LOD (Level of Detail) in OpenSceneGraph

Level of detail is a tool used to cut corners on visual fidelity in situations where the observer is unlikely to notice or care about the apparent degradation. There are many forms of LOD, including continuous LOD (CLOD), which is often used for terrain models, and discrete LOD which is used for objects that cannot be represented by some sort of continuously variable approximation by mathematical function. Continuous LOD is not discussed here.

To employ discrete LOD, multiple versions of a model must be created, each with different degrees of complexity and visual quality. If the model is being built from scratch, the modeler needs to bear in mind the constraints of multiple LODs, and can often create the various LODs during the modeling process without too much additional work.

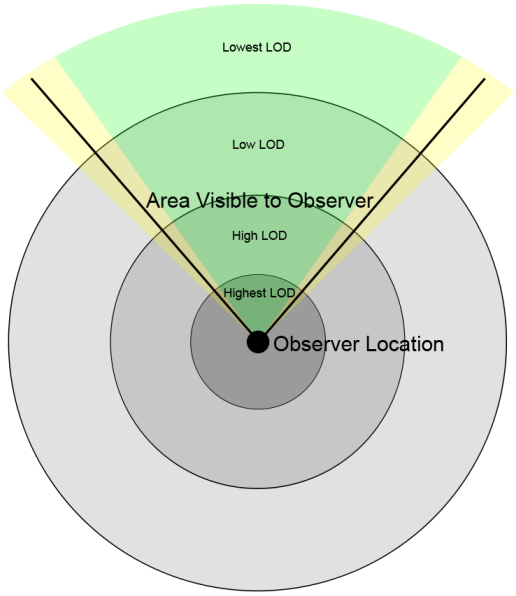
If a model is being used that has already been built, such as an engineering CAD file that has been repurposed for visualization use, additional work may be necessary to simplify the data into lower LODs. When converting a CAD model with primitives like spheres, cylinders and curved surfaces to a polygonal model such as used by a scenegraph, the conversion process may offer options to control how many polygons should be used to approximate these curves. As an example, a sphere could be represented by several different approximations:

Image					
Vertices	~5500	~2880	~1580	~670	140
Notes	Maximum detail, for closeups.				Minimum detail, very far objects.

The highest detail version produces something indistinguishable from a sphere, even when up close. The lowest-detail version also is indistinguishable from a sphere, when sufficiently far away (for example when the screen area covered by the sphere model is less than 20 pixels across).

Now apply this situation to an engineering model featuring cylindrical tanks with hemispherical ends, pipes with curved corner fittings and junctions, bolts with cylindrical shafts and possibly even spiral-cut threads. These models can quickly become excessively heavy, but careful choice of parameters during model conversion can keep the situation under control.

Each LOD is assigned a selection criteria that allows the scenegraph to determine when that model should be chosen and employed in the scenegraph. Typical LOD systems rely on a simple distance metric. Each LOD model will be assigned a (non-overlapping) range of distances. When the observer is within that defined range, the corresponding model is presented.



