Introduction

Scanline rendering is an algorithm for visible surface determination, in 3D computer graphics, that works on a row-by-row basis rather than a polygon-by-polygon or pixel-by-pixel basis. All of the polygons to be rendered are first sorted by the top y coordinate at which they first appear, then each row or scan line of the image is computed using the intersection of a scan line with the polygons on the front of the sorted list, while the sorted list is updated to discard no-longer-visible polygons as the active scan line is advanced down the picture.

Scan Line Algorithm

As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible.

Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view plane. When the visible surface has been determined, the inensity value for that position is entered into the refresh buffer.

In this algorithm we assume that tables are set up for the various surfaces, which include both an edge table and a polygon table.

The edge table contains coordinate endpoints for each line in-the scene, the inverse slope of each line, and pointers into the polygon table to identify the surfaces bounded by each line.

The polygon table contains coefficients of the plane equation for each surface, intensity information for the surfaces, and possibly pointers into the edge.

The active list will contain only edges that cross the current scan line, sorted in order of increasing ***x.*** In addition, we define a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right.

At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

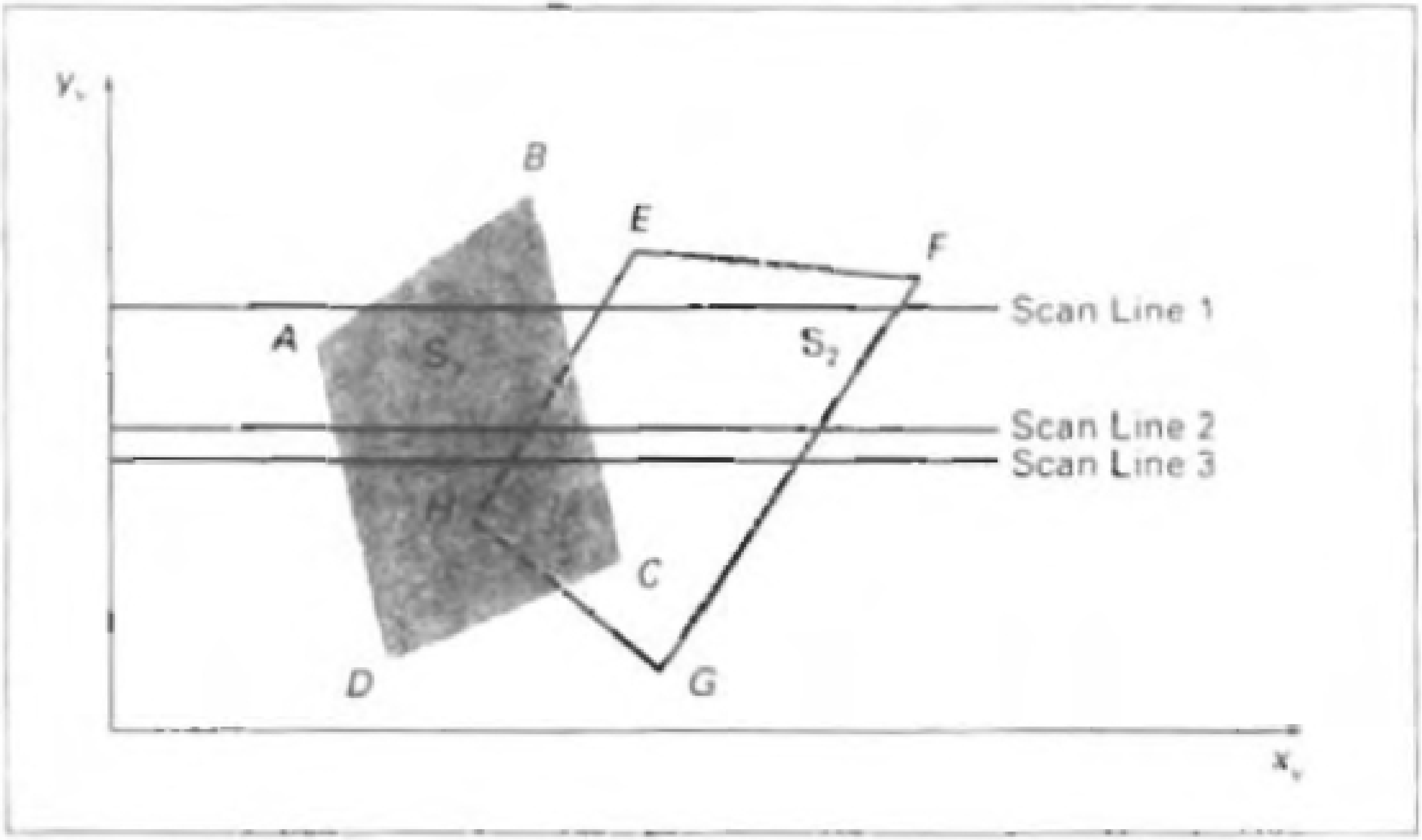


Figure 13-10
Scan lines crossing the projection of two surfaces, S_1 and S_2, in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

