The Deferred Accumulation Buffer

Patrick S. McCormick
Los Alamos National Laboratory

Charles Hansen
The University of Utah

Edward Angel
The University of New Mexico

Abstract. A well-known disadvantage of using the Z-buffer algorithm for medium-quality rendering is the overhead associated with shading, shadowing, and texturing pixels that do not contribute to the final image. This problem may be avoided by deferring shading calculations until the end of the rendering pipeline. In this paper, we review two approaches to deferred shading, discuss how to handle transparency, and then extend these ideas to implement a deferred accumulation buffer.

1. Introduction

In both low-quality rendering using Gouraud shading, and medium-quality rendering where more complex shading algorithms and texture mapping are used [Haines, Worley 96], shading requires a significant portion of the computation time for each displayed pixel. Typically, Z-buffer algorithms are used for both low- and medium-quality rendering. There are three main drawbacks of using the Z-buffer: its memory footprint, unnecessary calculations, and aliasing artifacts. As is evident by the large number of commodity graphics cards with hardware Z-buffers, the memory footprint of the algorithm is not as critical today as it was in the past. Even though processors have made incredible advances in both cost and performance, the wasted computations
introduced by the algorithm can still have a significant impact on rendering times. These wasted calculations are especially evident in medium-quality rendering where more realistic methods, such as Phong shading, texture mapping, and shadows are applied. One well-known approach for removing this overhead is to delay shading calculations until it is known which pixels contribute to the final image. In Section 2, we shall see how using this technique, commonly referred to as deferred shading, can dramatically improve rendering performance. In Section 3, we shall discuss the impact that transparency has on deferred shading. There are several possible solutions for overcoming the aliasing artifacts introduced by the point sampling of the Z-buffer. One method is to collect multiple samples through the use of an accumulation buffer. In Section 4, we shall extend the ideas of the accumulation buffer to include deferred computations.

2. Deferred Shading

When using Phong shading, texture mapping, and shadows it is common for a renderer to spend the majority of its computation time in the shading portion of the pipeline. As an example of this characteristic, Figure 1 shows a

![Rendering Pipeline Performance](image)

**Figure 1.** Shading calculations play a very significant role in the overall rendering time.

1In this paper we will refer to the shading stage of rendering such that it includes both texture mapping and shadow calculations.
comparison of the execution times for different portions of the rendering pipeline given a scene composed of randomly placed triangles; note that shading calculations are occurring before Z-buffer tests in this pipeline. Figure 2 shows two of these test images. Note the increase in shading time as more and more triangles are added to the scene. For reference, on a 250 MHz MIPS R10000 processor, 1,000 triangles take approximately 0.16 seconds to shade while 131,000 triangles take approximately 18.7 seconds. This time corresponds to about 69% of the total rendering time. In most cases, especially in scenes with high depth complexity, a large portion of this time is spent shading pixels that are not visible in the final image. For the details of how the depth complexity of a scene grows, and thus how the occurrence of reshadings increases, see [Cox, Hanrahan 93].

The basic ideas behind deferred shading have been in use for years [Cook et al. 87], [Glassner 88], [Greene, Kass 93], [Haines, Worley 96], [Perlin 85]. Several projects have also considered the advantages of using deferred computations in hardware [Deering et al. 88], [Eyles et al. 97], [Tebeles et al. 92]. Performance measurements and several of the topics presented in this paper are based upon the results given in [McCormick 96]. Both the two-pass and single-pass deferred shading algorithms presented in this section are not new but are presented here for completeness. In the following subsections, we review two approaches to deferred shading and discuss the advantages and disadvantages of each.

---

2We define high depth complexity as depth complexity greater than 2.
2.1. Two-Pass Deferred Shading

The approach described by Haines and Worley [Haines, Worley 96] is a two-pass algorithm that can be summarized as follows:

1. Render the scene once into the Z-buffer without performing any shading computations.

2. Render the scene a second time. For \( z \) values that are identical to those computed in the first pass, compute shading information.

