hull algorithm used to cull the list of vertices in the shadow to just those needed.

The performance of the correct shadow example here can be said to be reaching real-time performance, especially in terms of rendering time. However, it would benefit from a more efficient convex hull algorithm.

5.2.4 Vague Shadows

The results for vague shadows (39.7 milliseconds) are broadly similar to those for correct shadows. As with correct shadows, the bulk of the computation time is concentrated on calculating the area of the shadow. A performance gain over correct shadow generation is realised for vague shadows in the OpenGL operation of rendering the shadows.

6 Conclusions

In this paper, existing techniques in 3D computer graphics for generating shadows were appraised, focusing specifically on their relevance and suitability for real-time applications. This overview included an outline of several heuristical techniques to help improve the overall visual effect of the graphic with a minimum additional performance overhead. A practical implementation of one of these techniques was devised in order to illustrate some of the issues associated with real-time shadow generation.

In Section 2, the substantial body of work done on the theory and practice of shadow generation in computer graphics was outlined. The earlier ideas, centring on the straightforward projection of the outline of an object onto a projection plane, were quickly built upon with such practices as shadow volumes and shadow maps, which allowed more efficient inclusion of shadows in complex scenes. These methods are considered suitable candidates for development of real-time shadow generation. Some other important methods that allow the inclusion of shadows are not suitable for real-time applications. Ray tracing, radiosity and discontinuity meshing, while they are capable of rendering shadowed scenes of remarkable accuracy and detail, are employed in the area of photorealism and as such are time-consuming in the rendering process.

Real-time shadow generation remains a huge challenge to the developers of computer graphic applications. Fifteen frames per second is considered a standard for real-time animation. This allows a mere 66 milliseconds to construct and render the entire scene – a formidable task considering the fact that most viable shadow-generation techniques require at least one complete re-rendering of the scene in addition to the calculation of the shadow areas. Much of the literature available on shadows in computer graphics concentrates on accurate and realistic *static* images and the depth that this work achieves illustrates the problems that realistic real-time applications must surmount. There is a clear trade off between the effectiveness of real-time shadows as an animated element of a scene and the visual quality of the shadowed areas. Further complications are introduced when the ability to cast shadows, not only on a shadow projection plane – for example, a floor area – but also on other occluding objects in the scene is required. This vastly increases the computation time required, and rules out techniques such as projection shadows.

Also considered, in Section 3, was the use of some heuristical techniques to aid the generation of real-time shadows. These techniques are employed to speed up the rendering process, but at the expense of the quality of the graphic produced. Such techniques have been applied to many computer visualisation problems, and their applicability to the area of shadow generation was discussed. A number of heuristical techniques were outlined. These included such established computer graphics methods as taking into consideration the relative importance of an object based on its position and motion within a scene. Also, some techniques specific to the topic of shadow generation were outlined. These tended to concentrate on ways of producing shadows of good quality in an efficient way.

The application presented in Section 4 of this project – RTShadow - was written to illustrate some of the outlined aspects of real-time shadow generation. It employs three shadow types: detailed, correct, and vague. The detailed shadows emulate most accurately the way shadows should appear, being correct in outline and softened at the edges. The softening effect is achieved by rendering the shadows multiple times in slightly different positions, accumulating the renditions, a technique called *jittering*. While it would be desirable to render every shadow in a scene in this way, it is not practical due to the computational expense of this method. Correct shadows are accurate in outline, but dispense with the most time-consuming aspect of detailed shadows, namely the jittering. Vague shadows are drawn as simple polygon shapes, the emphasis being on rendering a dark area to represent the shadow as fast as possible.

RTShadow clearly demonstrates the dependence a good real-time shadow-generation engine would have on the use of graphics acceleration hardware. The poor performance of the jittering process is a direct result of the fact that the generic version of OpenGL provided with Microsoft Windows NT provides only a software implementation of the accumulation buffer. To generate such realistic shadows at real-time frame rates would certainly require good quality graphics hardware support.

The approach taken for the calculation of the shadow areas in RTShadow presents interesting possibilities in a more general 3D real-time animation application. The shadows are treated as independent entities which, while they are closely related to the objects that act as the occluders, are fully described as objects in the scene. This attribute can have crossover benefits for the process of collision detection. This is because the shadows provide additional information about the location of objects relative to each other. For example, from a certain viewpoint it may be difficult to discern the proximity of two objects if they are closely in line. However, if the light source projecting the shadows is sufficiently distant from the viewer, the shadows may clearly indicate that the relevant objects are not in contact with each other. Such techniques are useful for speeding up the decision process in collision detection systems. The real-time shadows considered in this discussion are of a dynamic nature. Typically generated on a frame-by-frame basis, they are calculated from objects and light sources that may be manipulated by a user of the system.

[Blinn 1998] Blinn, James, “Me and my (fake) shadow”, IEEE Computer Graphics and Applications, January 1998.

[Chin and Reiner 1989] Chin, N., Reiner, S. “Near Real-Time Shadow Generation Using BSP Trees”, Computer Graphics 23(3), pp 99-106, 1989.

[Crow 1977] Crow, F., “Shadow Algorithms for Computer Graphics”, Proc. SIGGRAPH, vol. 11, pp 242-248, July 1977.

[Fosner 1996] Fosner, Ron, “OpenGL Programming for Windows 95 and Windows NT”, Addison-Wesley Developers Press, 1996

[Heckbert and Herf, 1997] Heckbert, Paul S., Herf, Michael “Simulating Soft Shadows With Graphics Hardware”, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1997.

[Hubbard 1995] Hubbard, Philip M., “Collision Detection for Interactive Graphics Applications”, IEEE Transactions on Visualisation and Computer Graphics, 1(3), Sept. 1995, pp. 218-230

[McCool 1998] McCool, Michael D., “Shadow Volume Reconstruction”, Technical Report. Computer Graphics Laboratory, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, March 1998.

[O'Sullivan 1999] O'Sullivan, C. *A Model of Collision Perception for Real-Time Animation*. Technical Report TCD-CS-1999-02, Trinity College Dublin, January 1999.

[Sedgewick 1992] Sedgewick, Robert, “Algorithms in C++”, Addison-Wesley Publishing Company, 1992.

[Segal et al. 1992] Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., Haeberli, p., “Fast Shadows and Lighting Effects using Texture Mapping.”, Proc. SIGGRAPH, volume 26, pp 249-252, July 1992.

[Wanger et al. 1992] Wanger,L.R. Ferwerda,J.A. Greenberg, D.P. "Perceiving spatial relationships in computer-generated images", IEEE Computer Graphics and Applications, 12(3), pp.44-58, 1992.

[Whitted 1980] Whitted, T., “An Improved Illumination model for Shaded Display”, Communications of the ACM, Volume 32, number 6, June 1980, pp. 343-349.

[Williams 1978] Williams, L., “Casting curved shadows on curved surfaces.”, Proc. SIGGRAPH, volume 12, pp 270-274, August 1978.