Extraction of 3D Navigation Space In Virtual Urban Environments

Türker Yılmaz and Uğur Güdükbay

Bilkent University

Department of Computer Engineering

06533 Bilkent, Ankara, Turkey

Tel: +90 (312) 290 20 91, **Fax:** +90 (312) 266 40 47

e-mail: {yturker, gudukbay}@cs.bilkent.edu.tr

Corresponding Author: Türker Yılmaz

Abstract

Urban scenes are one class of complex geometrical environments in computer graphics. In order to develop navigation systems for urban sceneries, extraction and cellulization of navigation space is one of the most commonly used technique providing a suitable structure for visibility computations. Surprisingly, there is not much work done for the extraction of the navigable area automatically. Urban models, except the ones where the building footprints are used to generate the model, generally lack of navigation space information. Because of this, it is hard to extract and discretize the navigable area for complex urban scenery. In this paper, we propose an algorithm for the extraction of navigation space for urban scenes in three-dimensions (3D). Our navigation space extraction algorithm works for scenes, where the buildings are in high complexity and the virtual scene is constructed by populating these buildings without making any assumptions. The building models may have pillars or holes where seeing through them is also possible. Besides, for the urban data acquired from different sources which may contain errors, our approach provides a simple and efficient way of discretizing both navigable space and the model itself. Furthermore, terrain height field information can be extracted from the resultant structure, hence providing a way to implement urban navigation systems including terrains.

Keywords: Urban visualization, occlusion culling, cellulization, 3D navigation, view-cells.

1 Introduction

Urban visualization strongly requires culling of unnecessary data in order to navigate through the scene at interactive frame rates. There are efficient algorithms for view-frustum culling and back-face culling. However, occlusion culling algorithms are still very costly. Especially, object-space occlusion culling algorithms strongly need precomputation of the visibility for each view-point and for each viewing direction.

Since the amount of data to be stored increases drastically, cellulization of navigation space, thereby providing way to decrease the amount of the necessary information caused by preprocessed occlusion culling, is very crucial. For walkthroughs of architectural models, cellulization is easy because rooms naturally comprehend to cells [1]. However, for walkthroughs of outdoor environments like urban sceneries, cellulization is accomplished mostly in model design time [2], by using semi-automated ways [3] or by using building footprints where the models' complexity is limited [4, 5, 6, 7].

Almost all occlusion culling algorithms calculate occlusion with respect to ground walks, thereby eliminating the need for a 3D navigation space. However, for a general fly-through application, a cellulized navigation space can provide a suitable environment for a precomputable visibility information. The algorithm presented here calculates and extracts a navigation space for urban scenery where the models of buildings are highly complex. The buildings may have balconies, pillars, fences or holes where it is possible to see through them. No assumptions or restrictions on the model are made. The navigation space extracted looks like a jaggy sculpture mold and it is used in the cellulization process required for the occlusion culling algorithms. Besides, terrain height field information can be extracted from the resultant structure, hence providing a way to implement urban navigation systems with terrains.

In the next section, we give related work on the subject. In following sections, we describe our approach and the algorithm we developed as a solution to this problem.

2 Related Work

The method proposed in this paper automatically constructs the navigation space for complex urban scenes like the one in Figure 1. If 3D navigation is not required, the resultant navigation space structure can also be used for the navigations that are bounded to the ground.

Generally, navigation space extraction for building interiors is not necessary, because rooms of the architectural model naturally correspond to cells, where it is not important to cellulize the rooms again as in [1, 8, 9]. In [1], the cell-to-cell visibility is defined, where a portal sequence is