A clear advantage of this algorithm is that it requires only a minimal modification to existing code and does not increase the memory footprint of the renderer.\(^3\) A clear disadvantage of this approach is that it requires two traversals, including scan conversion and Z-buffer assignment, of the scene database. Figure 1 shows that this cost, \( \text{Geom} + \text{Scan} + \text{Z-Buffer} \), is about 30\% of the total time and therefore the two-pass approach still provides substantial benefits.

2.2. Single-Pass Deferred Shading

A single-pass approach to deferred shading can be implemented by storing deferred pixels in the Z-buffer and then shading the visible deferred pixels in a final pass at the end of the pipeline. In this case, a deferred pixel must contain all the information necessary for the final shading calculations.\(^4\) For example, consider the case of Phong shading. To shade a pixel correctly, we must store (at least) a surface normal and material properties. The actual memory requirements for a deferred pixel are dependent upon the features supported by a particular renderer. For example, deferred shading with texture mapping requires storing texture coordinates in a deferred pixel. In practice, we have found the cost of the deferred Z-buffer to be approximately four to six times larger than the requirements of the standard algorithm.

Even though the memory footprint of the single-pass algorithm is significant, it does have some advantages over the two-pass approach:

1. For extremely large polygon counts the single-pass approach performs better. As noted above, the cost of the database traversal is minimal so the differences in performance are often very small.

\(^3\)See the Haines and Worley paper [Haines, Worley 96] for more details on saving memory using this approach.

\(^4\)This approach is similar to the approach used in the hardware of the Pixel Flow machine [Eyles et al. 97].
2. Because shading operations occur in a tight loop at the end of the pipeline, the cache performance on modern architectures tends to be better for the single-pass. In some cases, this advantage has led to deferred shading actually outperforming the standard algorithm (by a small margin) on scenes with little or no depth complexity.

3. Certain advanced features, such as transparency and accumulation buffer techniques, are often easier to implement, or perform better, when using the single-pass algorithm.

The clear disadvantage of this approach is the extra memory space required by the deferred pixel Z-buffer. In addition, extra overhead is introduced by the increase in the number of assignment operations that must occur at each pixel in the deferred Z-buffer. Our performance studies have shown that the assignment overhead of the single-pass algorithm is similar to the cost of the extra pass required by the two-pass algorithm. Therefore, the performance of both algorithms is very similar and both have a substantial performance advantage over the traditional approach as we show in the next section.

2.3. Performance

Figure 3 shows the rendering times for the random triangle data set when using single-pass deferred shading. Comparing these results with Figure 1, the shading time is only slightly faster than with the standard Z-buffer for 1,000 triangles but is significantly faster, 0.7 seconds, for 131,000 triangles. The resulting speed-up in shading time of 1.02 and 26.43 over the standard approach has reduced shading time to 44% and 7% respectively. We can observe from the graph that the shading time eventually reaches a constant value once the entire view is covered by nonbackground pixels (i.e., the depth complexity is greater than 1 for all pixels). Notice that the Z-buffer time is slightly higher in the deferred case, due to the bookkeeping required for deferred shading. With the introduction of deferred shading, geometry processing, scan conversion, and z-buffering dominate the rendering time.

With the introduction of greater realism, such as texture mapping and shadows, the cost of shading can increase dramatically. In this situation, deferred calculations provide an even greater benefit. Figure 4 shows a comparison of the traditional and deferred approaches for a texture mapped version of the random triangle data set. In this case, the deferred approach is significantly faster for 131,000 triangles.

Even with the improvements in performance, the fact that the deferred algorithms are still using the basic Z-buffer algorithm can introduce problems in supporting some desirable features. In the next section, we consider the
Figure 3. Single-pass deferred shading rendering times.

case of handling transparency correctly in a Z-buffer-based renderer and the impact it has on deferred shading.

