HARDWARE ASSISTANCE FOR Z-BUFFER VISIBLE SURFACE ALGORITHMS

Kellogg S. Booth, David R. Forsey, and Alan W. Paeth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: KSBooth%watCGL@Waterloo.CSNet

ABSTRACT

The well-known "z-buffer" algorithm for solving the visible surface problem has a number of points in its favor, the main one being that it is amenable to very efficient hardware implementation at little additional cost in many existing frame buffer systems. The traditional software implementation of the algorithm assumes explicit initialization of both the image buffer and the z-buffer before each image is generated. This paper describes a simple technique for synchronizing initialization and image generation so the two can be performed in parallel, allowing complete overlap to be achieved and effectively eliminating the time needed for explicit initialization of the frame buffer. The technique assumes a modest investment in additional hardware within the frame buffer.

RÉSUMÉ

Parmi les algorithmes de surfaces cachés, l'algorithme du "z-buffer" a un certain nombre d'avantages à son actif. Le plus important de ceux-ci étant que cet algorithme est peu couteux à implémenter au niveau hardware dans les systèmes actuels de "frame-buffer" Les techniques traditionnelles d'implémentation de cet algorithme forcent le logiciel à initialiser explicitement le "z-buffer" et le buffer image avant le transfert de l'image. Cet article décrit une technique simple qui permet de synchroniser l'initialisation et la création de l'image afin qu'elles puissent être réalisées simultanément. Ceci permet d'éliminer le temps perdu lors de l'initialisation du "frame-buffer." Cette technique suppose un faible investissement en matériel additionnel à l'intérieur du "frame-buffer."

Keywords: *double-buffering, frame buffer, real-time, visible surfaces, z-buffer.*

INTRODUCTION

The problem of computing only the *visible surfaces* of a scene has by now been well-studied [12]. One approach that has gained popularity is the *z-buffer* algorithm first described by Catmull [4,5]. With a z-buffer the depth-sort required of a visible surface algorithm is accomplished by maintaining, for each pixel, a record of the *z-depth* of the object whose intensity (color) is stored at that pixel. As subsequent objects are scan converted into the frame buffer, their z-depths are compared and used to decide whether the new object is in front of or behind the object currently displayed at each pixel. In the former case both the intensity and z-depth are changed for the pixel, but in the latter case no action is taken. A complete explanation of the z-buffer algorithm appears in standard text books on computer graphics [7,10].

In the following sections we first define our model of a frame buffer and then look at different ways of adding additional hardware to the frame buffer to speed up the z-buffer algorithm. The first approach almost doubles the amount of memory in the frame buffer and is presented solely to motivate the other two. The second approach adds only a single bit to each pixel but requires a more complicated memory controller that could lead to significant timing problems in the video chain. The third approach adds two more bits to each pixel and hence admits an implementation that requires no additional function in the memory controller beyond what is available in current frame buffers, although a modest change is still required to hardware further down the video chain.

The basic z-buffer algorithm performs well in almost all respects except for considerations of *antialiasing*. This deficiency stems from the fact that the z-buffer maintains depth information on a pixel-by-pixel basis, and thus has no way to discriminate objects at subpixel resolution. This is unfortunate because aliasing artifacts introduced by the scan conversion process can be very objectionable in practice. Some researchers have suggested techniques to incorporate antialiasing strategies into a z-buffer algorithm, but those techniques either require auxiliary storage or