MIT Advanced Undergraduate Project: Generalized Conservative Visibility Algorithms

Peter Luka

May 30, 2003

1 Introduction

This AUP consists of two specific designs in the area of generalized conservative visibility algorithms for computer graphics.

First, a new algorithm for visibility computation is introduced. The basic idea is similar to that used in the traditional Weiler-Atherton hidden surface removal algorithm, but we make use of a constrained Delaunay triangulation (CDT) in view-space to enable object-precision depth resolution. We also apply the notion of conservative visibility to objects that may serve as either occluders and/or occludees, whether in fully detailed form or via various bounding volume representations. The algorithm is further accelerated through use of a bounding volume hierarchy.

Second, an outline is presented for the concept of VISAPI, a simple application programming interface (API) that would facilitate exploration in visibility algorithms, by supplying an interface between a rendering system and a visibility computation module. Given this API, different rendering systems and visibility modules could be hooked together and tested for performance with relative ease.

Both of these designs are presented with the intention that they will be developed further and actually implemented during the coming summer. The core functionality of the CDT visibility algorithm is described fairly rigorously here, but many suggestions for further exploration are included. Thus the next phase of work on it can be expected to include creation of an actual usable implementation of the core functionality, plus further delving into the questions left open here. The VISAPI is presented primarily as

a conceptual whitepaper, with some depth at points, but without strong commitment to any details given yet. The next phase for VISAPI is to proceed from the motivation and general directions given here to creating a careful, well-thought-out, detailed API design.

2 CDT-Assisted Conservative Visibility

We have developed a new algorithm for conservative visibility calculation. The intent is to make possible the rendering of extremely large datasets which can not be viewed efficiently on even the latest graphics hardware using existing algorithms. We specifically aim to deal with the "generalized" visibility case, whereby we do not wish to assume any *a priori* knowledge about the indoor/outdoor nature of the scene. An algorithm is desired which will perform well in a mixed walkthrough environment, composed of buildings with high levels of both interior and exterior detail, and a viewer that can travel anywhere in the scene (both looking into and out of buildings, notably).

The basic premise we are working on is that there is some existing back-end which is capable of rendering a reasonably sized scene, but not a scene of the magnitude we wish to explore. The ultimate bottleneck is of course that once the scene data becomes large enough that it cannot fit in graphics hardware memory (or even main system memory), performance will suddenly become intolerable as thrashing occurs. Hence we wish to do some processing that can classify large chunks of scene data as invisible from a given perspective, while pulling in as little of the actual scene data as it can (since every extra datum accessed may be the one that leads to a disk hit), and then retrieve and pass to the back-end renderer only that (hopefully much smaller) portion of the scene data that is left.

It is not at all important that the visibility classification be exact. It is vital however that it be "conservative." By this term, we mean that nothing may ever be misclassified as invisible. Classifying invisible items as visible however is fine, since the back-end renderer will deal with culling them out when computing the exact visibility solution on the remaining "conservatively culled" data set.

2.1 Background

The algorithm we have developed is superficially similar to the Weiler-Atherton hidden surface removal algorithm [Weiler-Atherton, 1977]. Specifically, we utilize the notion of being able to compute exact visibility on any dataset by starting with a rough front-to-back ordering of scene geometry, and then traversing it linearly, with recursion only to correct errors in the original sort. However, we will be applying that traversal to various abstractions other than "real" scene geometry and effect a conservative visibility cull rather than an exact visibility computation on the real scene.

Also we will not be using the clipping algorithm presented by Weiler and Atherton at all. Rather, we suggest a novel method for clipping and depth order resolution that requires only an efficient method for incrementally tessellating a 2-d space and indexing into that tessellation. For this, we recommend using the high-efficiency incremental CDT program written by Dani Lischinski [Lischinski].