The active list for scan line 1 contains information from the edge table for edges ***AB,*** BC, *EH*, and *FG*. For positions along this scan line between edges ***AB*** and BC, only the flag for surface **Sl** is on.Therefore no depth calculations are necessary, and intensity information for surface S1, is entered from the polygon table into the refresh buffer.

Similarly, between edges EH and FG, only the flag for surface S2 is on. NO other positions along scan line **1** intersect surfaces, so the intensity values in the other areas are set to the background intensity.

For scan lines 2 and 3 in Figure, the active edge list conntains edges ***AD,*** *EH*, ***BC,*** and *FG*. Along scan line 2 from edge ***AD*** to edge *EH*, only the flag for surface S1 is on. But between edges EH and ***BC,*** the flags for both surfaces are on.

In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface **SI** is assumed to be less than that of S2, so intensities for surface S1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface **SI** goes off, and intensities

for surface ***S2*** are stored until edge FG is passed.

Coherence property: along the scan lines as we pass from one scan line to the next, scan line 3 has the same active list of edges

as scan line 2. Since no changes have occurred in line intersections, it is unnecessary again to make depth calculations between edges *EH* and ***BC.***

Area Subdivision algorithm

Area coherence: locate those view areas that represent part of a single surface.

Successively divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of ***n*** single visible surface or no surface at all.

Initially divide the area into four equal parts at each step

A viewing area with a resolution of 1024 by 1024 could be subdivided ten times in this way before a.sub area is reduced to a pint.

