Guard Band Clipping in Direct3D

Sim Dietrich
NVIDIA Corporation
Sdietrich@nvidia.com

Guard band clipping is a hardware feature implemented in the RIVA 128, RIVA 128ZX, RIVA TNT, RIVA TNT2 and GeForce 256 graphics processors. It greatly reduces the cost of clipping polygons to the viewport. This paper will define guard band clipping, and discuss how to take advantage of it in 3D applications.

What Is Guard Band Clipping?

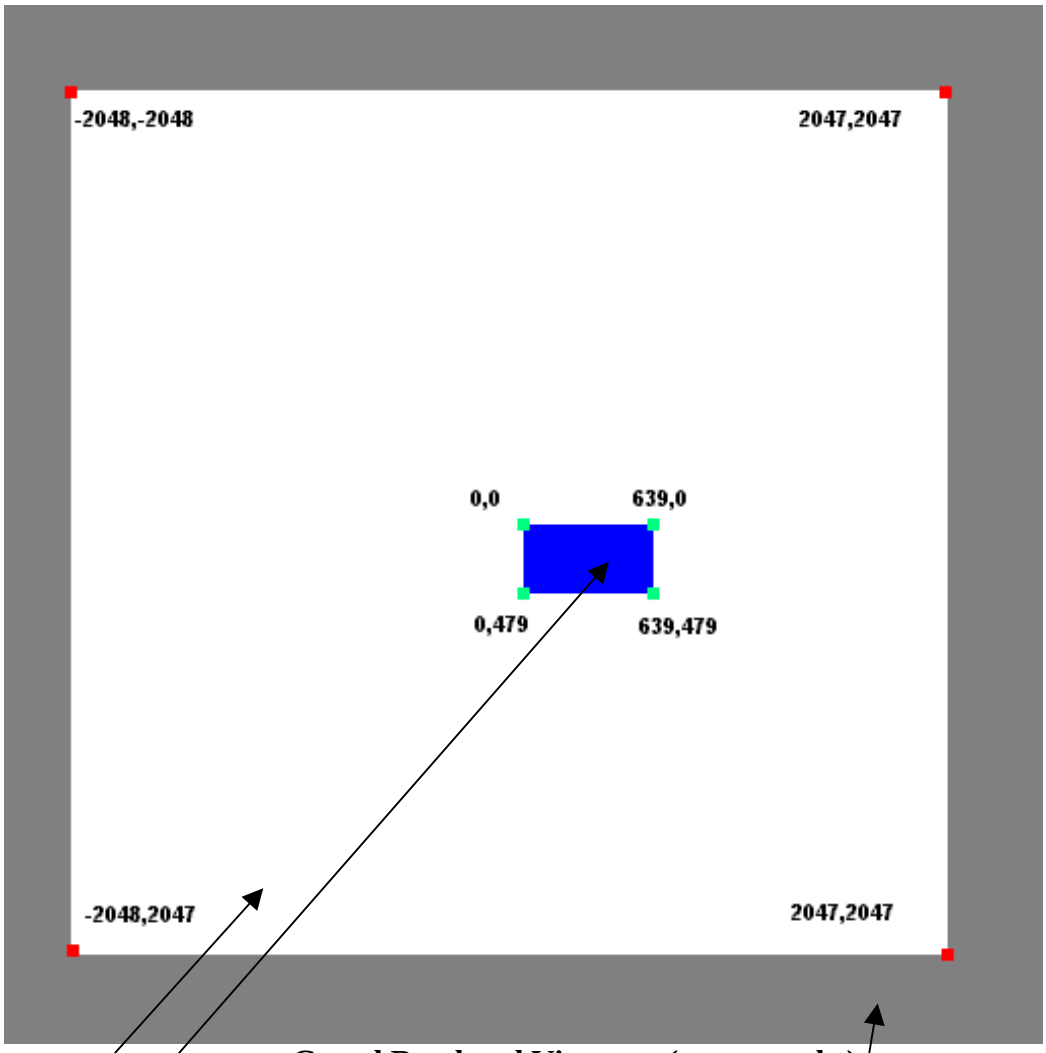
Hardware is said to support guard band clipping when it can accept screen coordinates outside of the current viewport range. For instance, the RIVA line of graphics processors support 2D screen coordinates from in the range -2048,-2048 to 2047,2047 inclusive, regardless of the screen resolution or viewport setting. This allows for more trivial accepting and rejecting of polygons, and less parametric primitive clipping, thus providing a performance benefit. The GeForce 256 supports a guard band of -100,000,000 to +100,000,000 both horizontally and vertically.

