Background and Related Work

Bill Hess

December 4, 2009

1 Background

In computer graphics, the View Frustum determines which parts of the scene are drawn on the screen. The View Frustum is an area that is defined by 6 planes. These planes represent the borders of what is visible on the screen defined by the camera. They constrain the top, bottom, left, right, front and back of the viewing area. The most important to us the back view plane. Unlike the top, bottom, left and right frustum planes, the back plane actually removes geometry from the scene that would otherwise be visible. As the camera moves through the scene, geometry in the background will suddenly appear and disappear as it moves across the back view plane.

The distance between the front and back view planes determines the depth of scene. The front view plane is usually fixed very close to the camera position. This means that the depth is determined by how far the back plane is placed away from the camera. Ideally we would like to place the back view plane infinitely far away from the camera so that no background geometry is removed from the screen. However, there are two limitations that prevent this.

The first is the limitation of the Z-buffer. The Z-buffer is an area of video memory that stores the depths at which polygons are drawn for every pixel. When polygons are drawn in the scene, their depth is calculated for every pixel they will occupy. If the depth recorded in the depth buffer is closer to the camera than the current polygon at that pixel, it will not be drawn. This allows graphics application writers to not worry about the order in which they decide to draw objects in the scene. This also allows geometry to intersect while preserving the proper depth ordering.

The Z-buffer can only hold a finite number of values. In any given scene, there is a range of possible distances from the camera that can map to the same Z-buffer value. This is not an issue in many graphics scenes because the depth of the scene is not very large and even very close objects are rendered properly. However, as the back plane is pushed away from the camera, the range of values mapping to the same Z-buffer value increases. The Z-buffer precision cannot change per application as it is implemented in hardware.

If two differently colored objects are drawn to the same depth in the Z-buffer, the hardware may choose which object to draw per pixel in an undefined way.
This results in flickering as alternating objects can be drawn in the same spot every frame of animation. It can also cause the image to look distorted and grainy since the graphics hardware can choose either polygon to draw on top on a per-pixel basis.

The second limitation imposed by the back plane relates to the total amount of geometry in the scene. In nearly all scenes a perspective camera view is used. The perspective camera view causes objects that are far away from the camera to appear smaller than objects closer to the camera. This is similar to how we view objects naturally. This also means that areas of the scene near the back plane map much larger amounts of geometry to the same number of pixels on the screen.

The volume of the scene that the view frustum contains is determined primarily by the field of view and the back view plane. As the distance between the back plane and the camera increases, the volume contained by the frustum increases with the depth’s cube. As the volume increases, the potential number of objects required to render the scene increases as well. If we double the depth of any given scene, we potentially increase the amount of geometry to render by 8.

A common technique to combat this problem is to use varying level of detail models according to their distance from the camera. A high resolution model composed of nearly a million polygons can be rendered normally when it appears close to the camera, but a lower resolution version of the same model that uses only 1000 polygons might be drawn instead when the model is far away from the camera. Smaller models that add detail to the scene can disappear entirely.

Unfortunately, this has to be taken to an extreme when the depth of a scene is very large. Even large models must be noticeably faded out near the back of the view frustum. This causes large backgrounds to appear featureless in many scenes.
Another technique for simplifying background geometry described in [1] is to use something called a Skybox. A skybox is a static, flat representation of the background. It is projected onto a cube surrounding the scene. The Skybox’s position does not move relative to the camera, giving it the appearance of being very far away.

A similar technique exists for general models in the scene. A complex model is rendered onto a texture image, and then that image is displayed in place of the original model. This technique is called an Impostor. As long as the viewing angle and distance to an Impostor do not vary greatly, the flat texture image is indistinguishable from the high resolution model.

2 Related Work

While adjusting level of detail and using Skyboxes are the more traditional techniques for rendering backgrounds, there have been a few notable methods for pushing the limits of scene depth and background geometry. One of these is the 3D Skybox technique used by Valve Software in their Source Engine[5][6] which has been used in the popular game Half-Life 2.

The Source engine allows level creators to greatly expand the depth of background scenes by creating 3D Skyboxes. 3D Skyboxes are 1:16 scale versions of the scene that are placed out of view of the player. The 3D skybox contains only background geometry that is not reachable by the player. All parts of the level that the player can reach are drawn to normal scale. This typically means that the 3D Skybox contains a convex empty space where the full-scale geometry would be. Geometry in the 3D Skybox and full scale models do not overlap with each other.

To draw the scene, the camera is first placed in its corresponding position within the scaled down 3D skybox. The depth of the back plane of the view frustum is adjusted so that the maximum precision can be used for rendering the relatively small skybox. Once the skybox geometry has been rendered to the screen, the camera is moved back to its original position, the depth buffer is cleared, and the full scale foreground geometry is rendered as normal. The end result is that the player perceives the geometry in the skybox to be 16 times larger than it actually is.

This process allows us to use the depth buffer twice. Once to render all of the background geometry and again to render the foreground geometry. This is made possible by a constraint placed on the player moving within the scene. The player is only allowed to move within the full scale, foreground geometry. For this reason, it is not possible for any geometry which appears in the background to occlude geometry in the foreground. Depth information for the 3D Skybox can be discarded and its entire precision can be reused when drawing the foreground.

While this technique solves the same problem, we cannot apply the same constraint to our solution. My technique needs to work in scenes where geometry can appear close to the camera or as part of the background. However, much
like the technique used in the Source Engine, by placing constraints on camera movement we can introduce new optimizations.

References