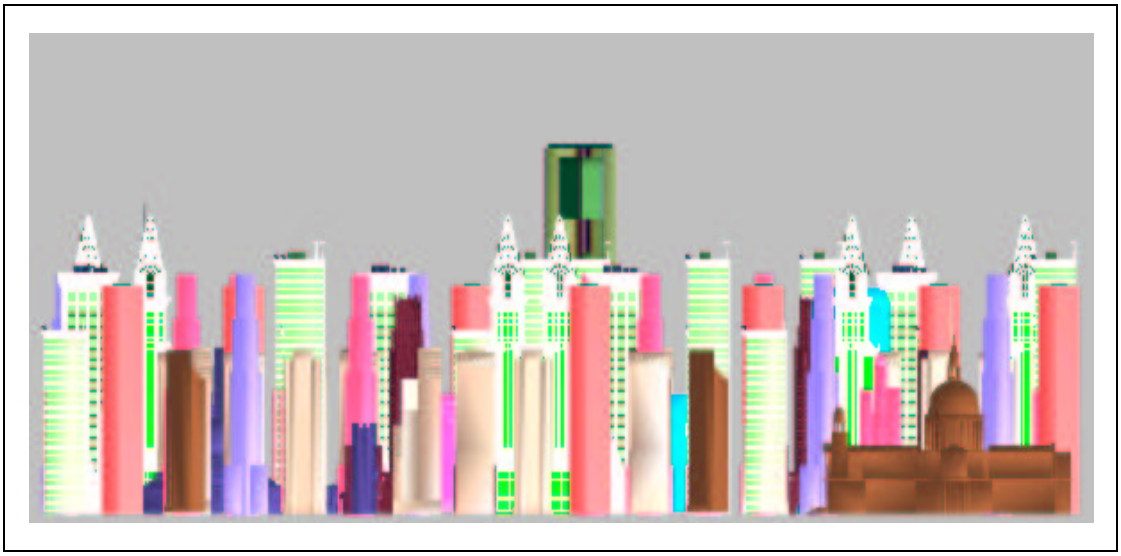


Figure 1: A sample urban scene with nearly 300 buildings and 500,000 triangles.

constructed from a cell to the others if a sightline exists, thereby making a whole cell navigable. In [4], the user is assumed to be navigating on the ground. Besides, the city they use was built using footprints, where the ground information becomes explicit.

Sometimes it is quite sufficient to determine the navigable area during model design time. In [2], the developed walkthrough system accepts streets or paths as navigable, where a triangle is defined as either a street or a path triangle. This means that in order to navigate over a triangle, it must be a part of a street or a path. Besides, only triangles are accepted for view-cells. Both of these properties make extending user navigation into the 3D space very challenging, although the algorithm for occlusion culling that the authors develop is suitable for this extension.

In [3], the user is assumed to be at two meters above streets. Besides, the created model has straight streets, making navigation space determination straightforward. Likewise, in [5, 6, 4] the authors also implement navigation assuming the user is on the ground, where the navigable space information is explicit and in 2D.

We need to mention that our aim is not to provide an environment where collision tests are optimized, although the resultant structure provides this. Specifically, we aim to create a suitable data structure where cellulization for occlusion culling preprocess becomes a simple task for

complex urban scenes.

As a summary, except [10], where 3D navigation is performed using parallel computing, almost all other algorithms perform 2D navigation where extraction of navigation space is straightforward and model complexity is limited into some extend. Hence, a simple and yet powerful navigation space determination in 3D becomes vital for 3D navigation applications.

3 Data Structures

Here, we present the data structures we used in our algorithm. We need to mention that the input data formats do not have significant importance on the efficiency of the algorithm, because our approach is nearly independent of the type of the input data. The only assumption is that the scene must be composed of triangles. One of the most common data format is *dxf* data format created by *Autodesk, Inc*. The components of this format makes up a huge list, however *ENTITIES* and *3DFACE* components with vertex lists of triangles are the only ones that we need. A part of the scene file is shown in Figure 2. The data structure used to store this file is a forest type data structure equipped with suitable fields designating the parameters of the other algorithms. The data structure is shown in Figure 3.

As is shown in Figure 3, `geomobject` structure is the main structure holding necessary pointers to the other structures. It stores the name of the object and the number of triangles making up of this object. It holds pointers to the bounding box of the object, the very first triangle of the triangle list, the parent of the octree defining the navigation space found within the box of the object and next object in the order. The scene file is read from the storage and a list of triangles are tied to each object with necessary vertex information defined in `tri` and `vertex` structures. The `octree` and `seed` structures will be used later while extracting the navigable area and discretized objects.

```
ENTITIES
0
3DFACE
8         --> Object Start
o156      --> Object Name
62
126       --> Color Code in autodesk color table
10
775.211   --> X-coordinate of first vertex of the triangle
20
2075.3    --> Y-coordinate of first vertex of the triangle
30
-387.872 --> Z-coordinate of first vertex of the triangle
11
736.085   --> X-coordinate of second vertex of the triangle
21
2075.3    --> Y-coordinate of second vertex of the triangle
31
-320.284 --> Z-coordinate of second vertex of the triangle
12
745.623   --> X-coordinate of third vertex of the triangle
22
2075.3    --> Y-coordinate of third vertex of the triangle
32
-306.143 --> Z-coordinate of third vertex of the triangle
13
745.623   --> Last X-coordinate repetition
23
2075.3    --> Last Y-coordinate repetition
33
-306.143 --> Last Z-coordinate repetition
0
```