2.2 Core algorithm

2.2.1 Simple CDT clipping and depth ordering

The first thing to describe is how we can use the CDT to effect object-precision clipping and depth ordering. To begin with, assume that we do have a correct front-to-back ordering for some set of polygons. Start with the first polygon and enter it in a rectangle representing the screen-space projected view at object precision, maintaining a CDT over that rectangle as you do so. Then proceed on down the list of polygons doing likewise, but at each step, first test the new polygon against the CDT. Any parts of a polygon which end up in spaces that have already been drawn in the CDT are occluded and may be culled. This concept is illustrated in figure 1, where polygon A fully occludes polygon B and partially occludes C. Given correct depth ordering, they are drawn in alphabetical order, and B can be safely removed since it is tested and found to be fully within the area already drawn by A. C is trickier, since it is not fully enclosed, and for conservative visibility there are two possible actions to take, both legitimate: subdivide C into two pieces, one of which is fully enclosed by A, and remove that one, or simply leave C marked as conservatively visible.

Next let's relax the constraint that our initial ordering is perfect and see what impact that has. To handle the possibility of polygons being traversed

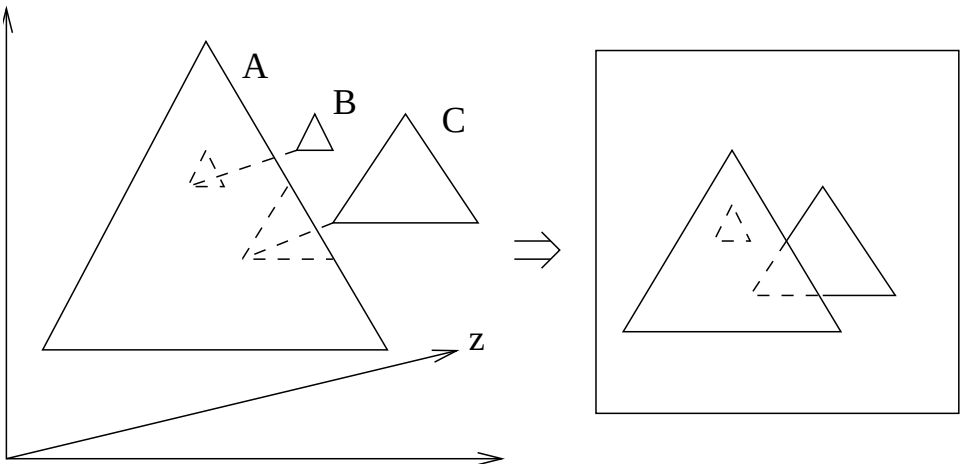

Figure 1: Depth resolution via front-to-back buffering.

out of order, there must be a method of detecting the misordering and correcting it. In the original Weiler-Atherton algorithm, this is easy because the output of their clipper is in the same domain as the input: polygons with holes, in 3-space. For our method as described so far, however, information is lost when writing the results into the CDT, since only two dimensions are retained. To keep depth information for later testing, we need to store, for each triangular region, the depth values at each of its corners. Note that we cannot simply store the depth value at each corner across the CDT, either, since a single corner may represent a z discontinuity and have multiple values for the various triangles coming together at it. This is illustrated in figure 2, where we see a single corner in a nominally 2-dimensional CDT is actually representing a pair of 3-dimensional points with different z values. Also note that naively attaching the z values to the triangular regions will not work, since while axially aligned polygons are shown in the figure, this is seldom going to be the case – 3 points determining the plane of a polygon must be stored, or more if complex objects are allowed. The best representation will probably involve maintaining back-references to the original geometry for each triangular region as the CDT is grown.