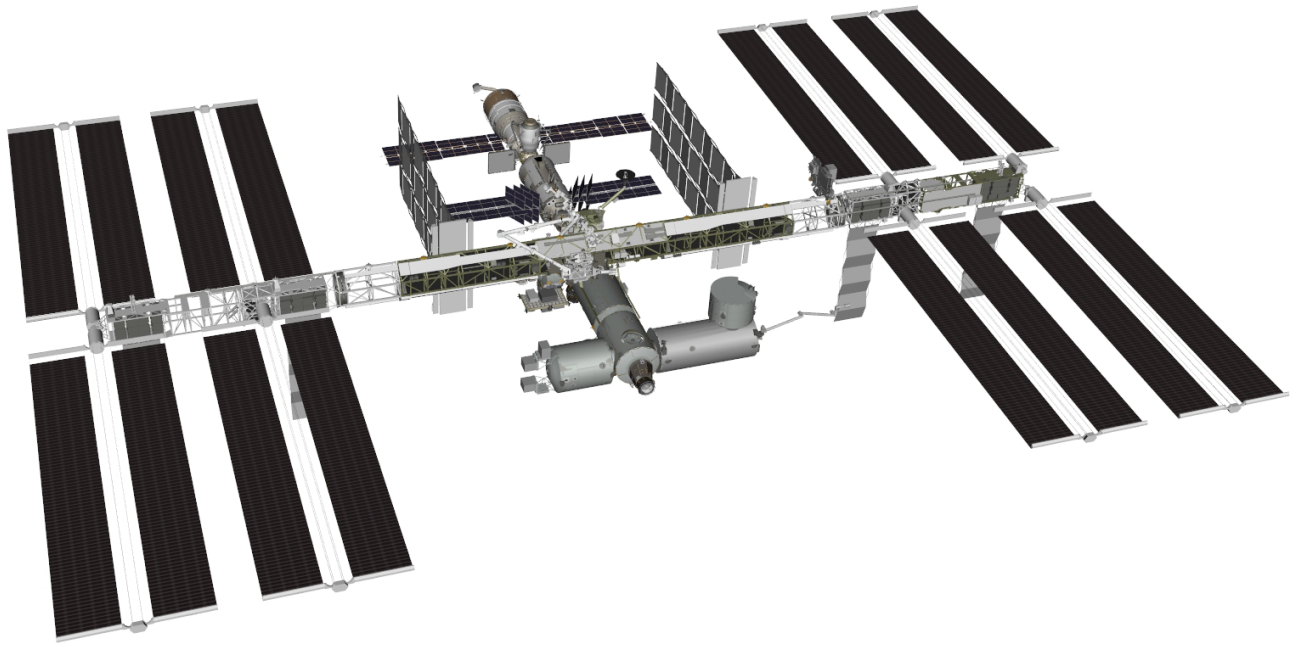
Some LOD systems may employ a transition strategy involving fading out one model while fading in the next to avoid a distracting visual 'pop' when the model switches from one LOD to another. OpenSceneGraph does not currently offer a Fade LOD functionality. This is because LOD transitions often happen at moments when maintaining draw performance is critical. The scenegraph might be trying very hard to switch to a lower LOD right away to keep redraw performance up (often due to a higher LOD simultaneously appearing elsewhere in the scene). A fade technique requires both the lower and the higher LODs to be drawn, as well as a blending operation to combine them. This more than doubles the time spent drawing the piece of the model that is changing LOD. Since this transition could possibly be happening in parallel in several places at once, the unbounded performance hit is impossible to predict or manage. Therefore, OpenSceneGraph does not employ Fade LOD.

The simple distance metric for LOD overlooks one critical aspect. The distance to the model is important, but equally important in the decision of how much detail will be perceivable is the fidelity and resolution of the observer's imaging surface. In a scenegraph, this is the resolution of the window or other output device the 3D graphics are being rasterized to. In the case of a 640x480 display surface, a low-detail model suffices until the observer is quite close. But if the observer is using an HD display (1920x1080) or even a large multi-screen/projector environment of thousands of pixels in both axes, the same metric will not choose an LOD adequate to satisfy the observer's requirements for detail.

In OpenSceneGraph, the traditional distance-metric is known as DISTANCE_FROM_EYE_POINT. OSG also offers a second option, known as PIXEL_SIZE_ON_SCREEN. In PIXEL_SIZE mode, the LOD Range numbers are no longer expressed as distances in scene units, but rather how big on screen (in pixels) a particular LOD is tuned for. In the case of DISTANCE mode, the highest-quality models are assigned LOD range values starting at 0 (units of distance from the camera) and increasing in magnitude for each lower-detail model. Conversely, with PIXEL_SIZE mode, the highest-quality model is assigned a high LOD Range value, corresponding with a large screen-space extent in pixels covered by the model. Lower-quality models are invoked as the visible extent of the model decreases with distance. In this way, you can specify that any time a model is displayed larger than a known number of pixels, whether it be because it is large and close up on a low-resolution display, or not as large/close but on a high-res display, the proper representation is chosen. A similar situation can occur with camera/lens/field-of-view style zooming. A level of detail defined as appropriate for 1Km distance is no longer appropriate if the viewer is using a pair of high-power binoculars from 1Km away. Therefore the DISTANCE_FROM_EYE_POINT mode is generally inadequate for expressing level of detail criteria and should be avoided in favor of PIXEL_SIZE_ON_SCREEN.

The current screen-space extent that OSG uses in this mode is determined by constructing a sphere that contains all of the model in question, then 3D projecting the diameter of the sphere into screen space and measuring the diameter in pixels.