Figure 2: A part of the *dxf* scene file.

```
struct vertex {
    float x;
    float y;
    float z;
};
struct tri
{
    struct colors color;
    struct vertex v1;
    struct vertex v2;
    struct vertex v3;
    struct vertex normal;
    struct tri *next;
};
struct octnode
{
    int level;
    char no;
    float minx, miny, minz, maxx, maxy, maxz;
    char type;     //1:parent, 2:inner, 3:leaf
    struct octnode * parent;
    struct octnode * n[8];
    char empty;
};
struct geomobject
{
    char name[12];
    int number_of_triangles;
    struct boundingbox * b_box;
    struct tri * first;
    struct octnode *octtree;
    struct geom_obj* next;
};
struct seed
{
    int xoff,yoff,zoff;
    int tag_fill;
};
```

Figure 3: The data structures used in our implementation.

4 Navigation Space Extraction Algorithm

The navigation space extraction algorithm mainly consists of two phases: 1) the seed test, and 2) the contraction and the octree construction part. In the first phase, the bounding boxes of objects are calculated and a seed box is travelled around each object to find the blocks that touch the surface of it. Filled seeds are later passed to a contraction algorithm in which the octree structure for the navigable area is constructed and the mold of the object is extracted. It should be noted that it is possible to find all holes and passages inside the objects within a user specified threshold using this approach. The flow diagram of our algorithm is shown in Figure 4.

After reading the scene database from the input file, the algorithm first calculates the bounding boxes of each object in the scene. Object discrimination is done while constructing the scene file and each object (i.e., building) is defined with a header and triangles are inserted into the list according to the object names, which is a property of the `dxf` file format. The bounding boxes are calculated in a straightforward manner and stored in the relevant structures. Seed testing and contraction parts of the algorithm take place in these bounding boxes and all space out of these boxes are accepted to be navigable.

4.1 Seed Testing

The seed testing phase is based on a box with a size of a user-defined threshold. We call this size as *threshold* because it defines the roughness of the extracted mold of the object. The time needed to extract the navigable area strictly depends on the size of the seed box.

At each time, the seed box is located at the next position in the bounding box of the object and tested against the triangles of the object. If any triangle is touched by the *seed box*, then that location is marked as *filled*, discretizing the occupied space by the object. A seed box and a triangle can interact in three different ways. These cases are shown in Figure 5.

