Comparison of Accelerating Techniques for Discontinuity Meshing

Karel Nechvíle
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic
kodl@fi.muni.cz

Petr Tobola
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic
ptx@fi.muni.cz

Jiří Sochor
Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic
sochor@fi.muni.cz

ABSTRACT

Creating an appropriate mesh is one of demanding tasks of many global illumination algorithms. Discontinuity meshing proved to diminish artifacts caused by other meshing strategies. Since naive discontinuity meshing would produce a great amount of geometric computations, accelerating techniques are usually involved. In this paper, we present results obtained with help of k-discrete orientation polytopes (k-DOPs) and oriented bounding boxes (OBBs) and compare them to previously used accelerating schemes like voxels, BSP-trees and octrees.

Keywords: discontinuity meshing, accelerating strategies, discontinuity surface propagation, bounding volume hierarchies

1 INTRODUCTION

In this article, we concentrate on radiosity methods that divide the environment into a set of patches and compute a radiosity on each patch. The radiosity function on a fully visible surface is a smooth function but intervening objects cause changes in its course [Hec92, LTG92]. In order to trap these changes, classical radiosity algorithms refine the mesh in places where radiosity function is changing sharply. However, such a posteriori meshing may not be effective.

Fortunately, it is possible to determine the location of discontinuity boundaries a priori. Discontinuities in radiosity function correspond to changes in visibility between the light source and the patch and they are of geometric nature. We will call an algorithm that computes shadow boundaries prior to radiosity computation as *discontinuity meshing*. Discontinuity meshing generally consist of three main actions.

- The selection of discontinuity surfaces.
- The propagation of those surfaces through the environment.
- The construction of meshes containing the discontinuity curves.

We would like to note that this article touches mainly the second item. Any issues concerning importance of individual discontinuities or combinations of meshes from several light sources are not handled here. On the other hand, we do not exclude discontinuities caused be so called EEE events. Our main intention was to analyse a speedup of computation using recent accelerating techniques like k-DOPs and OBBs for single light source in mind. The rest of the article is organised in the following way.

Firstly, we introduce basic definitions related to discontinuity meshing. Then previously used accelerating techniques are discussed. Next we outline the k-DOPs and an OBBs and show how we applied them to speed up the computations. After that, we present some technical details from implementation. Finally, we present comparison to other methods. To sum up, we discuss directions for further development.

WSCG SHORT PAPERS proceedings
WSCG 2003, February 3-7, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

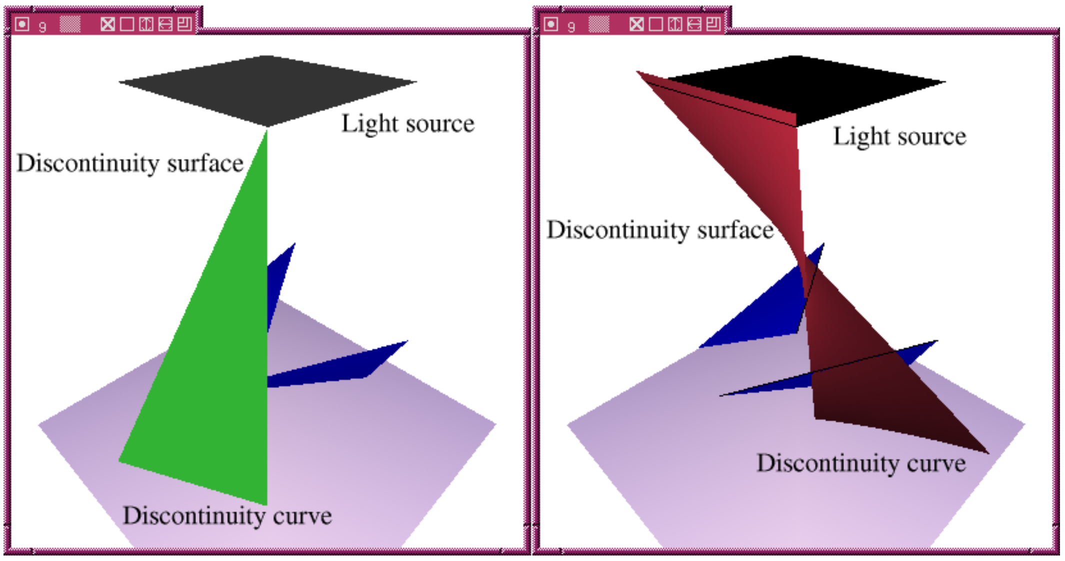

Figure 1: EV and EEE discontinuity surface

2 VISUAL EVENTS

As stated above, changes in radiosity function correspond to changes in visibility. Visibility changes, or *visual events*, are related to the interaction of edges and vertices in the scene [GM90, GCS91]. In this context, vertices and edges are sometimes called *features*. According to individual features involved, events can be split into:

