Andrew Frantz
ajfrantz@umich.edu
11-20-2006

# Backface Culling

It is simple reality of a 3D world that you do not usually see an entire object from a single perspective, and when you do it is usually only due to reflection or some other special case. In computer graphics we harness this reality in order to accelerate the drawing of objects on the screen, using a method known as "backface culling."

In the field of computer graphics probably the most frequent task is to take a representation of an object (or many objects) in some form, say, a mesh of vertices, and render that object from an arbitrary viewpoint. The easiest way to do this is to simply start from the vertex furthest from the viewpoint and begin drawing the object(s) from back to front. This way we know we will produce an image of the scene accurately—every vertex will be drawn, and parts of the object closer to the camera will be drawn later, so that those parts are visible.

However what backface culling adds to the equation is taking advantage of the knowledge that when you look at a scene you will only see the parts of the objects that are closest to you or, more specifically, the parts facing you. The parts of the object that are facing away (for example, the back of a cardboard box) are not going to be visible—even when we draw those faces, as we do in the algorithm presented above, they are simply going to be covered by some other forward-facing part of the same object. We therefore know that we can gain a rendering speedup by not ever drawing those faces that are facing away from the viewpoint.

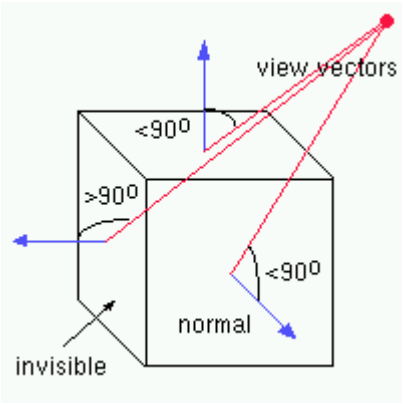

*Figure 1: It is easy to see that not all faces are always visible on a 3D object.*

The most common implementation of backface culling (indeed the one used in every implementation I found in my research) is based on the ordering of vertices during rendering. In OpenGL, vertices of faces are expected to be drawn in counterclockwise order. Then, while rendering, OpenGL generates a normal for the face being rendered. The process for generating this normal is dependent on the ordering of the vertices—OpenGL could have as easily determined it was going to prefer clockwise vertices over counterclockwise ones, but it uses the counterclockwise ordering. The dot product between the generated normal for that face and a vector from the center of projection to a point on the face is then taken, and a simple test to see if that dot product is less than 0 is made before OpenGL attempts to actually render the face. This allows backward facing polygons ("backfaces") to be skipped during rendering ("culled") quickly and easily.

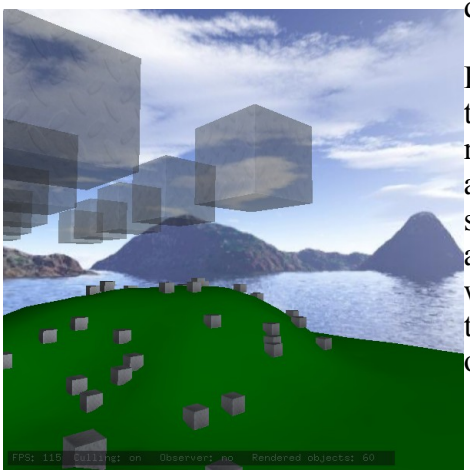
*Figure 2: Transparency in rendering with backface culling.*

Backface culling is not without its flaws, however. One flaw is that added constraint on vertex ordering inside of rendering routines. If care is not taken to be sure vertices are rendered in the appropriate order, backface culling could result in faces being skipped in the rendering process accidently. Some shapes, such as an open container, require that the back of faces be rendered as well as the front of faces. In those situations it is either required that backface culling be turned off during rendering of that object only, or a seperate set of vertices be created for the inside faces in

as well as the set for the outside faces. Additionally if transparency is involved rendering the backfaces may be required to produce the expected result. Finally, there are some situations where graphics hardware simply cannot perform backface culling as fast as it could draw the polygons in the first place —one example of this is the Playstation 2, where backface culling is often not used because it decreases performance.

There are also several other forms of culling used in computer graphics. Some, like binary space partitioning, are based on exploiting known properties of a scene to quickly cull objects, whereas others use more advanced algorithms hoping that the payoff in rendering time, for example, some kinds of occlusion culling which attempt to detect when another object is in front of the one you are rendering before you render it.

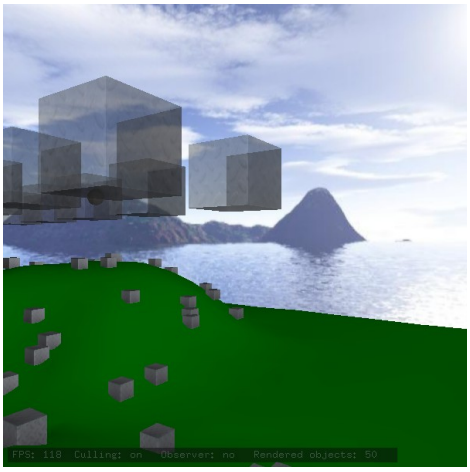


*Figure 3: Transparency in rendering without backface culling. Notice the difference from Figure 2--sometimes this may be a desired effect!*

Andrew Frantz
ajfrantz@umich.edu
11-20-2006

Sources

http://www.gamedev.net/reference/articles/article1088.asp

http://www.evl.uic.edu/aej/488/lecture10.html

http://www.kirupa.com/developer/actionscript/backface_culling.htm

http://en.wikipedia.org/wiki/Back-face_culling

http://www.cellperformance.com/mike_acton/2006/08/vblanks_mg5_engine.html

http://www.acc.umu.se/~erikw/