The basic idea of guard band clipping is that hardware with a guard band region can accept triangles that are partially or totally off-screen, thus avoiding expensive clipping work.

Typically, applications render into a viewport of 640x480, 800x600, 1024x768, 1280x1024, or 1600x1200. Primitives that lie partially or totally off-screen are typically clipped by the CPU to the screen or viewport boundary, which is represented by a 3D view frustum. This is a slow process, because each edge of each triangle that crosses the viewport boundary must have an intersection point calculated, and each parameter of the vertex (x,y,z diffuse r,g,b, specular r,g,b, alpha, fog, u and v) must be interpolated accordingly. Eliminating or reducing these calculations can lead to increased performance.

Guard band clipping reduces how often the CPU must perform these clipping calculations by accepting triangles that are partly off-screen. It allows the CPU to perform simpler, faster culling and clip tests in the majority of cases. Only rarely do triangles have to pass through the more expensive view frustum clipping code.

On the following page is a diagram of the guard band and the viewport.

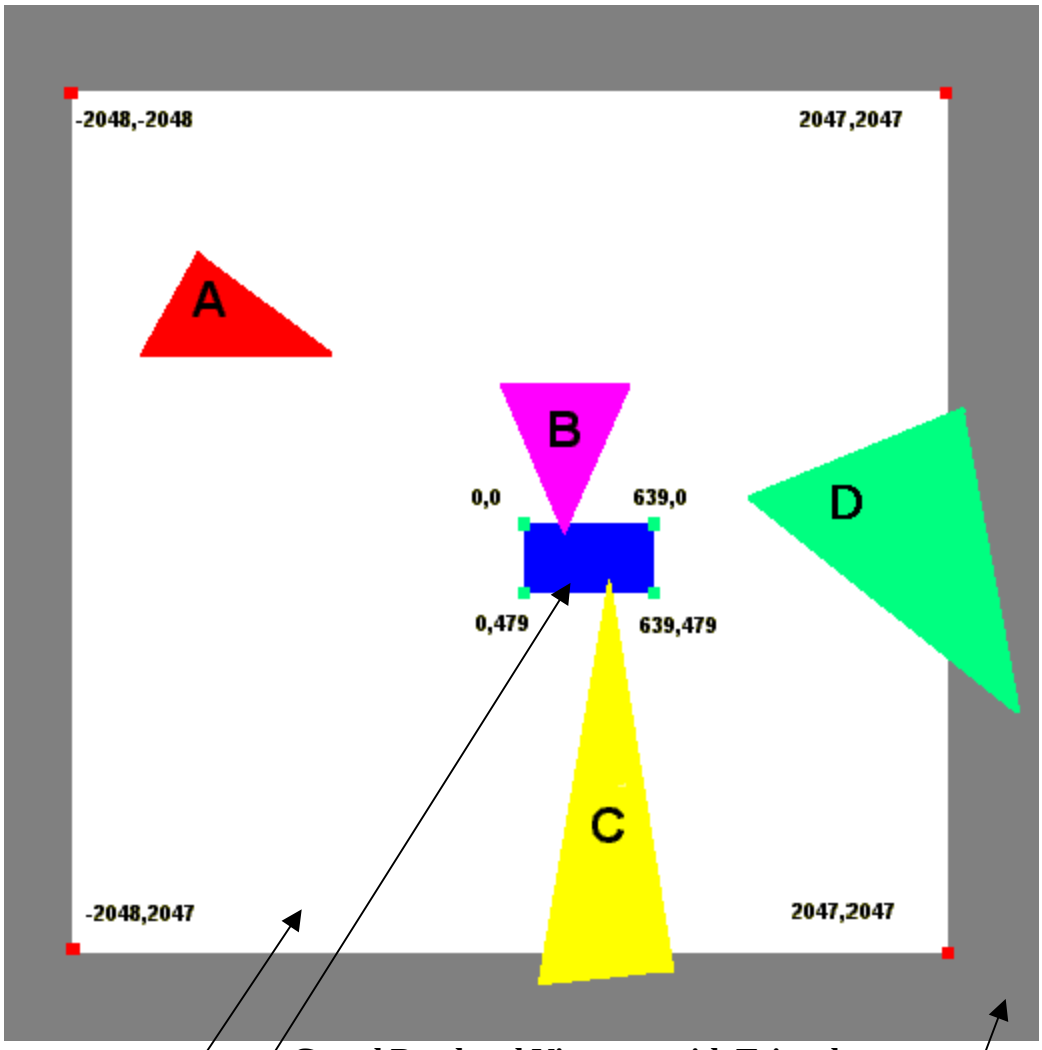


Guard Band and Viewport (not to scale)

Guard Band

Viewport

The hardware cannot handle coordinates outside the guard band.



Guard Band and Viewport with Triangles

Guard Band

Viewport

The hardware cannot handle coordinates outside the guard band.

In the above diagram, the viewport is indicated in blue. The guard band region is the white area whose borders are outlined in gray. Note that the guard band region is significantly larger than the viewport itself.

In the pictured case, the red triangle 'A' is completely within the guard band, and could be either trivially rejected or passed to the hardware as is, where it would be scissored away, thereby not requiring software clipping. However, if it were sent to the hardware, it would consume bus and 3D graphics processor bandwidth before being discarded on a pixel by pixel or line by line basis, which is slower than not sending the triangle at all.

The purple triangle 'B' straddles the Viewport boundary, but lies within the guard band completely. This also could be passed directly to the hardware, and the parts of 'B' that lie outside of the Viewport would be scissored away. This would be faster than software clipping it for all but very large triangles.

