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
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Français
Add links

Hidden surface determination

From Wikipedia, the free encyclopedia

In 3D computer graphics, **hidden surface determination** (also known as **hidden surface removal** (**HSR**), **occlusion culling** (**OC**) or **visible surface determination** (**VSD**)) is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint. A hidden surface determination algorithm is a solution to the visibility problem, which was one of the first major problems in the field of 3D computer graphics. The process of hidden surface determination is sometimes called **hiding**, and such an algorithm is sometimes called a **hider**. The analogue for line rendering is hidden line removal. Hidden surface determination is necessary to render an image correctly, so that one cannot look through walls in virtual reality.

Contents [hide]

Background [edit]

Hidden surface determination is a process by which surfaces which should not be visible to the user (for example, because they lie behind opaque objects such as walls) are prevented from being rendered. Despite advances in hardware capability there is still a need for advanced rendering algorithms. The responsibility of a rendering engine is to allow for large world spaces and as the world's size approaches infinity the engine should not slow down but remain at constant speed. Optimising this process relies on being able to ensure the deployment of as few resources as possible towards the rendering of surfaces that will not end up being rendered to the user.

There are many techniques for hidden surface determination. They are fundamentally an exercise in sorting, and usually vary in the order in which the sort is performed and how the problem is subdivided. Sorting large quantities of graphics primitives is usually done by divide and conquer.

Hidden surface removal algorithms [edit]

Considering the rendering pipeline, the projection, the clipping, and the rasterization steps are handled differently by the following algorithms:

- Z-buffering During rasterization the depth/Z value of each pixel (or *sample* in the case of anti-aliasing, but without loss of generality the term *pixel* is used) is checked against an existing depth value. If the current pixel is behind the pixel in the Z-buffer, the pixel is rejected, otherwise it is shaded and its depth value replaces the one in the Z-buffer. Z-buffering supports dynamic scenes easily, and is currently implemented efficiently in graphics hardware. This is the current standard. The cost of using Z-buffering is that it uses up to 4 bytes per pixel, and that the rasterization algorithm needs to check each rasterized sample against the z-buffer. The z-buffer can also suffer from artifacts due to precision errors (also known as z-fighting), although this is far less common now that commodity hardware supports 24-bit and higher precision buffers.
- Coverage buffers (C-Buffer) and Surface buffer (S-Buffer): faster than z-buffers and commonly used in games in the Quake I era. Instead of storing the Z value per pixel, they store list of already displayed segments per line of the screen. New polygons are then cut against already displayed segments that would

hide them. An S-Buffer can display unsorted polygons, while a C-Buffer requires polygons to be displayed from the nearest to the furthest. Because the C-buffer technique does not require a pixel to be drawn more than once, the process is slightly faster. This was commonly used with binary space partitioning (BSP) trees, which would provide sorting for the polygons.

- Sorted Active Edge List: Used in Quake 1, this was storing a list of the edges of already displayed polygons (see scanline rendering). Polygons are displayed from the nearest to the furthest. New polygons are clipped against already displayed polygons' edges, creating new polygons to display then storing the additional edges. It's much harder to implement than S/C/Z buffers, but it scales much better with increases in resolution.
- Painter's algorithm sorts polygons by their barycenter and draws them back to front. This produces few artifacts when applied to scenes with polygons of similar size forming smooth meshes and back-face culling turned on. The cost here is the sorting step and the fact that visual artifacts can occur.
- Binary space partitioning (BSP) divides a scene along planes corresponding to polygon boundaries. The subdivision is constructed in such a way as to provide an unambiguous depth ordering from any point in the scene when the BSP tree is traversed. The disadvantage here is that the BSP tree is created with an expensive pre-process. This means that it is less suitable for scenes consisting of dynamic geometry. The advantage is that the data is pre-sorted and error free, ready for the previously mentioned algorithms. Note that the BSP is not a solution to HSR, only an aid.
- Ray tracing attempts to model the path of light rays to a viewpoint by tracing rays from the viewpoint into the scene. Although not a hidden surface removal algorithm as such, it implicitly solves the hidden surface removal problem by finding the nearest surface along each view-ray. Effectively this is equivalent to sorting all the geometry on a per pixel basis.
- The Warnock algorithm divides the screen into smaller areas and sorts triangles within these. If there is ambiguity (i.e., polygons overlap in depth extent within these areas), then further subdivision occurs. At the limit, subdivision may occur down to the pixel level.

Culling and Visible Surface Determination [edit]

A related area to Visible Surface Determination (VSD) is *culling*, which usually happens before VSD in a rendering pipeline. Primitives or batches of primitives can be rejected in their entirety, which *usually* reduces the load on a well-designed system.

