The S-Buffer: A Scanline Depth Buffer Based on Critical Points Scan Conversion (Class Notes)

Dan Gordon*

1 Introduction

Image rendering in scanline order has many important applications. In these class notes we provide a description of the S-buffer, which is an efficient scanline depth-buffer method, as presented in [1]. In the standard approach to rendering in scanline order [2], a single active edge table (AET) is used for all the polygons. This results in many edge crossings during the sweep stage, requiring resorting at every scanline, even when most of the edges are not visible. In the S-buffer algorithm, each polygon has its own separate AET, so no sorting is required.

Current methods for scanline depth buffers are either restricted (e.g., limited to monotone polygons), or inefficient, since they depend on the standard scan conversion algorithm [2]. The S-buffer is based on the critical points scan conversion method [3, 4, 1], which is more efficient in both time and space. The conversion of the Z-buffer to a single scanline depth buffer can be generalized to any hidden surface algorithm that is based on polygon scan conversion, such as the display of BSP trees – see [1].

2 The Scanline Depth Buffer

We assume familiarity with the details and notations of CP. We now have a set of polygons PSET. Each polygon P in PSET has the following data associated with it:

- Coordinate arrays P.X and P.Y. This is image-space data, after all the necessary transformations on the original 3-dimensional data.
- P.n – number of vertices.
- P.CR – array of critical points.
- P.c – number of critical points.

*Dept. of Computer Science, Univ. of Haifa, Haifa 31905, Israel. Email: gordon@cs.haifa.ac.il
• $P.CC$ – index (in array $P.CR$) of current critical point.

• $P.AET$ – active edge table of polygon $P$. Note that each polygon has its own $AET$.

• Any other data relevant to the application, such as the polygon’s 3-dimensional data, lighting information, etc.

The scanline depth buffer is a 1-dimensional array $S$ whose size is equal to the number of pixels in a scanline. The procedures $DETCR$, $MOVEUP$, $DELETE$, $FILL$ and $POLINE$ now operate exclusively on the data of a polygon $P$, and will be referred to as $P.DETCR$, etc. In addition, $P.FILL$ is modified so that it renders a pixel $(i, H)$ only if its depth is less than the current value of $S[i]$, and the value of $S[i]$ is updated.

The S-buffer algorithm is specified below. The scanline generation of the image is obtained by an outer loop on the scanlines and an inner loop on the polygons which calls on $P.POLINE$ for every polygon $P$. $POLINE$ renders a single scanline of a single polygon and the S-buffer uses it in a dovetail fashion on all the polygons. (Note that a similar approach can be used with standard scan conversion, but at a prohibitive cost, since we either need an edge table for every polygon, or we need to traverse every list of the edge table many times – once for each polygon.)

Outline of the S-buffer Algorithm

```c
/*******************************
{ for (every polygon P) /* initialize P */
    { P.DETCR; P.AET = empty; P.CC = 1; }
 for (every scanline H) /* outer loop on scanlines */
    { initialize the S-buffer to infinity;
        for (every polygon P) P.POLINE; /* inner loop on polygons */
    }
}
```

The above is just a sketch, and details of implementation may vary. A more detailed description is presented below. If the scene is very dense, then the inner loop on the polygons can be done much more efficiently than implied by the above outline. We create a “polygon table” $PT$, with a list for every scanline. Every polygon $P$ is added to the list at line $ceiling(P.Y[P.CR[1]])$ – this is the first scanline at which the polygon becomes active (the list is not ordered, and each polygon is added to just one list). Also, we maintain a list $APT$ (for “active polygon table”) which is a list of all the polygons intersected by the current scanline. A polygon $P$ is added to the $APT$ according to its position in $PT$, and deleted from the $APT$ when $P.AET$ becomes empty.

Procedure S-buffer

```c
/*****************
PT is an array of lists (of polygons) whose size is equal to the number of scanlines. APT is a list of polygons. */

{ /* initialization stage: */
```
LS = y-value of lowest scanline in viewport;
HS = " " highest " " ;
/* initialize PT: */
for ( line = LS to HS ) initialize PT[line] to empty;
for (every polygon P in PSET)
    { P.DETCR; /* determine P's crit. pts. */
      if ( P.c = 0 ) ignore P from further consideration;
      line = ceiling( P.Y[P.CR[1]] );
      if ( line > HS ) ignore P from further consideration;
      if ( line < LS ) line = LS;
      add P to PT[line];
      P.AET = empty;
      P.CC = 1; /* index of P's first crit. pt. */
    }
/* if the polygons are expected to occupy just a restricted
   number of scanlines, LS and HS can be modified according
   to the lowest and highest y-values of the polygons. */
/* sweep stage: */
for ( H = LS to HS )
    { /* outer loop on the scanlines */
      initialize the scanline S-buffer to infinity;
      add PT[H] to APT; /* add new active polygons */
      for (every P in APT)
          { /* inner loop on the polygons */
            P.POLINE;
            if ( P.AET is empty ) remove P from APT;
          }
    }
/* end of S-buffer */

References