3. Transparency

Both the single-pass and two-pass deferred algorithms, like the standard Z-buffer, have difficulties supporting transparency. In practice, two common solutions are used. The first is to presort all polygons into depth order, and the second is to use screen-door transparency. Both of these approaches are prone to errors. In this section, we consider one possible alternative for supporting transparency.

In order to handle transparency correctly without presorting of geometry, we can use a modified A-buffer technique [Carpenter 84] which we call a deep Z-buffer; this technique is also similar to the visible point list used in [Cook et al. 87]. At each location in the buffer, the deep Z-buffer stores a depth-sorted list of visible pixels. This strategy is similar to an A-buffer but without the coverage masks. Next, we must modify the insertion rules of the Z-buffer to take not only z values into consideration but also the opacity of both the incoming pixel and the pixels already present in the buffer. For example, there is no need to store any pixels behind an opaque pixel. Finally the last step of the process is to make a pass through the buffer and composite the pixels in each location to produce the final result. The compositing approach we use is
the standard front-to-back method based upon the pixel algebra introduced by Porter and Duff [Porter, Duff 84].

By storing lists of deferred pixels in the deep Z-buffer, we can take advantage of the single-pass algorithm at the cost of increasing both the memory usage and the time required for the assignment of pixels into the lists. Using this technique, we can shade and composite during the final pass. We avoid unnecessary calculations by shading and compositing until either the entire list has been processed or the resulting pixel has become opaque.

While the introduction of transparency can increase the number of objects shaded at each pixel, deferred shading still provides performance improvements. For the random triangle data set, in which approximately one half of the triangles are transparent and one half are opaque, the deferred approach runs slightly slower than the standard method (see Figure 5). The slowdown is due to managing the pixel lists using a simple insertion sort-based algorithm. By replacing the insertion sort with a smarter algorithm, which only sorts the known visible pixels during the shading pass, run time for managing the lists can be reduced from $O(n^2)$ to $O(n \log n)$.

Although it is possible to use a similar approach with the two-pass algorithm, the arbitrary pixel order supported by the Z-buffer can lead to implementation difficulties and reduced performance. In the next section, we extend the application of deferred pixels for accumulation buffer techniques.

---

5As of this writing, this approach has not been implemented.
4. The Deferred Accumulation Buffer

The accumulation buffer provides many different rendering enhancements [Haeberli, Akeley 90]; one of these is antialiasing. Due to the point sampling that is inherent in the Z-buffer algorithm, the algorithm is very susceptible to aliasing errors. The idea behind antialiasing with the accumulation buffer is to render a scene multiple (N) times, with a slight subpixel offset for each pass. The resulting image from each pass is summed into the accumulation buffer. The final image is then obtained by averaging the summed pixel values stored in the accumulation buffer.

Given the multipass nature of the algorithm, and the performance results presented in Section 2.3, it is clear that by using the deferred rendering pipeline we can improve the performance independently for each of the N accumulation passes. While this approach provides a performance improvement for each pass, it does not take full advantage of using deferred shading. Building upon the single-pass approach to deferred shading, we can create a deferred accumulation buffer by storing deferred pixels in the accumulation buffer. With this approach, we can delay shading calculations until all N passes have been completed. The overall approach can be summarized as follows:

1. Clear the deferred accumulation buffer.

2. Render the scene by using the (single-pass) deferred pipeline up to the point of shading the deferred pixels. This step is equivalent to a deferred Z-buffer algorithm, as described in the previous sections.
3. Add the resulting deferred Z-buffer samples into the deferred accumulation buffer, taking care to combine only similar surfaces. This step is described in more detail below.

4. If we have completed $N$ passes, continue to the next step. Otherwise, jitter the scene by two randomly generated subpixel offsets and return to Step 2.

5. Average and shade the results stored in the deferred accumulation buffer, as described below, to produce the final result.

