Vol.1, No. 3: 1–6

Fast, Low Memory Z-Buffering when Performing Medium-Quality Rendering

Eric Haines
3D/EYE Inc.

Steven Worley
Worley Laboratories

Abstract. This article presents algorithms which both improve performance and decrease memory costs when using a Z-buffer for medium-quality rendering. The crux of the method is to perform rendering in two passes; the first quickly renders only Z-depth values, the second does all shading calculations. This method allows the reuse of memory used to store the Z-depths and colors, as only one of these two values is needed at any given moment for any given pixel. It also eliminates all unnecessary shading/shadowing/texturing calls, which typically take the majority of computation time in medium-quality algorithms.

Medium-quality rendering is easiest to define by stating what we consider low and high quality. Low quality is simply using Gouraud interpolated polygons, where the shade is computed at the vertices; this type of rendering is extremely rapid and has been migrating for years into special purpose hardware on many platforms. High quality includes subpixel algorithms such as the A-buffer [Carpenter 84] and antialiased ray tracing, where more than one sample per pixel is generated. Medium quality is, then, where the shading computation is a significant part of the process and only one sample per pixel is generated. By this definition the accumulation buffer algorithm [Haeberli, Akeley 90] is a medium-quality rendering method, since each separate pass uses a single sample per pixel.

1086-7651/96 $0.50 per page

1. Saving Time

The major expense of medium-quality Z-buffering is the shading calculation itself. This calculation can include texture evaluation, shadow computation, local shading, and ray tracing for reflections and refractions. This computation is wasted when a pixel color is carefully evaluated, only to be overwritten by a subsequent polygon. This cost is a well-documented problem [Watt, Watt 92], but surprisingly we have not found our solution in any text. We suspect others have independently discovered and used this method for years; we present it here for the benefit of everyone else.

The solution is simple: render the scene once into the Z-buffer, performing no shading, then render the scene again, shading only at pixels where the surface is known to be visible. The first pass fills the Z-buffer with the depths of the closest objects. When the second pass is performed, no pixels are shaded which are not the closest, i.e., only visible surfaces are shaded. The additional cost of this method is that the scene database is traversed and Z-depths are computed twice. However, the expense of computing Z-depths of pixels (often with a tight loop using only integer additions and compares) is usually trivial compared to the expense of texturing, shading, shadowing, and recursive ray calls.

The two-pass method attempts to minimize the wasted pixel computation at the expense of an extra (nonshaded) traversal. The major factor that determines the savings of this technique is the average depth complexity of the scene. As more surfaces overlap, the efficiency increases. Note the similarity of this algorithm to pure ray tracing, in which the first goal is to find the closest object along a ray, then the shade of this closest object is computed.

One of the advantages of this algorithm is that it is simple to implement. A Z-buffer renderer already has all of the code; make a simple Z-only renderer and add this extra rendering pass and we are done. In fact, a Z-only renderer may already exist, as it may be used for shadow buffer creation.

There are a few details that should be noted. The computation of the Z-depths must be absolutely identical for both passes, otherwise the second (rendering) pass will not shade the pixels where the Z-depths do not agree. The second pass essentially looks for when the Z-depth matches, which does mean that once in a great while a pixel's shade will be computed twice because two polygons are at the same depth (as it turns out, such minor waste is eliminated by the memory-saving scheme described later in this article).

Transparency is another issue which must be considered. A pixel where transparency occurs by definition has a few different objects affecting the shade. For a hybrid renderer which uses ray tracing for refraction there is no problem, since a ray is traced to compute the effect. Similarly, a transparency scheme in which the transparent object is rendered by shading, say, alternating pixels in a checkerboard pattern, avoids the problem since only one object is

stored at each pixel. For other algorithms other solutions are necessary, e.g., the Z-depth of only opaque objects might be stored.

It is worth noting that this basic algorithm can be used selectively. If the object is deemed sufficiently simple, it can be fully rendered into the color and Z-buffer in the first pass; if it is more complex, it is rendered in two passes.

2. Saving Memory

Given this two-pass method, there is a way to halve the standard memory costs for image computation. Normally separate color and Z-buffers are allocated. Using this new scheme only one buffer is allocated and reused for both types of data. This process is done by reducing the range of Z-depths slightly, allowing some bits to flag whether the memory stores a color or a Z-depth. Additionally we can identify which pixels show the background, for use in assembling a transparency bit for compositing. Separate bit arrays could be used instead [Libes 93], but our scheme results in less time spent in memory access.