The yellow triangle 'C' straddles both the viewport boundary and the guard band boundary. This must be software clipped, either to the guard band edge, or to the Viewport boundary. It would seem most advantageous to simply clip it to the Viewport. An alternate strategy would be to subdivide the triangle and perform additional clip testing on each part.

The Green Triangle ‘D’ Crosses the guard band, but does not intersect the Viewport. This could either be clipped to the guard band or trivially rejected.

Clipping and Culling Strategies with a Guard Band

Applications typically employ two levels of clipping for many objects in the scene. First a rough cull is performed on each object, followed by actual clipping of the object's polygons only if necessary. Guard bands can be employed in both stages to increase performance by reducing the number of clipping calculations performed.

Applications typically perform a rough bounding box or bounding sphere 3D culling test on objects in the scene before determining if they can be thrown away completely or whether the object's polygons need to be clipped.

If objects cross a view frustum boundary, they are typically passed through a 3D frustum clipper, which can be a significant performance penalty. Some of the costs associated with 3D clipping are :

- Extra vertices produced, costing more bandwidth
- CPU cost for interpolation of x,y,z, u,v, color, specular, alpha and fog
- Breaking up of strips and fans
- Poor vertex locality of new vertices, which hurts CPU and vertex cache coherency

Guard band clipping can reduce 3D clipping costs by trivially accepting more objects as well as more polygons.

Adding Guard Band Support to the 3D Culling Step

Here is a sample 3D culling test.

```
Frustum theViewFrustum;
int i, j, k;

for( i = 0; i < theObjectList.size(); ++i )
{
    // This is a 3D culling check against the 3D viewing frustum
    if ( theObjectList[ i ].BoundingSphere().IsCompletelyWithin( theViewFrustum ) )
    {
        theObjectList[ i ].TransformAndProject( theViewFrustum );
    }
    else
    if ( theObjectList[ i ].BoundingSphere().IsCompletelyOutside( theViewFrustum ) )
    {
        continue; // skip this object altogether
    }
    else // Then the object is partly in the View frustum
    {
       theObjectList[ i ].ClipToFrustumAndDraw( theViewFrustum );
    }
} // for i
```

Note that objects partly within the viewing frustum are passed through the relatively expensive view frustum clipper.

