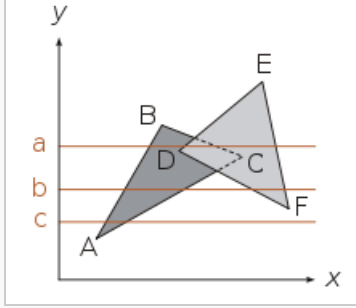
Scanline rendering

From Wikipedia, the free encyclopedia

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 scanline of the image is computed using the intersection of a scanline 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 example

The main advantage of this method is that sorting vertices along the normal of the scanning plane reduces the number of comparisons between edges. Another advantage is that it is not necessary to translate the coordinates of all vertices from the main memory into the working memory—only vertices defining edges that intersect the current scan line need to be in active memory, and each vertex is read in only once. The main memory is often very slow compared to the link between the central processing unit and cache memory, and thus avoiding re-accessing vertices in main memory can provide a substantial speedup.

This kind of algorithm can be easily integrated with many other graphics techniques, such as the Phong reflection model or the Z-buffer algorithm.

Algorithm [edit]

The usual method starts with edges of projected polygons inserted into buckets, one per scanline; the rasterizer maintains an active edge table (*AET*). Entries maintain sort links, X coordinates, gradients, and references to the polygons they bound. To rasterize the next scanline, the edges no longer relevant are removed; new edges from the current scanlines' Y-bucket are added, inserted sorted by X coordinate. The active edge table entries have X and other parameter information incremented. Active edge table entries are maintained in an X-sorted list by bubble sort, effecting a change when 2 edges cross. After updating edges, the active edge table is traversed in X order to emit only the visible spans, maintaining a Z-sorted active Span table, inserting and deleting the surfaces when edges are crossed.

Variants [edit]

A hybrid between this and Z-buffering does away with the active edge table sorting, and instead rasterizes one scanline at a time into a Z-buffer, maintaining active polygon spans from one scanline to the next.

In another variant, an ID buffer is rasterized in an intermediate step, allowing deferred shading of the resulting visible pixels.

History [edit]

The first publication of the scanline rendering technique was probably by Wylie, Romney, Evans, and Erdahl in 1967.[1]

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
Français
Italiano
Suomi
Українська
中文
Edit links

Other early developments of the scanline rendering method were by Bouknight in 1969,[2] and Newell, Newell, and Sancha in 1972.[3] Much of the early work on these methods was done in Ivan Sutherland's graphics group at the University of Utah, and at the Evans & Sutherland company in Salt Lake City.

Use in realtime rendering [edit]

The early Evans & Sutherland ESIG line of image-generators (IGs) employed the technique in hardware 'on the fly', to generate images one raster-line at a time without a framebuffer, saving the need for then costly memory. Later variants used a hybrid approach.

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.

The technique was used in the first Quake engine for software rendering of environments (but moving objects were Z-buffered over the top). Static scenery used BSP-derived sorting for priority. It proved better than Z-buffer/painter's type algorithms at handling scenes of high depth complexity with costly pixel operations (i.e. perspective-correct texture mapping without hardware assist). This use preceded the widespread adoption of Z-buffer-based GPUs now common in PCs.

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.

Similar techniques [edit]

A similar principle is employed in tiled rendering (most famously the PowerVR 3D chip); that is, primitives are sorted into screen space, then rendered in fast on-chip memory, one tile at a time. The Dreamcast provided a mode for rasterizing one row of tiles at a time for direct raster scanout, saving the need for a complete framebuffer, somewhat in the spirit of hardware scanline rendering.

Some software rasterizers use 'span buffering' (or 'coverage buffering'), in which a list of sorted, clipped spans are stored in scanline buckets. Primitives would be successively added to this datastructure, before rasterizing only the visible pixels in a final stage.

Comparison with Z-buffer algorithm [edit]

The main advantage of scanline rendering over Z-buffering is that the number of times visible pixels are processed is kept to the absolute minimum which is always one time if no transparency effects are used—a benefit for the case of high resolution or expensive shading computations.

In modern Z-buffer systems, similar benefits can be gained through rough front-to-back sorting (approaching the 'reverse painters algorithm'), early Z-reject (in conjunction with hierarchical Z), and less common deferred rendering techniques possible on programmable GPUs.

Scanline techniques working on the raster have the drawback that overload is not handled gracefully.

The technique is not considered to scale well as the number of primitives increases. This is because of the size of the intermediate datastructures required during rendering—which can exceed the size of a Z-buffer for a complex scene.

Consequently, in contemporary interactive graphics applications, the Z-buffer has become ubiquitous. The Z-buffer allows larger volumes of primitives to be traversed linearly, in parallel, in a manner friendly to modern hardware. Transformed coordinates, attribute gradients, etc., need never leave the graphics chip; only the visible pixels and depth values are stored.

See also [edit]

- Raster scan
- Ray tracing
- Z-buffering
- Bresenham's line algorithm
- Caulking (video games)

References [edit]

1. ^ Wylie, C, Romney, G W, Evans, D C, and Erdahl, A, "Halftone Perspective Drawings by Computer," Proc. AFIPS FJCC 1967, Vol. 31, 49
2. ^ Bouknight W.J, "An Improved Procedure for Generation of Half-tone Computer Graphics Representation," UI, Coordinated Science Laboratory, Sept 1969
3. ^ Newell, M E, Newell R. G, and Sancha, T.L, "A New Approach to the Shaded Picture Problem," Proc ACM National Conf. 1972

External links [edit]

- University of Utah Graphics Group History

Categories: Optics | 3D rendering | Computer graphics algorithms