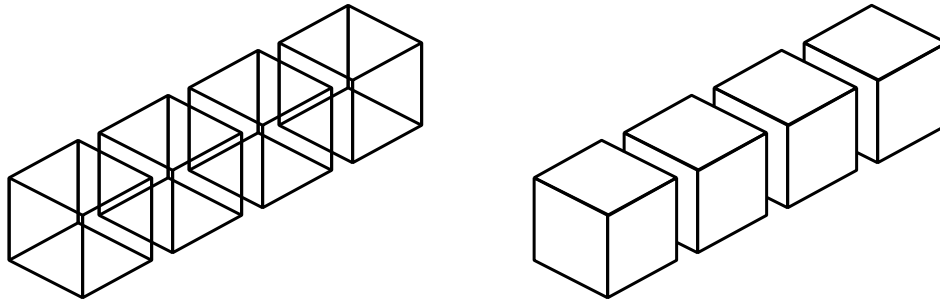


On-Line Computer Graphics Notes

**THE DEPTH-BUFFER  
VISIBLE SURFACE ALGORITHM**

Kenneth I. Joy  
Visualization and Graphics Research Group  
Department of Computer Science  
University of California, Davis

To accurately represent a complex model, it is imperative that those surfaces normally invisible from a certain point, be also invisible in the computer generated image. This problem normally called the visible-surface problem (determining the surfaces that are visible), or the hidden-surface problem (determining those that are invisible), and has been the fundamental research problem in computer graphics over the past 20 years. The effects of this algorithm are easily seen, even in relatively simple pictures. If we consider the following illustration, we can obtain much more information about the relative positions of the objects by using the right hand figure, rather than using the left hand one.



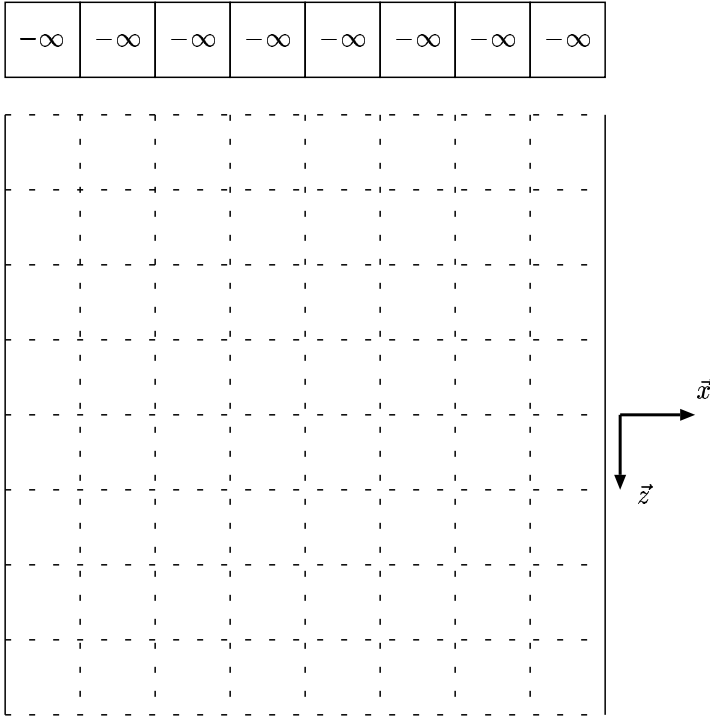
Of all algorithms for visible surface determination, the depth-buffer is perhaps the simplest, and is the most widely used. For each pixel on the display, we keep a record of the depth of the object in the scene that is closest to the viewer, plus a record of the intensity that should be displayed to show the object. When a new polygon is to be processed, a  $z$ -value and intensity value are calculated for each pixel that lies within the boundary of the polygon. If the  $z$ -value at a pixel indicates that the polygon is closer to the viewer than the  $z$ -value in the  $z$ -buffer, the  $z$ -value and the intensity values recorded in the buffers are replaced by the polygon's values. After processing all polygons, the resulting intensity buffer can be displayed

---

The z-buffer algorithm works in device space and it can be easily implemented as a modification of the scan-conversion algorithms that have been discussed in previous sections. The z-buffer, from which this algorithm derives its name, is an  $n \times n$  array for which the  $(i, j)$ th element corresponds to the  $(i, j)$ th pixel. This array holds the image-space  $z$  value of the currently visible object at the pixel. There is also another  $n \times n$  array whose elements correspond to the color that is to be assigned to the pixel.

We illustrate the operation of the z-buffer algorithm by considering a two-dimensional example, with an eight-pixel-wide screen and polygons represented as lines. The steps to perform the algorithm are nearly identical in three dimensions.