The advantage of culling early on the pipeline is that entire objects that are invisible do not have to be fetched, transformed, rasterized or shaded. Here are some types of culling algorithms:

Viewing frustum culling [edit]

The viewing frustum is a geometric representation of the volume visible to the virtual camera. Naturally, objects outside this volume will not be visible in the final image, so they are discarded. Often, objects lie on the boundary of the viewing frustum. These objects are cut into pieces along this boundary in a process called clipping, and the pieces that lie outside the frustum are discarded as there is no place to draw them.

Backface culling [edit]

Main article: Back-face culling

With 3D objects, only half of the surfaces are facing the camera, and the rest are facing away from the camera, i.e. are on the back side of the object, hindered by the front side. If the object is completely opaque, those surfaces need never to be drawn. They are determined by the vertex winding order: if the triangle drawn has its vertices in clockwise order on the projection plane when facing the camera, they switch into counter-clockwise order when the surface turns away from the camera.

Incidentally, this also makes the objects completely transparent when the viewpoint camera is located inside them, because then all the surfaces of the object are facing away from the camera and are culled by the renderer. To prevent this the object must be set as double-sided (i.e. no backface culling is done) or have separate inside surfaces.

Contribution culling [edit]

Often, objects are so far away that they do not contribute significantly to the final image. These objects are thrown away if their screen projection is too small. See Clipping plane.

Occlusion culling [edit]

See also: Z-buffering § Z-culling

Objects that are entirely behind other opaque objects may be culled. This is a very popular mechanism to speed up the rendering of large scenes that have a moderate to high depth complexity. There are several types of occlusion culling approaches:

- Potentially visible set or *PVS* rendering, divides a scene into regions and pre-computes visibility for them. These visibility sets are then indexed at run-time to obtain high quality visibility sets (accounting for complex occluder interactions) quickly.
- Portal rendering divides a scene into cells/sectors (rooms) and portals (doors), and computes which sectors are visible by clipping them against portals.

Hansong Zhang's dissertation "Effective Occlusion Culling for the Interactive Display of Arbitrary Models" [1] describes an occlusion culling approach.

Divide and conquer [edit]

A popular theme in the VSD literature is divide and conquer. The Warnock algorithm pioneered dividing the screen. Beam tracing is a ray-tracing approach which divides the visible volumes into beams. Various screen-space subdivision approaches reducing the number of primitives considered per region, e.g. tiling, or screen-space BSP clipping. Tiling may be used as a preprocess to other techniques. ZBuffer hardware may typically include a coarse 'hi-Z' against which primitives can be rejected early without rasterization, this is a form of occlusion culling.

Bounding volume hierarchies (BVHs) are often used to subdivide the scene's space (examples are the BSP tree, the octree and the kd-tree). This allows visibility determination to be performed hierarchically: effectively, if a node in the tree is considered to be *invisible* then all of its child nodes are also invisible, and no further processing is necessary (they can all be rejected by the renderer). If a node is considered *visible*, then each of its children need to be evaluated. This traversal is effectively a tree walk where invisibility/occlusion or reaching a leaf node determines whether to stop or whether to recurse respectively.

Sources [edit]

- http://www.cs.washington.edu/education/courses/cse557/07wi/lectures/hidden-surfaces.pdf
- http://design.osu.edu/carlson/history/PDFs/ten-hidden-surface.pdf

See also [edit]

- PowerVR
- Dreamcast
- List of Sega arcade system boards

v · t · e	Virtual reality · Mixed reality · Reality [hide]
Concepts	Virtuality · Virtual cinematography · Augmented reality · Augmented virtuality · Real life · Projection augmented model · Reality–virtuality continuum · Artificial reality · Simulated reality · Ubiquitous computing · Virtual world (Persistent world) · Multimodal interaction · Telepresence · Immersion
Technology	Compositing · Camera resectioning · Haptic suit · Head-mounted display (Optical head-mounted display) · Head-up display · Image-based modeling and rendering · Real-time computer graphics · Virtual retinal display · Wearable computer · Stereoscopy (Computer stereo vision, Computer vision) · Chroma key · Visual hull · Free viewpoint television · Omnidirectional treadmill · **Hidden surface determination**
Tracking	Motion capture · Tracking system · Types (Optical · Inertial · Magnetic) · Devices (Wired glove · Gametrak · HoloLens · PlayStation Move · Leap Motion · Kinect · Sixense TrueMotion)
Immersive devices	Personal (Google Cardboard · HTC Vive · Oculus Rift · Samsung Gear VR · Project Morpheus · Razer OSVR) · Rooms (AlloSphere · Cave · TreadPort) · History (Sensorama · Virtual Boy · Famicom 3D System · Sword of Damocles · Sega VR · Virtuality)
Applications	Pervasive game · ARToolKit · Interactive art (Virtual graffiti)
See also	Simulated reality in fiction

Categories: 3D rendering