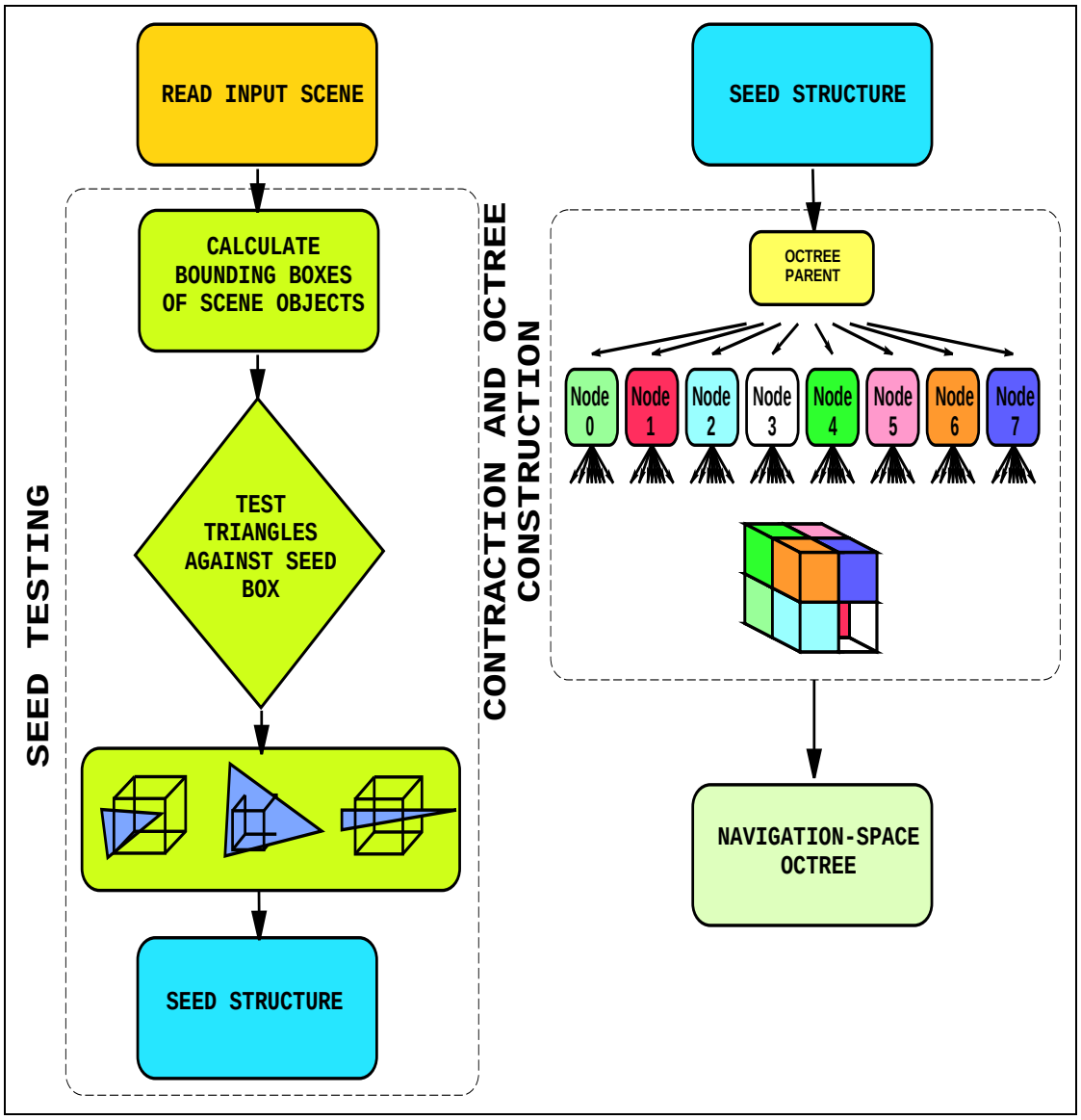
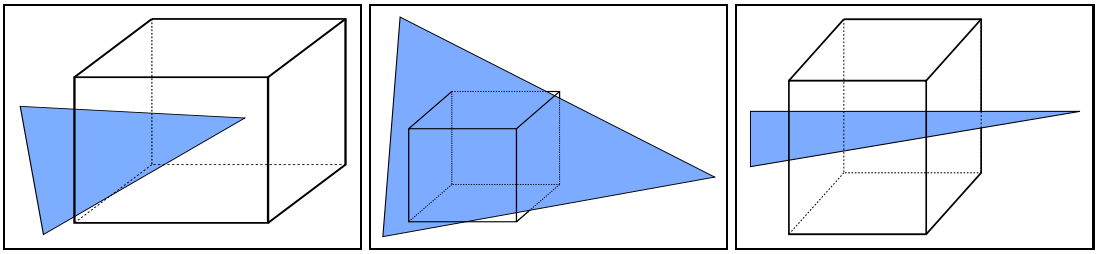


Figure 4: Flow diagram of the *Navigation Space Extraction* algorithm.

The first case is where any vertex of the triangle is inside the seed box. This case is the easiest to determine, where a range test gives the intended result. It is illustrated in Figure 5(a). The next case is illustrated in Figure 5(b). Here, a seed box touches the triangle but none of the vertices of the triangle is inside the box but the triangle plane intersects the edges of the seed box. We used ray tracing for the detection of such cases. The pseudocode algorithm for this case is given in Figure 6. In this algorithm, the main idea is to shoot rays from each corner of the seed box to each direction. The last case, where the triangle penetrates the seed box without touching any of its edges, is handled in the opposite way (see Figure 5(c)). In this case, the rays are shooted from the vertices of the triangle and checks are made against the surfaces of the seed box.