So far this seems like a heavyweight computation, and in fact as a general purpose exact hidden surface eliminator it is clearly inferior to Weiler-Atherton. However our desire only for conservative visibility is what makes it worthwhile. Because the goal is only conservative visibility, there are ac-

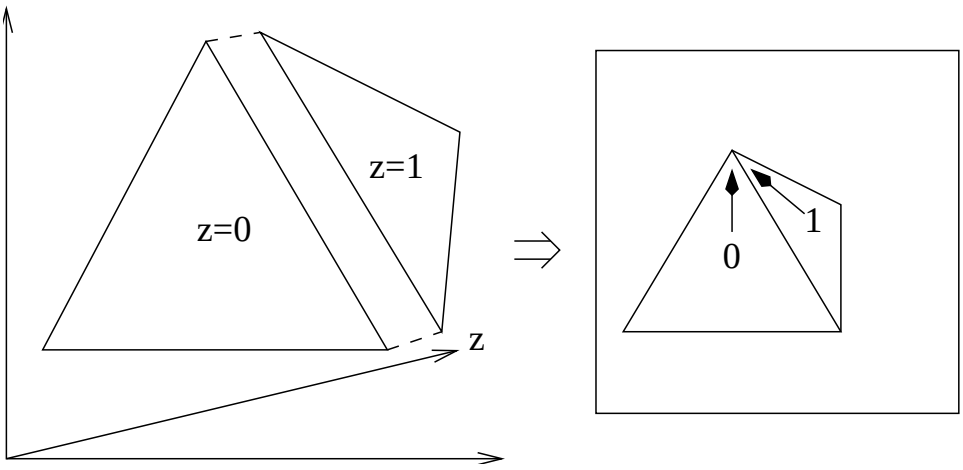

Figure 2: 3-d discontinuities in the "2-d" viewspace.

tually two ways out of the general depth ordering problem. We can solve the problem as described above by noting z values in the CDT, or we can shamelessly dodge the issue by simply marking as mutually visible any groups of objects for which a known depth ordering was not previously ascertainable. As we will see shortly, failing to impose depth order on even large amounts of the scene is not necessarily a loss, due to the prospect of region-based culling.

2.2.2 Occluders/Occludees and Bounding Volumes

The CDT-based clipping and depth-ordering algorithm thus far is still trying fairly hard to be a low-performance exact visibility algorithm, but given the sense of how it could work in an exact manner, and what basic relaxations are possible on ordering, we are ready to begin generalizing further to attain high-performance conservative visibility culling.

First note two facts about simple bounding volumes for occluders and occludees. When testing for whether a polygon is occluded yet, it is permissible to use any exterior bounding volume for it, since if that is occluded then so is the polygon which is fully contained inside it. Likewise, when drawing occluders into the CDT, any *inner* bounding volume may be used. Doing either of these does not yield a correct exact visibility solution any longer, but is perfectly valid for conservative visibility.

Suddenly, we have the ability to speed up the algorithm greatly by choosing bounding volumes cleverly. In order of increasing complexity, we shall

consider: polygon bounding areas, object bounding volumes, multi-object (region) bounding volumes, and hierarchical spatial subdivision structures.

We look first at simple bounding areas (fully enclosing occluders or fully enclosed by occludees). These can be used as a basic measure to accelerate our visibility algorithm when the polygons could otherwise be arbitrarily complex shapes. Shapes originally possessing lots of vertices, concavity, and holes can be translated to simple axially aligned rectangles, for example (a traditional choice of bounding box), thereby making the clipping tests much faster and the CDT insertion cost much lower. Note that a non-traditional choice of triangular bounds may be more appropriate for this algorithm however, because of the CDT insertion factor. It is worth exploring different types of bounding volume generation for this visibility algorithm.

Next, it is very sensible given typical hierarchical modeling techniques to group polygons together into "objects", for which it then makes sense to generate simplified bounding volumes. Objects can be incredibly complicated to provide a rich visual experience, but a much coarser bound is often just as good as the real thing for determining conservative visibility.

Above the level of objects (which are typically something like connected, closed, complex polyhedral surfaces), it is also intuitively appealing to look at bounding various groups of polygons, logically related or not, to simplify the geometry being processed by our algorithm. (Occluding one of these simple bounding volumes is a rapid computation if successful, which then culls potentially huge amounts of contained geometry immediately, never even touching the data structures for the actual contents.)