For a practical example of LOD, this NASA model of the International Space Station (as of 2009) contains over 600,000 Polygons, some with many more than three points. It consists of detailed representation of every signifcant module, though it does not go so far as to include individual fasteners, wires or interior assemblies the way an engineering CAD file might.



This model performs well enough (60fps+) on a modern, powerful computer, but for the sake of example, let's assume it needs to be optimized for LOD. This could be because it is a much more complex model than it really is, or perhaps there are other operations being done at runtime that leave less CPU cycles available to draw the model, necessitating slimming it down to draw faster.

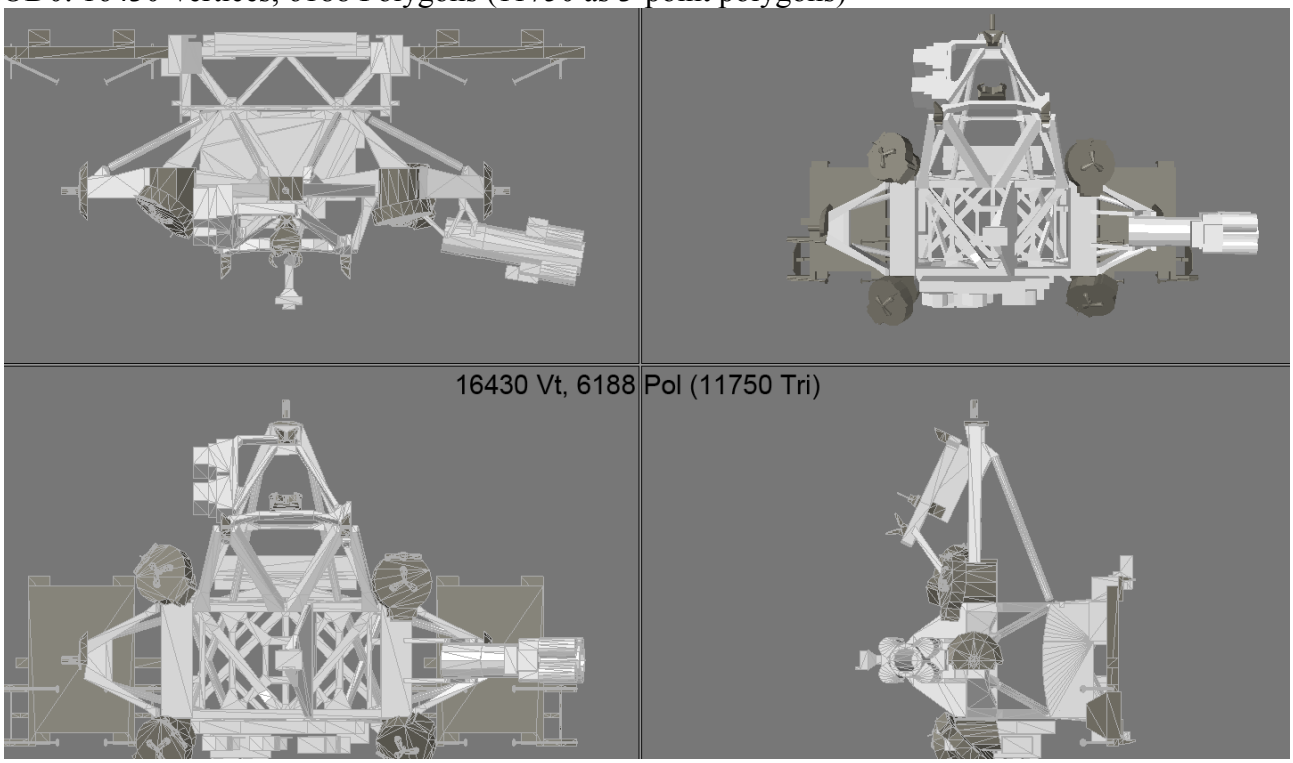
The first step in optimizing for LOD is to break the full model into major pieces. Each piece will then be able to choose its own LOD independently, allowing areas of the station very near to to observer to attain full detail, while distant objects of this large structure will be skimped on.

This model was created by the original artist as no less than 75 distinct sub-assemblies, each corresponding with a logical station module or independent part. This is typically done so that a particular part (for example, a solar array) can move or rotate independelty of the rest of the model. This sub-assembly scheme is already perfect for the LOD optimization. With other models that were not pre-built this way, the original structure may need to be broken up into sub-pieces in advance so each can be individually edited.