- *EV events*, caused by interaction of a vertex and an edge, and
- *EEE events*, caused by the interaction of three edges

Points in space where a ray connecting the features exist (without intersecting the interior of any other polyhedron in the scene) form a *discontinuity surface*. The intersection of a discontinuity surface with a scene face is called *discontinuity curve*, see 1. Discontinuity surfaces given by EV events form planar wedges and their intersection with a face is a line segment. EEE events give to rise a ruled quadric surface and corresponding discontinuity curve is a quadric curve.

From pragmatic reasons, we further refine the possible events into five categories. We will use this notation later when we will discuss accelerating techniques in detail. The suffix n is introduced for some events in order to emphasize that the features involved come from non-emitter faces.

- **VE events** - events involving light source vertex and a scene edge.
- **EV events** - events given by light source edge and a scene vertex.
- **EEE events** - events involving one light source edge and two environment edges.
- **EV$_n$ events** - events compromised of two scene features whose corresponding discontinuity surface intersects the light source.
- **EEE$_n$ events** - events involving three scene edges where the discontinuity surface intersects the light source.

From purely combinatorial point of view, the discontinuity meshing would be almost unusable approach. For a scene with n polygons, there are $O(n^2)$ possible EV events and $O(n^3)$ possible EEE events. Each discontinuity surface can interact with up to n polygons, yielding a possible $O(n^4)$ discontinuity curves, resulting in up to $O(n^8)$ mesh elements. As stated elsewhere [LTG92, Hed98], these upper bounds are too pessimistic in practice and our results also confirm that truth.

3 PREVIOUS RESEARCH ON ACCELERATING TECHNIQUES

A simple pseudocode for performing discontinuity meshing is outlined in figure 2, where combination of features can be comprised of either two or three items.

The most critical operation is the *surface propagation* through the scene. It includes calculation of intersections between the discontinuity surface and all faces in the scene. Various acceleration schemes are usually applied in order to speedup this task.

For EV$_n$ and EEE$_n$ events, there is another possibility how to speed up the computation. We should minimize the number of events tested that do not intersect

```
foreach combination of features {
    define a discontinuity surface formed by these features
    if (discontinuity surface misses the light source) continue with following combination[a]
    propagate the discontinuity surface into the environment
    solve the visibility within the discontinuity surface
    insert resulting discontinuity curves into elements
}
```

[a]this checking is not needed if one of the features is the part of the light source

Figure 2: Simple discontinuity meshing algorithm

the light source. In our implementation, we applied the approach presented by Drettakis. In [DF94], a bounding volume is created between the light source and a feature. Other features are then selected from within the bounding volume only. The bounding volume is usually described by a set of hyperplanes. We will call that kind of selection as the *constrained feature selection*.

At the beginning, we would like to note that some computer resources can be saved by making a careful choice of discontinuity surfaces selected for further processing. We can trace only those discontinuity surfaces whose features are in silhouette with respect to the light source. Tracing non-silhouette features would lead to discontinuity surfaces that point into an adjacent body. Thus, such events are immediately rejected because they do not influence visibility of other faces.

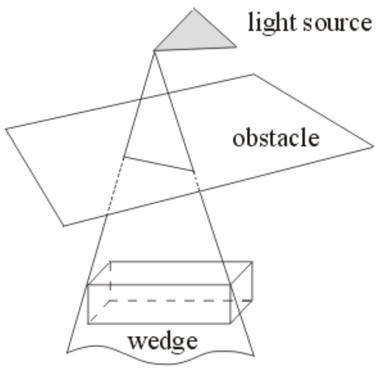

Figure 3: An example of scene where BSP-tree helps to finish wedge tracing early.

3.1 BSP-trees

Lischinski $et\ al.$[LTG92], and others [GH94, CS95] used BSP-tree to accelerate wedge tracing. A BSP-tree is built in a preprocessing step and a front-to-back traversal is performed to determine the ordering of polygons. Thus, the visibility is determined by a suitable tree traversal. Such approach has other nice property—the wedge tracing can be terminated as soon as the whole discontinuity surface is processed, see figure 3.

On the other hand, BSP-tree enforces polygon splitting[1] and the grow of the number of polygons can be impractical for large scenes. Building a minimal and/or well-balanced BSP-tree is not a simple task either. Furthermore, there is no obvious assistance for constrained feature selection and EEE-events tracing.

