4.9

Never Let 'Em See You Pop—Issues in Geometric Level of Detail Selection

Yossarian King

Objects and characters are represented in computer graphics as geometric models. Models can be created at different levels of detail (LODs), with more polygons and larger textures for the more detailed models, and fewer polygons and smaller textures for the less detailed models. Why would you want to do this? To improve rendering performance and visual quality. Drawing fewer polygons when objects are far away from the camera reduces the polygon count of the scene, and so speeds up rendering. Having a more detailed model for use when an object is close to the camera improves visual quality. If only a single model is used, then there is always a tradeoff between performance and quality—multiple levels of detail help to achieve both.

To implement LOD rendering, multiple models are created at different levels of detail, and the model to be rendered is chosen, each frame based on distance from the camera. As a rough rule of thumb, each level of detail should have about twice the number of polygons as the preceding level. The models are created to reduce "popping" as much as possible—when the character or object switches from one level of detail to another, the visible change in geometry (especially at the silhouette edge) and texturing must be minimized. The artist's job is to create models that are as similar as possible when rendered at the scale where the transition will occur. The programmer's job is to determine when to change LODs to achieve the desired performance and quality while minimizing the number of LOD transitions. This article explains how.

Note that for objects or characters that stay a relatively constant distance from the camera (such as the hero character in a third-person game), level of detail selection is not necessary. Also note that this article does not address level of detail issues for terrain rendering.

LOD Selection

The simplest way to select which level of detail to render is to apply a threshold to the distance of the object from the camera. For example, use the high detail model when the object is closer than 500 units, the medium detail model for distances of 500–1500 units, and the low detail model when the object is further than 1500 units from the camera. At first glance this seems reasonable—when the object is closer, use more detail; when it is farther away, use less detail—however, there are two problems with this method.

First, it doesn't account for the field of view of the camera. If the object is a long way from the camera, but the field of view is very narrow (e.g., a zoom lens) then the object may appear large on screen, and a detailed model may be appropriate. Similarly, an object may be relatively close to the camera, but if the field of view is very wide (a macro lens), then the object may appear small on screen and a low detail model should be used. Figure 4.9.1 shows that the same object at the same distance from the camera does not always appear the same size on screen. Rather than distance from the camera, we really want to use the projected size of the object on the screen as a basis for choosing the detail level. Size on screen is obviously related to distance from the camera, but the field of view must also be accounted for.

The second problem with the simple distance threshold approach is that if the object remains close to the threshold distance, then there may be rapid toggling back

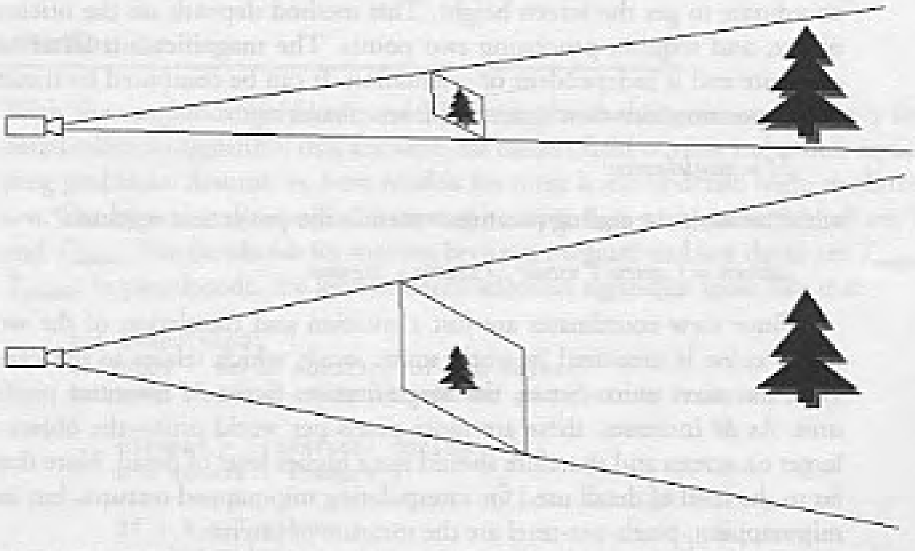

FIGURE 4.9.1 Varying the field of view changes the projected size of objects on the screen. *Top:* A narrow field of view produces a larger image on screen. *Bottom:* A wider field of view produces a smaller image. In both cases, the size of the tree and the distance from the tree to the camera are the same, demonstrating that camera distance is not sufficient for choosing level of detail.

and forth between levels of detail. This can happen when a character is running across the field of view close to the threshold distance. Popping once from one level of detail to another might be noticeable, but rapid cycling back and forth between levels will be very distracting and undesirable.