(a) (b) (c)

Figure 5: Test cases: (a) *the easiest case, where any vertex is inside the seed box*; (b) *the vertices of the triangle is not inside the seed box, but the seed box edges intersect with the triangle surface*; (c) *the last case, where the triangle edges intersect with the surfaces of the seed box.*

4.2 Contraction and Octree Construction

After the seed test phase is finished, we have a discretized version of the scene objects, where we exactly know the spatial locations occupied by the triangles of the object within a certain threshold. Although, the occupied seed cell information is enough for us to determine the navigable space, its memory requirement is very high. Therefore, we need to contract this area and determine the navigable space using another structure requiring less memory space. An octree structure is used for this purpose.

The octree structure constructed is shown in Figure 4. The algorithm for the octree construction

```
Algorithm to determine the second case
    calculate box coordinates
    for each triangle of the object
        define plane of triangle
        for each corner of the seed box
        shoot rays towards the other neighboring corner
        if (the ray hits the triangle)
            calculate the intersection point
            translate the point to the origin
            for each edge of the seed box
                if (the horizontal line to the right
                    of the intersection point intersects
                    the triangle odd number of times)
                    report as inside
                else
                    report as outside
        for each neighboring corners of the seed box
            if (any two neighboring corners have
                intersection on the triangle)
                report as INTERSECT
        report as NOTINTERSECT
```

Figure 6: The pseudocode for the detection of the case where a triangle passes through a seed box but none of the vertices of the triangle is inside the seed box and the triangle plane intersects with the seed box edges.

simultaneously contracts the space occupied by the seed cells into larger blocks of space as much as possible, thereby eliminating the need to keep filled cell information for every small seed. This is done as follows.

The constructed octree allows the navigable space to be traversed hierarchically. The algorithm first sets the bounding box of the object as the parent of the octree and checks if there is any filled cell within the range of the node. If there is any filled cell, then it recursively subdivides itself into eight octants and repeats the same procedure for the newly created nodes until the size of the node decreases below the size of the seed box or there is left nothing but empty cells. This structure is tied to the spatial forest of octrees after all the scene objects are processed. It should be noted that the numbering scheme as seen in the Figure 4, provides neighboring

information of the octree nodes.

4.3 Resultant Structure

The algorithm is concluded after all the scene objects are processed. The resultant octree structure represents the navigable area, where bounding boxes are tied up to spatial forest of octrees. An example of the created structure is shown in Figure 7. After this, the user exactly knows the locations in 3D where navigation is possible. The user will also know where the objects are and a hierarchical subdivision of them will also be provided as will be explained in the next section.

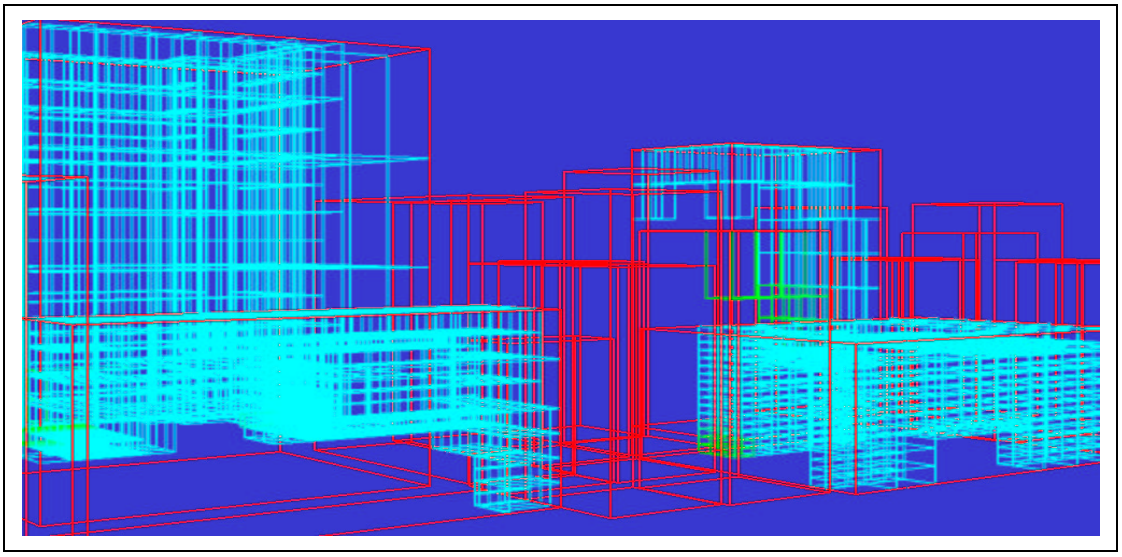


Figure 7: The octree forest of a scene created by the navigation space extraction algorithm having 24 buildings.