An alternative is to create an additional, much bigger frustum that passes through the guard band extents. This allows us to identify the cases where the object is partly within the view frustum, but fully within the guard band frustum. These cases allow us to avoid clipping to the view frustum completely and trivially accept all of the object's polygons. This optimization will not reduce clipping to the near or far clip planes, but it will reduce clipping to the left, right, top and bottom viewport clip planes.

Here is an updated version of the above code :

```
Frustum theViewFrustum, theGuardBandFrustum; // Note the new frustum
int i, j, k;

for( i = 0; i < theObjectList.size(); ++i )
{
    // This is a 3D culling check against the 3D viewing frustum
    if ( theObjectList[ i ].BoundingSphere().IsCompletelyWithin( theViewFrustum ) )
    {
        theObjectList[ i ].TransformAndProject( theViewFrustum );
    }
    else
    if ( theObjectList[ i ].BoundingSphere().IsCompletelyOutside( theViewFrustum ) )
    {
        continue; // skip this object altogether
    }
    else // Then the object is partly in the View frustum
    {
       // Now test the Guard Band Frustum to try to avoid clipping
       if ( theObjectList[ i ].BoundingSphere().IsCompletelyWithin( theGuardBandFrustum))
       {
           theObjectList[ i ].TransformAndProject( theViewFrustum );
       }
       else
       {
           theObjectList[ i ].ClipToFrustumAndDraw( theViewFrustum );
       }
    }
} // for i
```

How to take Advantage of Guard Band Clipping

Applications that use Direct3D clipping will automatically see the benefits of guard band clip testing, although they must specifically add code to perform guard band-aware view frustum culling. The Direct3D pipeline automatically recognizes the guard band capability and will use it if present, and does not require the application to enable it.

Applications that perform their own clipping can still take advantage of guard band clipping, but they first need to detect its presence.

How to detect a Guard Band

Under Direct3D, there are four cap fields for Guard Band support in the D3DDEVICEDESC structure : dvGuardBandLeft, dvGuardBandRight, dvGuardBandTop, and dvGuardBandBottom. These represent the boundaries of the guard band.

```
LPDIRECT3DDEVICE3 device;

D3DDEVICEDESC aD3DHWDevDesc, aD3DSWDevDesc;

HRESULT hr = device->GetCaps( &aD3DHWDevDesc, &aD3DSWDevDesc );

if ( hr == D3D_OK )
{       // These fields now contain the Guard Band Extents
        aD3DHWDevDesc.dvGuardBandLeft;
        aD3DHWDevDesc.dvGuardBandRight;
        aD3DHWDevDesc.dvGuardBandTop;
        aD3DHWDevDesc.dvGuardBandBottom;
}
```

These extents will be 0 in the case that guard band clip testing is not supported.

Other Considerations

In the case where the application trivially accepts a group of polygons to be rendered (ie they all fall completely within the viewport), the application should make sure Direct3D does not try to clip them as well. In the DrawPrimitive() or DrawIndexedPrimitive() call, specify the flags to include

D3DDP_DONOTCLIP | D3DDP_DONOTUPDATEEXTENTS.

If an application provides its own transformations by using the D3DTLVERTEX structure, but relies on Direct3D to perform clipping, the Direct3D layer will un-project clipped points, perform clipping in 3D, and then re-project them back to 2D. Direct3D handles this problem by dividing **sx** and **sy** coordinates by **rhw** and then by the projection matrix's **x** and **y** scale factors, thus converting a D3DTLVERTEX to a D3DLVERTEX. It then performs clipping on the 3D coordinates, and puts them back through the projection matrix. This is obviously far from optimal, so if applications do their own transforms, they should perform their own clipping as well.

Summary

Guard band clipping provides a hardware-assisted, high-speed alternative to traditional view frustum clipping for the vast majority of triangles near the edge of the screen. Applications that utilize the Direct3D transformation pipeline automatically take advantage of guard band clipping on supporting hardware. Other applications can detect the guard band capability and improve speed by performing more culling and clip testing and less frustum clipping.