Fortunately, both of these problems are easily solved. A better alternative to camera distance is a "magnification factor," which is the screen size of the object relative to its physical size. As this ratio increases (i.e., as the object grows larger on screen), we choose higher levels of detail. Screen size accounts for both camera distance and field of view, so the first problem is eliminated. The magnification factor is easy to calculate, as explained in the next section.

The problem of rapid popping back and forth is solved by using hysteresis thresholding. Normal thresholding selects an output based on applying a single threshold to an input value. Hysteresis thresholding uses an upper and a lower threshold, and decides which to apply based on the previous output value. As long as the input value remains between the upper and lower threshold, the output value doesn't change, thus stabilizing the thresholded output and, in our case, maintaining stability of LOD selection. Details are described later.

Magnification Factor

The screen size of an object could be determined by transforming and projecting the highest and lowest point on the object and subtracting the screen position of each coordinate to get the screen height. This method depends on the orientation of the object, and requires processing two points. The magnification factor is simpler to compute and is independent of orientation. It can be computed by transforming the object position into view space and then calculating:

$$M = xscale / zview$$

where *xscale* is the scaling parameter used in the projection equation:

$$xscreen = (xview * xscale) / zview + xcenter$$

Since view coordinates are just a rotation and translation of the world coordinates, *zview* is measured in world units. *xscale*, which relates to the camera field of view, has pixel units; hence, the magnification factor *M* measures pixels per world unit. As *M* increases, there are more pixels per world unit—the object is relatively larger on screen and therefore should use a higher level of detail. Note that *M* is similar to the level of detail used for interpolating mipmapped textures, but in the case of mipmapping, pixels-per-texel are the measure of interest.

M accounts for both camera distance (via *zview*) and field of view (via *xscale*), and makes a much better choice for determining level of detail selection than simple camera distance. However, applying a simple threshold to *M* will have the same popping problems described previously. Hysteresis thresholding is the solution.

Hysteresis Thresholding

Hysteresis thresholding is a fancy term for thresholding against a range of values, rather than a single value. A simple threshold T is applied as:

$$output = \begin{cases} 1 \text{ if } input >= T \\ 0 \text{ if } input < T \end{cases}$$

Hysteresis thresholding uses an upper and a lower threshold and remembers the previous output value. The output doesn't change if the input is between the upper and lower thresholds:

$$output(t) = \begin{cases} 1 \text{ if } input >= T_{high} \\ 0 \text{ if } input < T_{low} \\ output(t-1) \text{ otherwise} \end{cases}$$

If the input is increasing, then the output will be 0 until the input reaches T_{high}. If the input is decreasing, then the output will remain 1 until the input falls below T_{low}. Regardless of whether the value is increasing or decreasing, the output doesn't change when the input is between the upper and lower thresholds. Using this approach for level of detail selection means there is no single point at which the object will toggle back and forth between levels of detail—the hysteresis thresholding ensures that all we get is a single pop, never a toggling behavior. A visual comparison of simple thresholding and hysteresis thresholding is shown in Figure 4.9.2.

Implementation

With the magnification factor and hysteresis thresholding we can create a level of detail selection algorithm that accounts for camera field of view and avoids rapid popping problems. Assume we have models for three levels of detail: high, medium, and low. The hysteresis thresholds for moving between high and medium detail are T_{hupper} and T_{hlower}. The thresholds for moving between medium and low detail are T_{mupper} and T_{mlower}. In pseudocode, the level of detail selection algorithm looks like this:

```
int computelod:
worldpos    world position of the object
lodprev     level of detail chosen in previous frame
{
    viewpos = transform( worldpos )
    M = xscale / viewpos.z

    if ( M < T_mlower )
        lod = low
    else if ( M < T_mupper )
        lod = lodprev           hysteresis range for medium/low
    else if ( M < T_hlower )
        lod = medium
    else if ( M < T_hupper )
```