To complete this algorithm, we must define the specifics of an add operation for the components of a deferred pixel. For each distinct material within an object, we simply average all incoming surface properties. For example, surface normals are summed and the result is normalized before shading. Due to the subpixel offsets used during the $N$ rendering passes, it is very likely that for a given pixel, in the deferred accumulation buffer, different passes will result in samples that represent different materials. This situation occurs along polygon edges. Consequently, to produce reasonable results, it is necessary to keep a list of distinct materials for each object at each pixel in the deferred accumulation buffer. Thus, each node in the list is used to represent pixels with different material properties and/or objects. During the add operation, deferred pixels (samples) with identical surface properties are summed together and deferred pixels with a new (unseen) surface value cause a new node to be inserted into the list. Each node in the list is also given a hit count which represents the number of times a sample with the same surface property has been encountered.

The shading pass involves averaging the values stored in the deferred accumulation buffer followed by shading calculations. Note that for pixels containing differing materials in the deferred accumulation buffer (where more than one node is stored on the linked list), the shaded contribution of each node is weighted by the number of hits it has received.

The disadvantage of this approach is that it is only an approximation of the standard accumulation buffer. For example, because shading occurs only once per material, this technique lacks the ability to antialias specular highlights. Despite this shortcoming, for medium-quality rendering the deferred accumulation buffer offers reasonable results with fewer computations. When high-quality rendering is desired, a different rendering system can be employed. Figure 6 shows a visual comparison of the results. Notice the antialiasing on the grill, bumper, and wheels.

The deferred accumulation buffer can also be used for other accumulation buffer techniques such as motion blur. Figure 7 shows the results of a 34-sample motion blur with the geometry not moving for the last two samples.
Figure 6. Comparison of aliased and antialiased results. The image on the left shows aliased results, the middle picture shows the results of the accumulation buffer, and the right picture shows the results of the deferred accumulation buffer.

Figure 7. Results of a motion blur effect using the deferred accumulation buffer.

4.1. Performance

We now compare the performance of the accumulation buffer using standard and deferred Z-buffer renderers with the deferred accumulation buffer. We begin by studying the differences between using the standard and deferred pipelines.

Based upon the results from Section 2.3, it is clear that using the deferred algorithms provides substantial savings in rendering times. We expect speed-up results nearly identical to those measured in Section 2.3 because both approaches use identical accumulation buffer operations and therefore the savings are directly related to the speed-ups achieved by deferred shading. Based upon this finding, we can predict the savings introduced by the deferred accumulation buffer.

The deferred accumulation buffer introduces savings by making only a single shading pass, instead of the $N$ passes required by the other two approaches. Because the deferred pixel's sum and average operations are more expensive, they are key to the performance of the overall algorithm. To have better performance, the deferred accumulation buffer operations must cost less than the $N - 1$ extra shading passes used by the standard accumulation buffer. The performance results show that this is indeed the case, but only by a small amount. For 131,000 polygons, the multipass deferred approach renders approximately 1.97 times faster than the standard approach. The deferred
accumulation buffer renders approximately 2.28 times faster than the standard approach. While the savings are fairly modest, the deferred accumulation buffer method shows improvements over the brute-force multipass method.

Acknowledgements. Thanks to Jamie Painter, James Ahrens, Alan Keasley, and Allen McPherson for reviewing early versions of this work and to the final reviewers for their suggestions and comments. We acknowledge the Advanced Computing Laboratory at Los Alamos National Laboratory. Portions of this work were performed on computing resources located at this facility.

References


Web Information:

http://www.acm.org/jgt/papers/McCormickHansenAngel99

Patrick S. McCormick, Los Alamos National Laboratory, Advanced Computing Lab, MS B287, Los Alamos, NM 87545 (pat@acl.lanl.gov)

Charles Hansen, The University of Utah, Computer Science Department, 3190 Merrill Engineering Building, Salt Lake City, Utah 84112 (hansen@cs.utah.edu)

Edward Angel, The University of New Mexico, Computer Science Department, Albuquerque, New Mexico 87131 (angel@cs.unm.edu)

Received September 29, 1999; accepted January 16, 2000.