How to generate these groups is an open question, but a simple yet powerful observation is how these bounding volume visibility processing concepts can be legitimately applied to arbitrarily deep and complicated hierarchies of bounding "cells" in a global spatial subdivision structure. Many such structures have been explored and we will look next at applying some of them to reveal the full benefits possible with our visibility algorithm.

2.2.3 Spatial Subdivision Cell Structures

Cells in a spatial subdivision structure (such as a BSP tree or kd-tree) are basically just specially crafted bounding volumes. One desirable feature often present is the ability to provide a guaranteed-correct (and often unique) front to back traversal from any viewpoint in the structure. Another is a hierarchical nature, such that large bounding volumes can be tested for a

decision procedure first, that will either return true or false for the entire volume (thereby affecting all the contents at once), or get applied to sub-volumes recursively if the parent volume's result was indeterminate. (These features are each true for both BSP and kd-trees.)

Thus it is safe to make the following assumptions when using such a subdivision. We can get a valid front-to-back ordering of cells from any viewpoint, just from the tree structure itself, and therefore any contents of those cells which are fully contained in the cells will also be ordered correctly – no further depth testing is needed! If it can be shown that a cell is fully occluded, then all of its subcells, and any fully contained objects in the cell, are known to be occluded as well and do not need to be examined at all. If a cell is not deemed fully occluded, it may still have subcells which can be examined before traversing further back, and be subject to individual occlusion, without violating our ordering assumptions.

2.2.4 Core Algorithm

Assuming that all cell contents are fully contained, we can now propose a very simple core visibility algorithm that integrates the observations made so far. Start with the tightest cell around the current viewpoint, and traverse away from the viewer (possibly expanding into larger cells as you go). At each step, take any objects contained in the current cell that are to be treated as occluders (or an inner bounding volume thereof), and draw their projection into the screen-space CDT. First, however, test the cell boundary against the CDT so far: if it falls entirely within a drawn region (from previous, necessarily more-forward cells) then it is occluded and all its contained occludees may be discarded!

Figure 3 illustrates this in two dimensions. Neither of the large lines occludes much alone, but they are traversed in the first two cells, and their subsequent shadow volume (plus area automatically culled outside the frustum) occludes the entire right half of the tree. The next volume to traverse should be the parent volume covering the entire right half, which tests as fully enclosed in the already drawn area of the view-space projection and gets removed. Implicitly, all its child nodes and their contents get removed from visibility candidacy as well, without ever having to be explicitly traversed. (If full occlusion did not occur, recursive subdivision could happen, and the sub-cells would be tested for total occlusion. Only failing occlusion on all cells do we end up testing individual objects – or deciding to just pass

them through as conservatively visible once it seems that too much time is going into our calculations.)

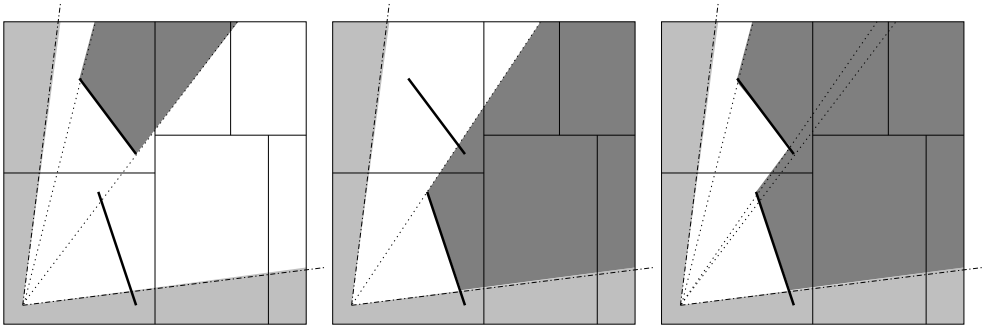

Figure 3: Two polygons together occlude all of the right hand parent cell.

