Ilya Seletsky

Real Time Occlusion Culling Related Work

In the past visibility data was precomputed. BSP trees dominated the 3D engine market. id Software's idTech engines, starting from the engine that powered DooM used statically precomputed BSP trees. They licensed their engines out to many other companies for them to make games with. The Unreal Engine also used BSP trees. This is a great data structure for complex static indoor environments. It takes time to precompute, so level designers don't get to view their creations on the fly as they will show up in game. Destructible environments are only possible if the artist specifies a certain part of the environment to be destructible, and that part won't even be part of the static BSP data. This was a very good technique for the technology at the time because it allowed very complex environments to be rendered at interactive framerates. When the BSP data is built it also creates a potential visibility set for each area of the level. This way the 3D camera only needs to attempt to render a small percentage of the environment.

The Crysis engine has a WYSIWYG level editor. The engine avoids as much precomputed static data as possible. This allows artists to use the editor and see exactly how the game will look. This also makes for easily destructible environments. The Battlefield 3 engine works similarly, but they don't release any details about their editor. Both use a software rasterizer to perform occlusion queries before drawing the final frame using the 3D accelerated hardware. They avoid the overhead of performing hardware occlusion queries that are often a frame late.

There have been many other papers on real time dynamic occlusion culling, which is also useful for CAD programs. Hanson Zhang's Effective Occlusion Culling for the Display of Arbitrary Models paper presents a technique using a hierarchical pyramid depth buffer for dynamic occlusion culling. This would require a software implementation as well since this isn't built into the pipeline of GPU hardware.

Dynamic Scene Visibility Culling using a Regular Grid presents a technique for storing the scene in a 3D uniform grid. It then roughly traverses the view frustum front to back through the grid cells. This paper doesn't take advantage of hardware occlusion queries and manually filters out scene cells that are occluded by other scene cells.

Another paper presents a way to use hardware occlusion queries on a scene using a hierarchical octtree for storing the scene rather than a uniform 3D grid scene. They then use previous occlusion queries from previous frames to better predict how to traverse the tree hierarchy of the scene in later frames.