In this example we will start with the "Mobile Base System" module, part of the station's Canadarm2 robotic arm. The MBS is the base that attaches the arm to the station, and allows the arm to move around the station on a rail. This object is contained in a file named mss.lwo, in Lightwave 3D format. It is highly detailed with many tubular struts, filets and rounded surfaces.

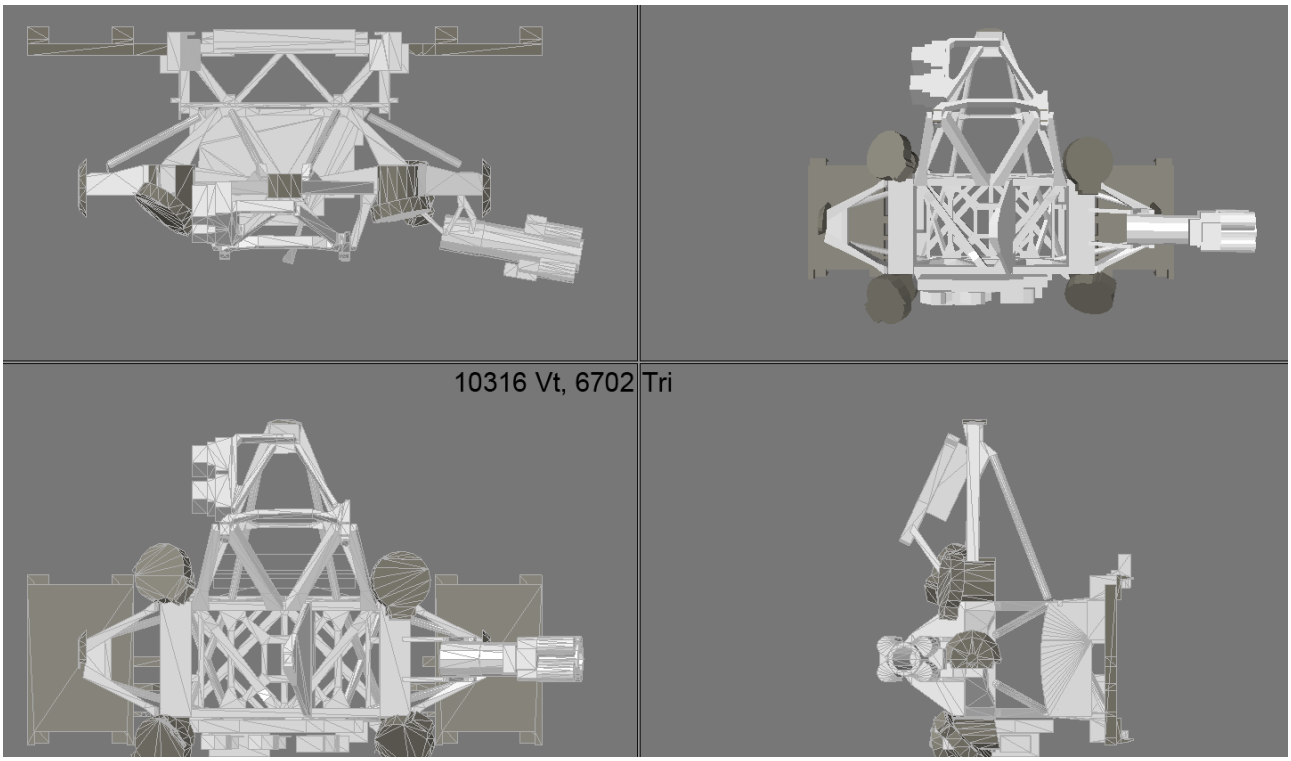
Example Model: ISS "MSS" module
Entire station is 600,000+ Polygons

LOD0: 16430 Vertices, 6188 Polygons (11750 as 3-point polygons)