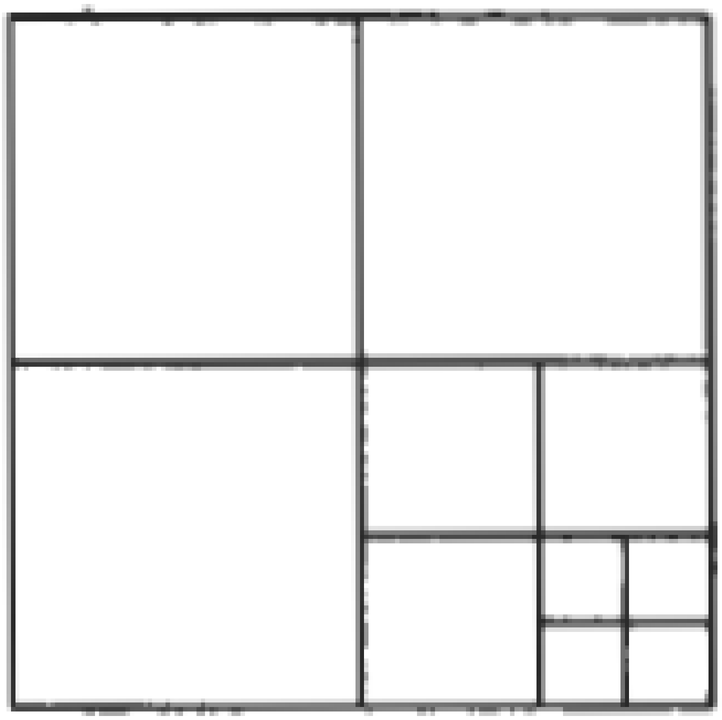
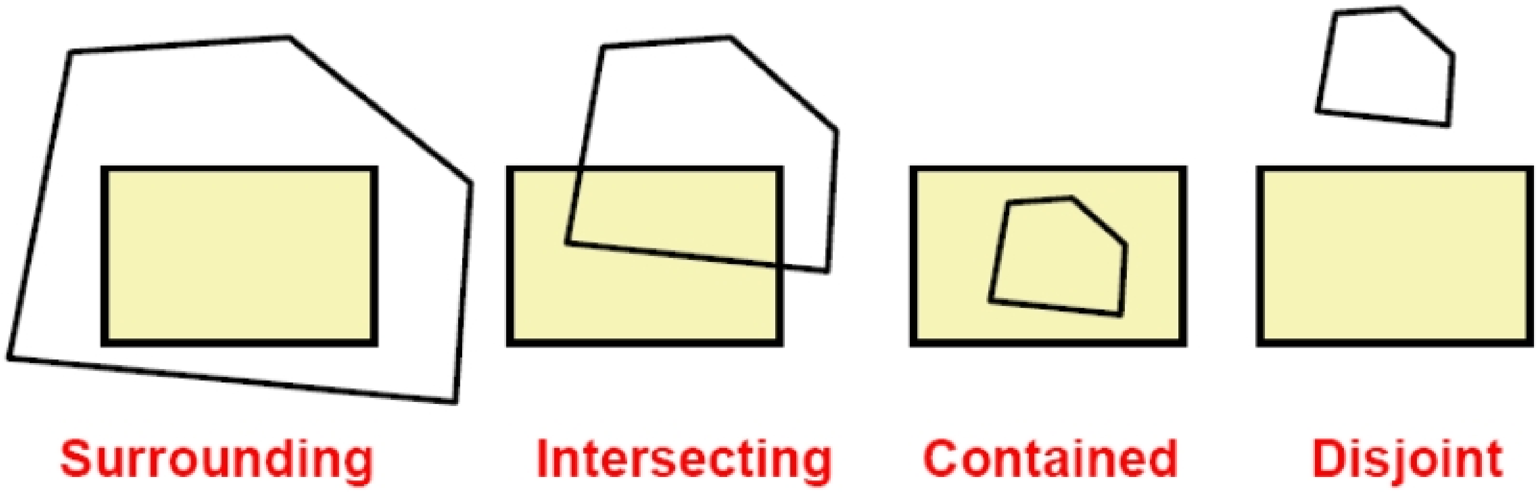


Figure 13-20
Dividing a square area into equal-sized quadrants at each step.

- Each polygon has one of four relationships to the area of interest:



No further subdivisions of a specified area are needed if one of the following conditions is true:

1. All surfaces are outside surfaces with respect to the area.

2. Only one inside, overlapping, or surrounding surface is in the area.

3. A surrounding surface obscures all other surfaces within the area boundaries.

•Test 1 can be carried out by checking the bounding rectangles of all surfaces against the area boundaries.

•Test 2 can also use the bounding rectangles in the xy plane to identify an inside surface.