Note that this "core" algorithm implicitly retains any incident (non-contained) occludees as conservatively visible, since it can draw no conclusions about them. This is an important case for which to look at alternate treatments, as we do in the extensions below.

2.3 Extensions to Core Algorithm

Beyond the core algorithm described above, various extensions are possible to the general idea. The following are some that we definitely intend to explore during the next phase of development.

2.3.1 Non-Contained Occluders

It is not always possible or desirable to to make all objects be fully contained in some reasonably tight cell boundary. If we wish to use such *incident* (not fully contained) objects as occluders, then a naive implementation may yield the error illustrated (in 2-d) by figure 4.

In this figure, we see a large polygon incident on the first (left-hand) cell, and some small polygons both in front and behind it, fully contained in the second (right-hand) cell. How to handle the incident polygon as an occluder is not clear, even though it would be desirable to do so as it is large and could potentially occlude lots of other things. If we encounter the large polygon in the first cell, and treat it as a normal occluder, writing it into the CDT, then the polygon in front of it will be incorrectly removed when the

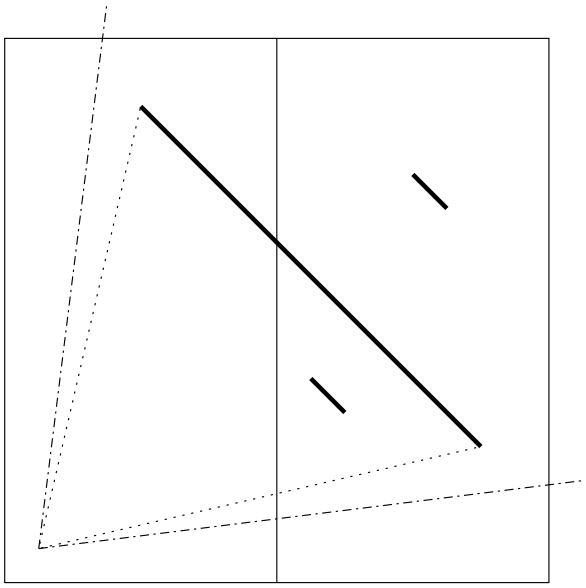

Figure 4: Incident polygon. (Naive treatment yields incorrect results.)

second cell's objects are explored and deemed fully enclosed by its shadow and implicitly behind it (which is the incorrect assumption). Simply not letting the incident polygon act as an occluder (not writing it to the CDT) gives a correct conservative visibility solution, but makes the clearly occluded object behind it get passed through as visible, simply because we were not smart about dealing with the large incident occluder in this particular spatial subdivision. The alternative to this choice (given the existence of useful but incident occluders) is to do the complicated depth storage in the CDT. Thus we pay a high price because suddenly we need to start maintaining and testing depth in the CDT for everything whereas before we did not need to do so for anything.

What follows is one possible way to compare the relative value of each method. If most occluder geometry can be stored as contained (perhaps with fragmentation into several contained polygons), then we should try to avoid the overhead of depth buffering at the cost of having a slightly looser visibility bound. Notably if a sufficiently general spatial subdivision (e.g. BSP) is used, such that containment can be almost guaranteed (the exception being cases like cyclic overlap), then depth buffering should not be done as it will gain almost nothing and cost a lot in overhead.

On the other hand, if for example an axially-aligned tree structure is used, and an antagonistic geometry is present which can not be separated at all, then clearly depth buffering must be used to gain anything at all from this algorithm!

It is worth testing the relative computational weights of these methods experimentally. We should also see if we can come up with a good heuristic for switching between them based on a precomputation pass over the contents of the spatial subdivision DB.

2.3.2 Non-Contained Occludees

Similar analysis can be applied to incident occludee geometry. For now it is simply worth noting that such a consideration exists, but when proceeding to implement the core algorithm for real, and analyze extensions more closely, it seems clear that coming up with a clever treatment for incident occludees would be a major performance gain (or loss, if done incorrectly).