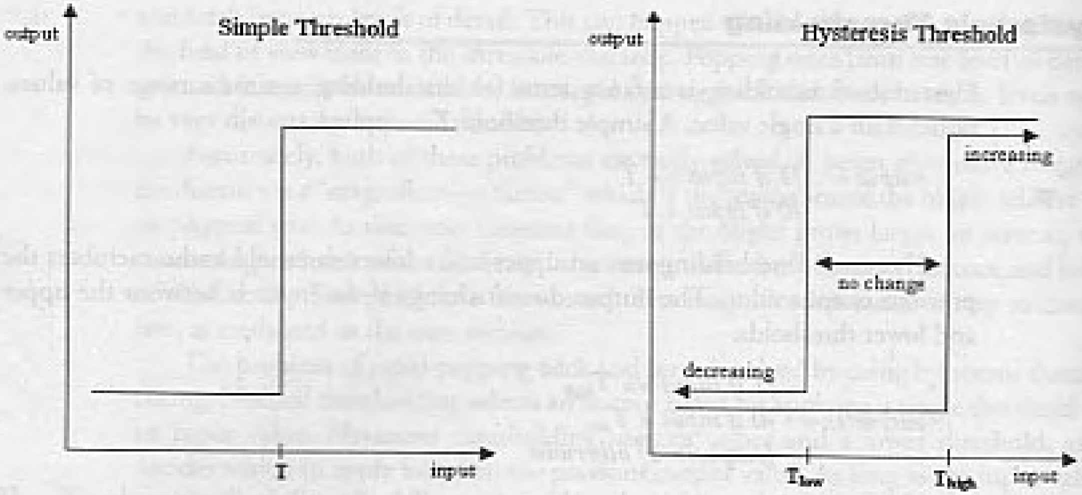

FIGURE 4.9.2 *Left:* An input value is thresholded against a single threshold *T*. *Right:* An input value is thresholded against hysteresis thresholds T_{low} and T_{high}, with the appropriate threshold chosen based on the preceding output value. As long as the input remains between T_{low} and T_{high}, there is no change in the output.

```
        lod = lodprev          hysteresis range for high/medium
    else
        lod = high             M >= T_hupper

    return lod
}
```

Note that if *M* is between T_{hlower} and T_{hupper}, then this will always return the previous level of detail, even if it was low detail. If you expect your object to be magnifying this quickly, then the algorithm is easily adapted.

The equivalent algorithm using a simple distance threshold would just use one threshold between each level of detail, and would look like this:

```
int computelodwithpopping:
worldpos
{
    viewpos = transform( worldpos )

    if ( viewpos.z < T_m )
        lod = low
    else if ( viewpos.z < T_h )
        lod = medium
    else                         viewpos.z >= T_h
        lod = high

    return lod
}
```

As can be seen, solving the field of view dependence and popping problems does not add significantly to the complexity of the level of detail selection.

Other Issues

Threshold Selection

For any thresholding method, hysteresis or otherwise, you need to choose your thresholds. For level of detail selection, choosing the thresholds is a tradeoff between performance and visual quality. If the thresholds are set too low, then the higher levels of detail will be drawn more often, and you will have to render more polygons per frame, which will slow down the rendering. If the thresholds are too high, then the lower quality models will be drawn more often, and popping between levels will be more noticeable.

To reduce popping, you can implement the selection algorithm, then move the object toward and away from the camera, moving the thresholds until the popping is acceptable. Keep in mind that in a game-play situation with a moving object, moving camera, and distracted user, the popping will be less noticeable than in a testbed environment. Achieving performance targets is a combined effort by the artist and programmer to reduce the polygon counts and adjust the thresholds to achieve a suitable balance.

User Attention

So far, we have only considered the size of an object on screen to determine which level of detail to render. Another factor to consider is where the user is looking. In general, we expect the user to be paying attention to things that are close to the camera, but we may have additional knowledge about the particular game situation to help us know where the user is probably looking. For example, in a sports game, a player character controlled by the user will probably be the focus of attention, as will a player character with the ball, or one involved in the current play. This "expected focus of attention" can be used in the level of detail selection algorithm by biasing the magnification factor when the user is likely to be watching an object or character. In the sports game example, we can multiply the magnification factor by a scaling parameter (e.g., 1.1) when the character is under user control.

Biasing the Magnification Factor

The idea of biasing the magnification factor can be used in other ways as well. If there is a game situation in which rendering quality is more important than performance (such as a non-interactive rendered cut scene), the magnification factor for all objects can be biased higher in order to render objects at higher detail. Or the magnification factor can be biased dynamically depending on frame rate—when frame rate drops, bias the magnification lower to select more lower polygon models and improve the frame rate.

Limiting Number of Models or Polygons

If a scene has multiple instances of the same object, then there may be limits on the number of objects that can be rendered at each level of detail. Such constraints may be imposed in order to conserve memory in the object representation, or you may simply wish to limit the maximum number of high polygon models in order to improve performance. Limits on the number of models at each level of detail can easily be built into the selection algorithm—sort the objects by magnification factor, and then take the *N* largest objects at each level of detail, demoting any remaining object to the next lower level of detail. With a little more work, the selection algorithm can be modified to select objects so the total polygon count for all objects falls below some target.

Progressive Meshes

A final issue worth mentioning is the use of progressive meshes, or other dynamic level of detail methods. Increasing processor performance and increasing polygon counts in game models are starting to make these techniques feasible. With these techniques, the polygon count of objects can be varied on the fly across a continuous range. A suitable polygon count still needs to be chosen for each frame, and so the magnification factor is still useful. If polygon count varies continuously, then hysteresis thresholding is no longer needed. However, this may cause distracting popping effects as polygons are continuously dropped from or added to the model, so it may still be desirable to use hysteresis thresholds to decide when to change the polygon count.