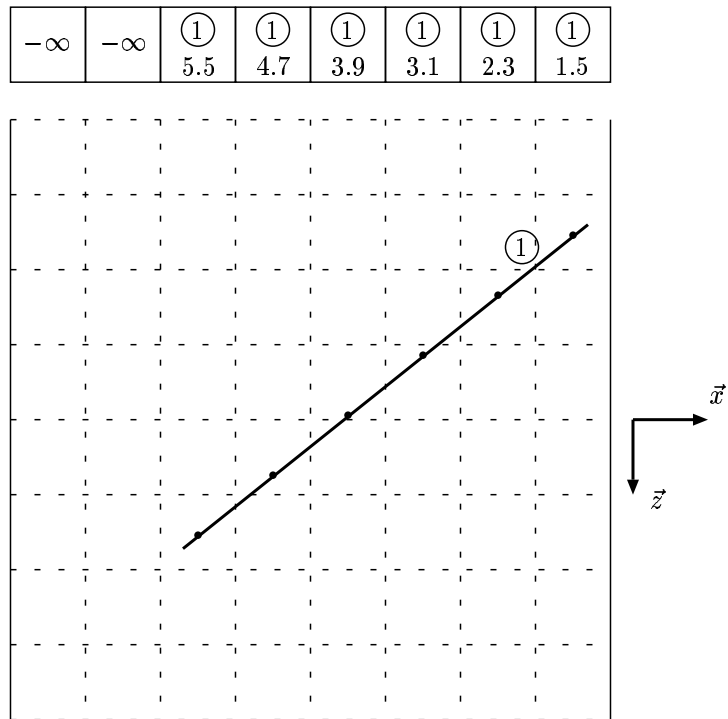
- Initially, since we have rendered no polygons yet, we set all elements of the z-buffer array to  $-\infty$  (actually, some suitably large negative number). We also initialize the color array to black (or suitable a background color).



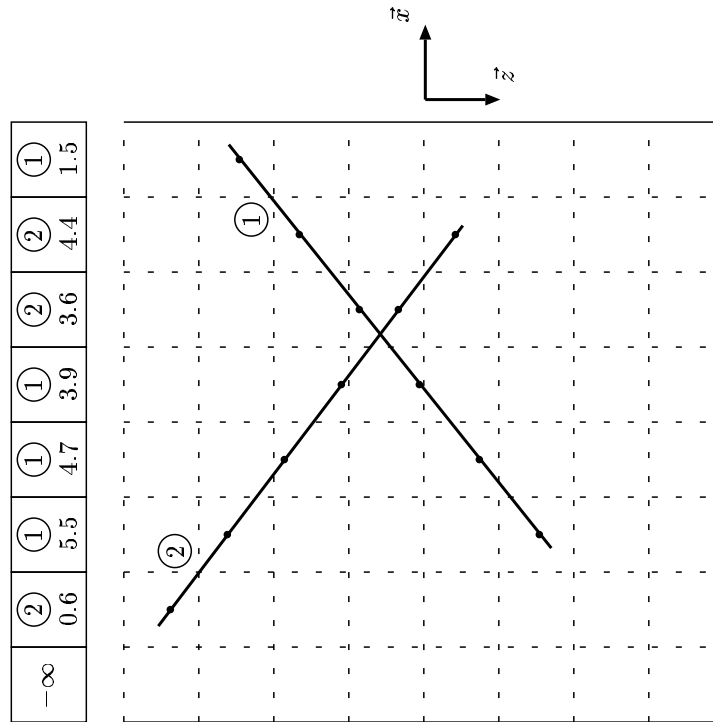
- Taking each polygon, scan convert the polygon and, within the scan-conversion algorithm, perform the following steps

- For each pixel that intersects the polygon, if the image-space  $z$  value associated with this pixel is greater than the corresponding value in the  $z$  buffer, then calculate the color associated with the pixel, insert this color into the corresponding entry in the color array, and insert the pixel's  $z$  value into the corresponding element in the  $z$  buffer. If the  $z$  value associated with this pixel is not greater than the corresponding value in the  $z$  buffer, then no action is taken.

As an example, inserting one line into our two-dimensional  $z$ -buffer, we log the depth value at the center of each pixel, and obtain



Inserting a second line in the same way, we note that the  $z$ -buffer has been accurately updated to indicate the line that is “visible” at each pixel



We can give a simple pseudocode implementation of the z-buffer algorithm as follows:

### Z-Buffer Algorithm

**Given**

List of polygons  $\{P_1, P_2, \dots, P_n\}$

An array **z-buffer**[x,y] initialized to  $-\infty$

An array **Intensity**[x,y]

**begin**

**for** each polygon  $P$  in the polygon list **do** {

**for** each pixel (x,y) that intersects  $P$  **do** {

    calculate z-depth of  $P$  at (x,y)

**if** z-depth < **z-buffer**[x,y] **then** {

**Intensity**[x,y] = intensity of  $P$  at (x,y)

**z-buffer**[x,y] = z-depth

    }

  }

}

Display **Intensity** array

**end**

---

Since any polygon can be divided into a set of trapezoids, it is sufficient to consider the z-buffer algorithm on trapezoids. We can directly modify the rasterization algorithm given in the rasterization notes to create the  $z$ -buffer algorithm above. We utilize the same update and initialize procedures as in the rasterization algorithm.

---

**All contents copyright (c) 1996, 1997, 1998, 1999  
Computer Science Department, University of California, Davis  
All rights reserved.**