2.3.3 Silhouettes (and other advanced bounding reps)

Various advanced bounding representations for geometry may be possible that could significantly assist this particular algorithm in being highly efficient. For example, it seems intuitive that if we can group geometry into objects, where one object is known to be fully in front of another, then the 2-d silhouette of the object projecting in viewspace is entirely sufficient for the CDT update. It is desirable to explore the possibility of maintaining silhouette data efficiently. In addition to just detecting fully ordered cases, it may also be possible to attach "just enough" contour information to a silhouette to make dynamic depth testing highly efficient.

3 Visibility API

The second major component of this AUP is the preliminary design and motivation for a visibility API (VISAPI). Much is left to be done on the real design for this, but it is important to see why the final goal is desirable, and what the overarching impacts of the high-level goal should be during the first real design phase.

3.1 Abstract Goals

The high-level purpose of VISAPI is to facilitate exploration in visibility algorithms, by supplying an interface between a rendering system and a visibility computation module. Given this API, different rendering systems and visibility modules could be hooked together and tested for performance with relative ease. Currently no such API exists. Hence it is desirable for us to work on making such a thing exist at all, but also important to proceed carefully and think hard about the design decisions we make in the process, as there is not much in the way of known good prior art to work from.

3.2 Design Paradigms

Probably the biggest challenge in designing VISAPI will be deciding exactly where to put the abstraction barrier between the renderer and visibility module. We need something that has "obviously necessary" features directly integrated in the core data structures, for efficiency, but we also do not want to make too many assumptions about what is or is not going to be part of any given user system. It will be necessary to say at some specific point that a given visibility algorithm is strange enough that it can not be implemented efficiently using this API, but that such an algorithm is unlikely enough to be useful that it is okay.

The current intention is to design a core API that includes direct support for various auxiliary data structures that we consider important to common visibility algorithms, and for typical procedure calls. This core functionality is only special in that support for it is hardwired in the API for maximal efficiency though – a given user of the API may or may not fill in and make use of all the structures. It should also be possible to add any further data and procedural support via a generic interface.

3.3 Early Design Suggestions

Some data that clearly must always be passed in directly on entry to the visibility module are current viewpoint location and view frustum information. Specific additional data structures that we do also wish to have in the core specification include: cells, cell portals, and winged-edge or silhouette data (for occluder fusion algorithms). Procedural support should be included for ordered (front-to-back or back-to-front) traversal of the dataset, and should

probably be implemented via a callback interface – the user application would set up the visibility module by passing in a pointer to any traversal methods it implements, and the visibility module can then call any that are available as it needs them.

Additionally, it is very desirable to have some way to preserve state between calls to the visibility module in a way that allows reuse of existing data and exploitation of temporal coherency in a typical walkthrough style of application. How best to do this is not yet clear, but certainly it would be a bad idea to specify the core protocol such that the entire scene (even by reference) must be passed in again on every call to the visibility module. Beyond that, references should be used, the visibility module should be considered internally stateful, and probably some core methods for informing the visibility module of changes to the scene database would be good, thereby allowing it to maintain an internal processing cache, updated only when it detects changes in its working invariant, or when the application signals a change to the actual scene data.

4 Summary Conclusions

This term has seen a lot of brainstorming over ways to attack the hard problem of visibility acceleration for generalized extremely-large-scale datasets. The results described here encompass the best ideas so far, in a form sufficiently mature to be further developed by the graphics group over the course of the summer, with confidence that the results of pursuing them will be meaningful and productive. It is expected that these methods will help address both the immediate problems inherent in rendering of the MIT and Cambridge datasets, plus serve as a platform for more broad-based explorations in the long run, both at MIT and elsewhere.

References

[Lischinski] http://www.cs.huji.ac.il/ danix/code/cdt.tar.gz

[Weiler-Atherton, 1977] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (Proc. of SIGGRAPH 77)*, 11(2):214-222, 1977.