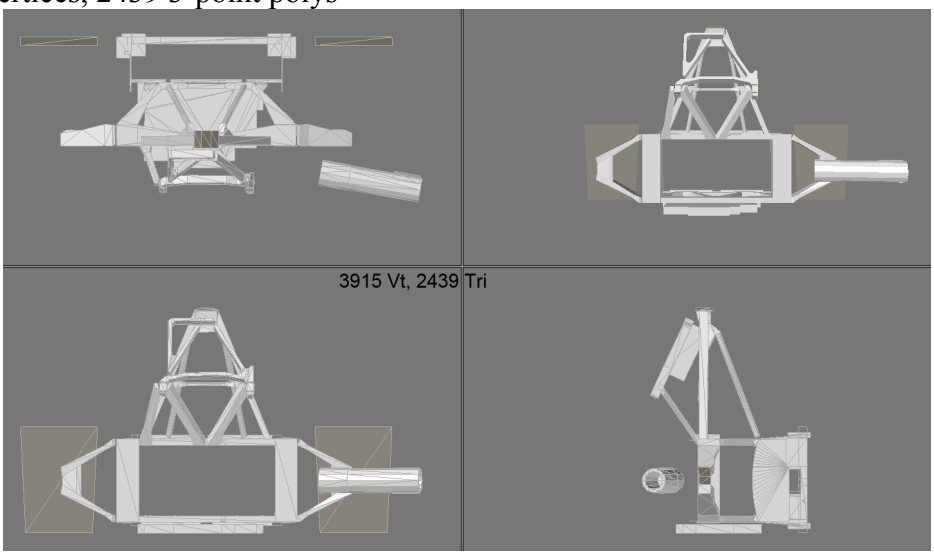
Here, LOD 0 refers to the original model configuration supplied by NASA/modeled by the original artist. It consists of 6188 polygons, about 1% of the total station's polygon count. Many of these polygons are more than three vertices each – when tesselated into 3-point polygons (which may be necessary during the load/display process) it results in 11,750 three-point polygons, or tri-gons.

LOD1: 10316 vertices, 6702 3-point polys



In LOD1, a number of the tiny details like antennas, small fittings and less-important structures have been removed. These omissions are unlikely to be noticed at any significant distance. The vertex count is 62% of the original, and the tri-gon count is 57% of the original. LOD1 should perform roughly twice as fast as LOD0.

LOD2: 3915 vertices, 2439 3-point polys

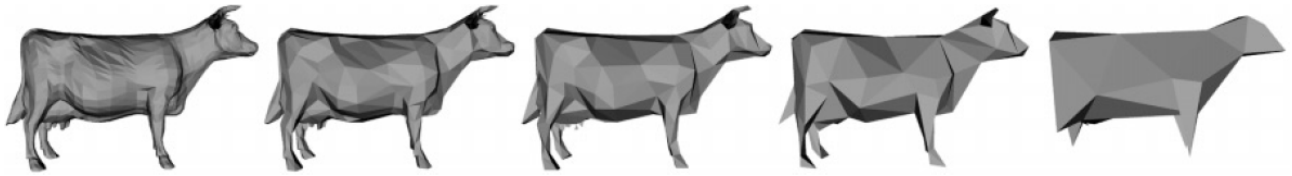


In LOD2, more substantial removals have been made, eliminating all the interior bracing structure,

most of the small framework and a large number of less-significant components. LOD2 has 24% of the vertices and 20% of the tri-gons as LOD0 and should perform roughly 4-5 times faster. The overall shape and silhouette is still essentially the same, and this model will suffice for any application where the screen-pixel extent of the model is 40 pixels or less:

There are many tools for reducing detail of models without manually making choices about what too keep or discard. Many of the automated approaches perform best on continuous surfaces.

QEM http://mgarland.org/files/papers/quadrics.pdf



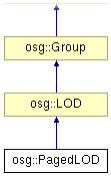
The osgWorks library (http://code.google.com/p/osgworks/) contains a Reducer feature that can be used for systematic simplification of models.

In OpenSceneGraph, the LOD Node type is derived from the Group node type and adds members concerning the center of the Node and the set of ranges associated with each Child.

PageLOD (PLOD)

OpenSceneGraph also supports an enhanced LOD Node known as PagedLOD. The conventional LOD node requires all of its immediate children be present in memory at-once (a requirement inherited from Group). For scenegraphs with large numbers of LODs, this can become prohibitively costly in terms of memory consumption, even though only one LOD child of each LOD Node is being drawn at any one time.

The answer to this predicament is the PagedLOD node. PagedLOD is derived from LOD, and adds the ability to defer the loading of child nodes until they are actually required. A PagedLOD node may have a child immediately present, or it may not have the child present, instead storing a path/name where the child node can be loaded from on-demand.