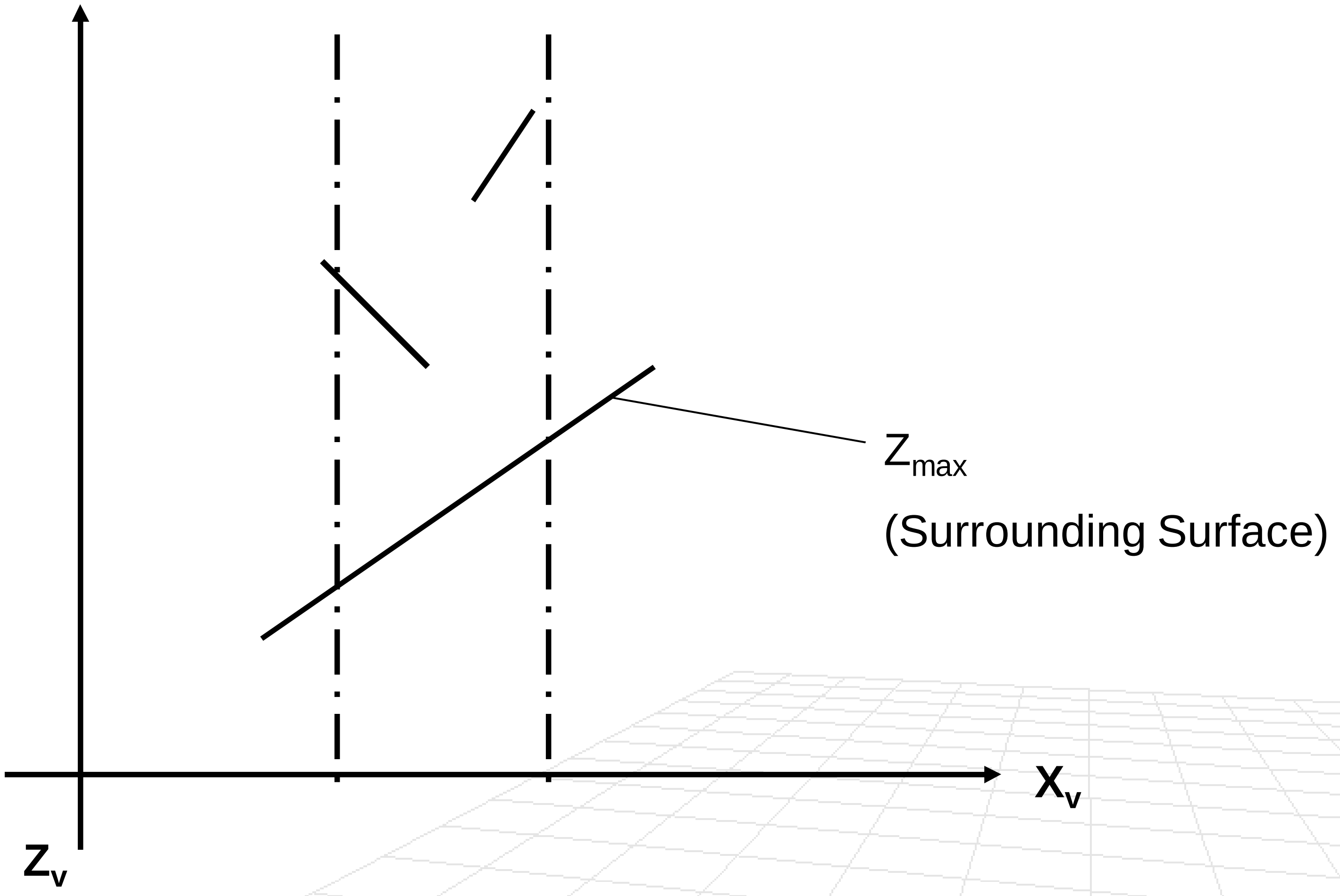
•For other types of surfaces, the bounding rectangles can be used as an initial check. If a single bounding rectangle intersects the area in some way, additional checks are used to determine whether the surface is surrounding, overlapping, or outside.

•Once a single inside, overlapping, or surrounding surface has been identified, its pixel intensities are t r a n s f e d to the appropriate area within the frame buffer.