One example of this technique assumes a 32-bit, fixed-point (integer) Z-buffer. Traditionally, the range of depths will range over the full values (in hex) from 00000000 to FFFFFFFF. However, in the reuse scheme, we will "reserve" the bottom end of this range, along with the highest value, and map the depths from 01000000 to FFFFFFFE. Note that this is a reduction of the Z-range of well less than 1 percent (i.e., only about 1/256) so the resolution of the Z-buffer is barely affected.

Initialize each Z-value to FFFFFFFF for the first pass, which computes only Z-depths. As in a normal Z-buffer, we scan convert geometry into the buffer, replacing a Z-value in the buffer with the geometry's if the geometry is closer. After this first pass is complete, we have a depth map of the scene, with the background buffer values still set at FFFFFFFF.

Now we traverse the geometry again, scan converting it a second time. If the geometry's depth matches the stored depth, compute the proper color for that pixel, including any texturing, shading, shadows, etc., which will result in an RGB color. Now store that RGB color in the buffer by concatenating the value "00" with the R, G, and B byte values to produce a 32-bit value. Store this value into the Z-buffer. Since no object will have a Z-depth less or equal to 00FFFFFF (the "maximum" color value), this pixel will never be overwritten by subsequent geometry.

After the second pass is complete, the buffer can be examined for pixels which contain FFFFFFFF, as these are background pixels. The background pixels' colors can then be computed and stored in the buffer. At this point the first byte is otherwise unused (all Z-depth comparisons have been done), so we could leave it as "FF" to denote that it is a background pixel.

When the rendering process is complete, the buffer contains the proper color values for each pixel, stored as the second through fourth bytes in each 32-bit buffer value. The transparency flag is stored in the first byte, with FF for background and 00 for foreground.

3. Storage of Higher Quality Colors

A higher quality renderer may require a better color representation than a single byte for R, G, and B. One solution would be to store the color values in a nonlinear form which still uses only 24 bits, as in [Schlick 94]. However, the ability to use even more bits to represent each color will obviously help. More bits per pixel is one option; another possibility is a floating point color representation such as in [Ward 91].

A 32-bit Z-value is enormously precise; the rendering of a large room with a far clipping plane of 10 meters implies the Z-buffer values are accurate to approximately 1 angstrom (10^{-10} meters), comparable to the diameter of an atom. This result implies that it is quite safe to use less than a full 32 bits of precision. [Lathrop et al. 90]

A simple scheme allowing greater color precision storage is to steal a high bit from the Z-buffer and use it as a flag bit. Setting this flag bit when initializing the Z-buffer, and shading only when this flag bit is set, allows full memory reuse. This flag bit is cleared when the pixel is shaded. This algorithm provides 31 bits for color storage, allowing colors of 10+ bits per channel, or with a 7-bit exponent for Ward's color storage method. Any pixel with the high bit set after both passes is a background pixel.

4. Summary

By performing two passes when doing per pixel shading, wasted shading computation can be avoided. The Z-buffer can also be reused to store the final pixel colors, saving significant memory.

References

[Carpenter 84] L. C. Carpenter. "The A-buffer, An Antialiased Hidden Surface Method." *Computer Graphics*, Proc. SIGGRAPH '84, 18(3):103–108(1984).

[Haeberli, Akeley 90] Paul Haeberli and Kurt Akeley. "The Accumulation Buffer: Hardware Support for High-Quality Rendering." *Computer Graphics*, Proc. SIGGRAPH '90, 24(4):309–318(1990).

[Lathrop et al. 90] Olin Lathrop, David Kirk, and Doug Voorhies. "Accurate Rendering by Subpixel Addressing." *IEEE Computer Graphics and Applications*, 10(5):45–53(September 1990).

[Libes 93] Don Libes. *Obfuscated C and Other Mysteries*, pp. 25–29. New York: John Wiley & Sons, 1993.

[Schlick 94] Christophe Schlick. "High Dynamic Range Pixels." In *Graphics Gems IV*, edited by P. Heckbert, pp. 422–429. Boston: Academic Press, 1994.

[Ward 91] Greg Ward. "Real Pixels." In *Graphics Gems II*, edited by J. Arvo, pp. 80–83. Boston: Academic Press, 1991.

[Watt, Watt 92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques*, p.22. Reading, MA: Addison-Wesley, 1989.

Web Information:

Eric Haines, 3D/EYE Inc., 1050 Craft Road, Ithaca, NY 14850 (erich@acm.org)

Steven Worley, 405 El Camino Real, Suite 121, Menlo Park, CA 94025
(steve@worley.com)

Received April 19, 1996; accepted November 11, 1996