This loading is triggered during the Cull pass of scenegraph traversal and is done asynchronously, so large models that take time to load and prepare for display will not jam the redraw process (especially if OSG is running with multiple threads and on multiple CPUs or CPU cores). The background loading is done by the DatabasePager object and thread, and is triggered by the Cull traversal of the scenegraph

(executed by the main thread).

Every time the PagedLOD node is traversed, it determines which child node should optimally be displayed. If that child node is not currently loaded, a load request will be prepared, with a priority determined by how important the PagedLOD node feels it is to get its child loaded. This priority is influenced by user-configurable offset and scale factors associated with each PagedLOD child.

For example, if the PagedLOD determines it is just on the threshold of needing to switch to a higher LOD, the priority will not be very high. However, if it determines that it is well into the range where it should have the higher LOD, and somehow the new LOD still isn't loaded (perhaps other PagedLODs elsewhere were clogging up the DatabasePager queue with high-priority loads) then the PagedLOD will re-submit the request with a now-higher priority. Each time the scenegraph is traversed by Cull (effectively, once per frame) the PagedLOD criteria are evaluated and the request re-submitted until the DatabasePager completes the load and installs the new child into the PagedLOD.

The process is similar in reverse, though not identical. If the PagedLOD determines it needs to switch to a lower LOD, it can do so immediately, because (and this is important!) PagedLOD never unloads lower LODs. It wishes to keep them handy so it can down-res the scenegraph detail at a moment's notice if it feels that conditions require less drawing burden. This means that as you load more and more LOD children, memory resource consumption can pile up.

PagedLOD does however, unload higher LODs as they become unnecessary. When a higher LOD ceases being displayed, it is not immediately destroyed, but the moment is was last actually shown is recorded. If sufficient time elapses without it being needed anymore, it is scheduled for discard with the actual work again being performed by the DatabasePager thread. In the event that the high LOD is again requested prior to it being discarded, it is revived from the to-be-discarded pool and put back into action, with its lifespan-determining timestamp updated to indicate its renewed youthfulness.

PagedLOD is used extensively by VirtualPlanetBuilder/OSGDEM terrain model databases.

ProxyNode

A ProxyNode is another type of demand-loaded child, similar to PagedLOD. Also derived from Group, ProxyNode contains a list of child file paths. A ProxyNode can be configured to automatically load its children whenever the ProxyNode itself is traversed by the scenegraph Cull. Unlike PagedLOD, there are no distance/size criteria, no series of different children (each child is considered independent and all may be loaded/displayed at once) and no automatic unloading.

Other Detail-Management Techniques

For other circumstances where actual Level of Detail control is insufficient, there are additional techniques for reducing the amount of time spent rendering complex geometry.

Imposters

Imposters are a technique for pre-rendering a piece of the scene and then storing the rendered image as a 2D bitmap. On subsequent renders, the 2D bitmap is inserted into the scene in place of the 3D subassembly and oriented to face the camera. In the event the observer moves sufficiently to notice the odd perspective of the (now-stale) 2D image, the image will be automatically re-generated for the new

view-angle. Imposters are falling out of favor with modern graphics hardware because the 2D image transfer operation can become as much or more of a performance bottleneck than the actual geometry rendering. The osgImposter example and source demonstrates how this technique operates.

OcclusionQueryNode

The OcclusionQueryNode is another core osg class that utilizes the GL_ARB_occlusion_query extension to determine if (or how much of) a portion of the scenegraph is visible (not occluded or hidden behind other geometry). It allows complex models that are in the visible observer frustrum, and yet hidden by other objects to skip drawing. This is useful in scenes with generally-convex geometry that can easily hide other complex models. Scenes with lots of transparency and detailed objects with holes in them may not gain any performance from OcclusionQuery. OcclusionQuery can help in scenes where the actual geometry cannot be simplified because it needs to stay perfectly engineering-accurate for visualization or data-analysis purposes. The osgOcclusionQuery example and source demonstrate this technique. It is worth noting that the occlusion data is typically gathered using a simple, fast-to-draw shape (like a bounding sphere or box) big enough to contain the actual model. This shape is invisibly fed to the graphics card during a regular draw, and then the number of pixels that WOULD HAVE been visible arecounted. On the NEXT frame draw, this information is used to decide if the real shape should be made visible and drawn. So, the occlusion data is typically one frame out of date, but in practice this is not a real problem.