One method for implementing test 3 is to order surfaces according to their minimum depth from the view plane.

For each surrounding surface, we then compute the maximum depth within the area under consideration.

If the maximum depth of one of these surrounding surfaces is closer to the view plane than the minimum depth of all other surfaces within the area, test 3 is satisfied.



Within a specified area, a surrounding surface with a maximum depth of Z_{max} obscures all surfaces that have a minimum depth beyond Z_{max}.

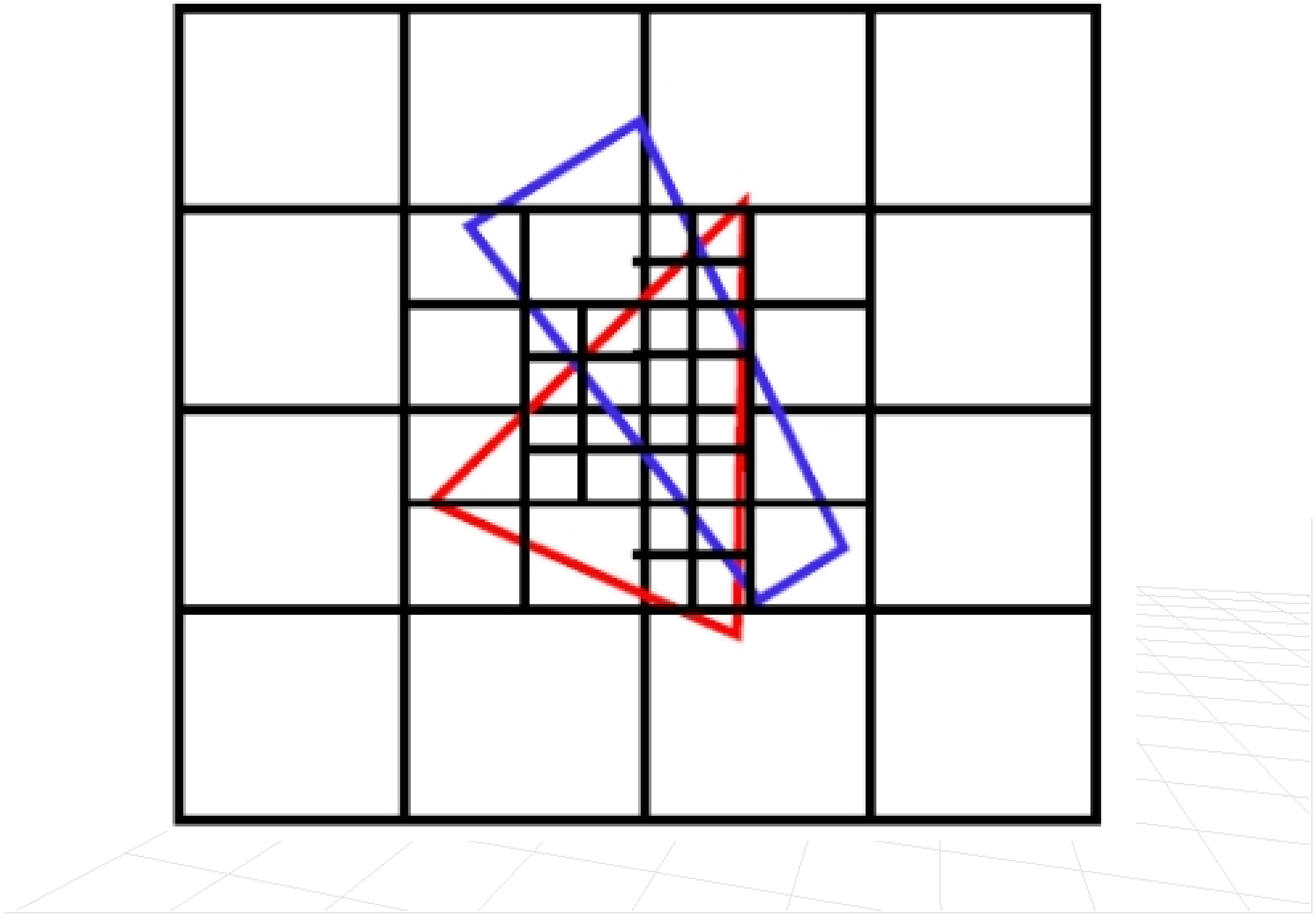
Another method for carrying out test 3 that does not require depth sorting.