It should be noted that we did not make any assumptions on the type of scene objects or on their respective locations, while determining the navigable space information. The objects may contain any type of architectural features, such as pillars, holes, balconies, etc. Our algorithm indiscriminately finds the locations where there is not any object part (i.e., triangle). This property makes our approach very suitable for the models that are created from different sources, because the only information needed is triangles, where most model formats certainly provide. Other graphics primitives such as lines or polygons having more than three vertices are not handled, but they can easily be converted to triangles.

5 Creating Object Structure

In addition to creating the navigable space information of the scene database, it is very easy to create the octree for the scene itself, where further calculations on them can be performed. One important process that can be applied to the hierarchy is occlusion determination, where hierarchical calculations are strongly needed.

After the contraction process is performed and the octree for navigation space constructed, the octree construction algorithm is repeated once more seeking full seed cells to discretize the object. This time the contraction part of the algorithm finds the cells which contain geometry in it, whereas we did it for empty cells while constructing the octree for the navigation-space. The same procedure is applied to each object and scene objects are tied to the spatial forest of object octrees. We are left with the two forests of octrees, one with the navigable space information and one with the object hierarchy, which are useful for 3D navigation and scene object processing, respectively.

6 Conclusion

During the development of navigation systems for urban sceneries, the navigation determination becomes one of the most vital parts of the work. The navigation space determination is simple for the scene databases where the building footprints are used and the navigation is bounded to the ground. However, for the systems that need 3D navigation and the scene database is composed of complex objects where footprints do not define the navigable area, the navigation space determination becomes one of the most daunting tasks.

At this point, our approach becomes a solution to the definition of navigable space determination. It also constructs the hierarchical scene database as an additional feature. One important feature of our approach is architecture independence of the scene objects and data type independence. The building models may have pillars or holes where seeing through them is also possible. The method can be applied to any type of unstructured scene files composed of objects such as buildings. The application of the method produces two octrees; one containing the

definition of the navigation area and another another one containing the scene hierarchy, both in the form of the forest of octrees. In the future, we plan to develop platforms that make use of these structures that are simple and adaptable.

Acknowledgements

The first author is supported by a Ph.D Scholarship from Turkish Scientific and Technical Research Council (TÜBİTAK).

References

[1] Thomas A. Funkhouser, Carlo H. Sequin, and Seth J. Teller, "Management of large amounts of data in interactive building walkthroughs," in *ACM Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, 1992, vol. 25(2), pp. 11–20.

[2] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion, "Conservative volumetric visibility with occluder fusion," in *ACM Computer Graphics (SIGGRAPH '00 Proceedings)*, 2000, pp. 229–238.

[3] Laura Downs, Tomas Möller, and Carlo H. Séquin, "Occlusion horizons for driving through urban scenes," in *ACM Computer Graphics (SIGGRAPH '01 Proceedings)*, 2001, pp. 121–124.

[4] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg, "Visibility preprocessing with occluder fusion for urban walkthroughs," in *Proceedings of Rendering Techniques*, 2000, pp. 71–82.

[5] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg, "Fast walkthroughs with image caches and ray casting," *Computers & Graphics*, vol. 23, no. 6, pp. 831–838, 1999.

[6] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech, "Conservative visibility preprocessing using extended projections," in *ACM Computer Graphics (SIGGRAPH '00 Proceedings)*, 2000, pp. 239–248.

[7] Dieter Schmalstieg and Robert F. Tobler, "Exploiting coherence in 2.5-D visibility computation," *Computers & Graphics*, vol. 21, no. 1, pp. 121–123, 1997.

[8] C. Saona-Vázquez, I. Navazo, and P. Brunet, "The visibility octree: a data structure for 3D navigation," *Computers & Graphics*, vol. 23, no. 5, pp. 635–643, 1999.

[9] Carlos Andújar, Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet, "Integrating occlusion culling and levels of detail through hardly-visible sets," *Computer Graphics Forum*, vol. 19, no. 3, pp. 499–506, 2000.

[10] Douglass Davis, William Ribarsky, T. Y. Jiang, Nickolas Faust, and Sean Ho, "Real-time visualization of scalably large collections of heterogeneous objects," in *Proceedings of IEEE Visualization '99*, 1999, pp. 437–440.