It uses plane equations to calculate depth values at the four vertices of the area for all surrounding, overlapping, and inside surfaces.

If the calculated depths for one of the surrounding surfaces is less than the calculated depths for all other surfaces, test 3 is true.

Then the area can be filled with the intensity values of the

surrounding surface.

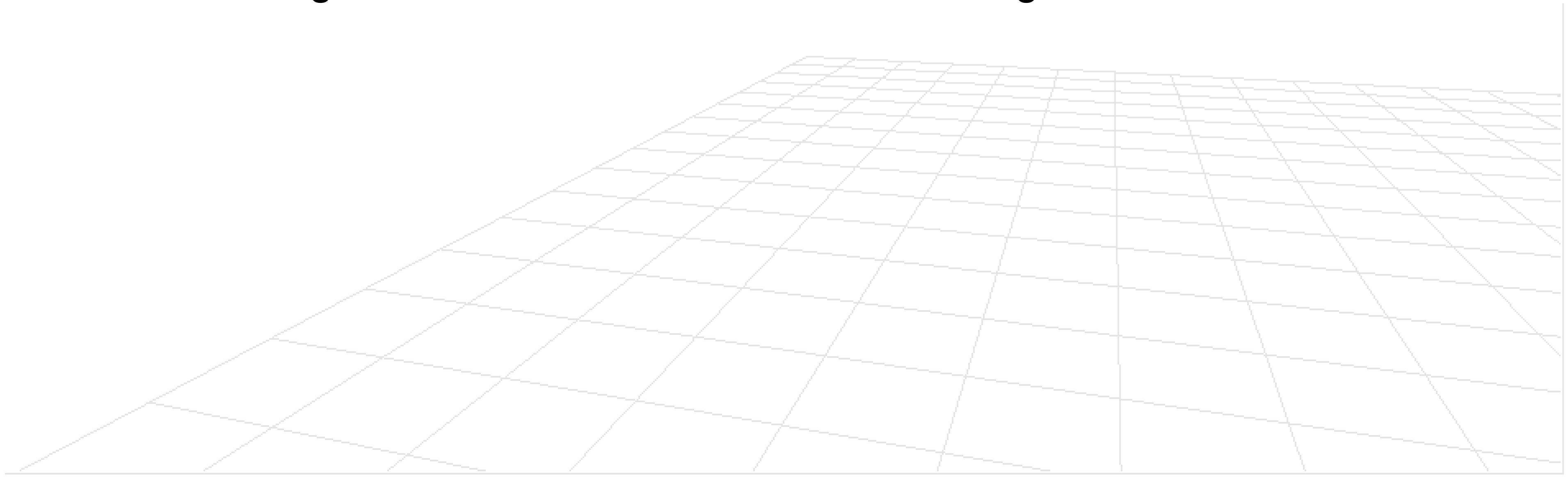


Application

The **Nintendo DS** is the latest hardware to render 3D scenes in this manner, with the option of caching the rasterized images into VRAM.

The **sprite hardware** prevalent in 1980s games machines can be considered a simple 2D form of scanline rendering.

Sony experimented with software scanline renderers on a second Cell processor during the development of the **PlayStation 3**, before settling on a conventional CPU/GPU arrangement.



Scope of Research

The Nintendo DS is the latest hardware to render 3D scenes in this manner, with the option of caching the rasterized images into VRAM.