3.2 Voxels

Drettakis and Fiume [DF94] used a voxel grid to limit the number of processed polygons. When an EV-event is cast, the wedge is scan converted into the grid and only the polygons comprised in processed voxels are tested for intersection. When tracing an EEE-event, the intersection of quadric surface with the scene bounding box is found. The resulting quadratic curves are used to form bounding boxes on the scene boundaries that are then used to form a bounding box of the quadric surface. Within this box, only the objects contained in voxels cut by the quadric surface are examined.

When generating non-emitter events, from all combinations of features only those intersecting the light source are of interest. For EV$_n$ events, at each vertex v in the scene a pyramid is formed with v as its apex and the light source polygon as its base. The pyramid is also extended to the other side of the vertex. Faces that intersect the pyramid are added to the candidate list. The list is then traversed and every edge is tested to find out if the EV wedge formed of these features cuts the light source or not. If an intersection occurs, the extent of the event is possible adjusted and the discontinuity surface is cast into the environment. Processing of EEE$_n$ events is performed a similar way to EV$_n$ events processing, only the pyramid is not extended behind the generating edge, see the figure 4.

[1]Splitting can be avoided but special care must be taken during a traversal.

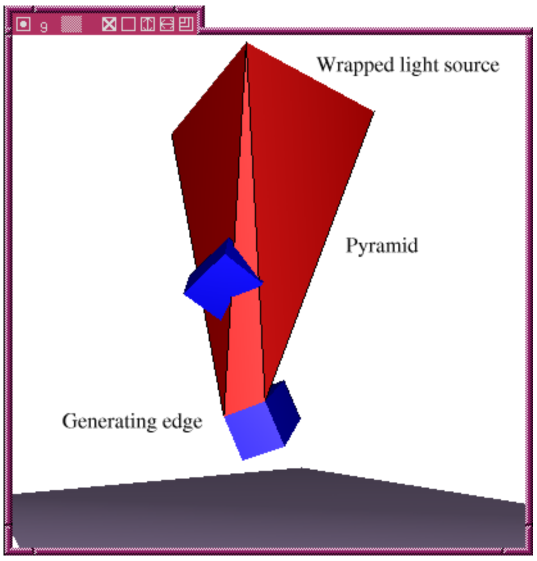

Figure 4: Bounded volume defined by an edge e and the light source

3.3 Octrees and Bounding Box Hierarchies

Octrees and bounding box hierarchies were utilized for surface propagation by Hedley [Hed98]. They have similar properties like voxels. However, they are adaptable to various local densities within a scene. Hedley presented comparison of acceleration techniques for planar surfaces that do not require the features selection step. For that case, octrees and bounding box hierarchies give similar performance.

4 NEW EXAMINED ACCELERATION STRATEGIES

In general, during the surface propagation we look for intersections of a 2D object with the scene. It resembles other computer graphics areas, such as ray tracing or collision detection where we look for intersections between scene and 1D or 3D object respectively. During a constrained feature selection, we look for intersections between a 3D volume and the scene.

In the past years, new acceleration strategies emerged in the field of collision detection—namely oriented bounding boxes and discrete orientation polytopes. Thus, we decided to prove their applicability for discontinuity meshing too.

4.1 Oriented Bounding Boxes

Oriented bounding boxes can be arbitrary oriented in 3D space. They are usually described by a centre point C, three orthonormal axes A_i and three extents e_i, for $i = 1..3$. Oriented bounding boxes were presented by Gottschalk $et\ al.$ [GLM96] and an extended discussion concerning these bounding volumes can be found e.g. in [Got98].

In collision detection systems, the coordinate system to which axes A_i are related is usually the parents node coordinate system. The reason is that the scene can be dynamic and some parts of the hierarchy are allowed to move. On the contrary, in discontinuity meshing systems the scene is static. Thus, the axes A_i can be specified in a global coordinate system.

4.2 Discrete Orientation Polytopes (k-DOPs)

k-DOPs are convex polytopes whose facets are determined by halfspaces whose outward normals come from a small *fixed* set of k orientations [KHM$^+$98, Zac98].

In order to keep the cost as low as possible, k orientations are usually selected from pairs of collinear, but oppositely oriented, vectors. Kay and Kajiya referred to such pair as bounding slabs [KK86]. Choosing such orientations has the advantage that the test for intersection between two k-DOPs is trivial: it consists of $k/2$ interval overlap tests.

k-DOPs are composed into a bounding volume hierarchy. Klosowski $et\ al.$ presented a variety of options that can adjust the hierarchy. However, these aspects are outside of scope of this work and can be found in the original paper [KHM$^+$98].

Axis aligned bounding boxes are a special case of 6-DOPs, with orientation vectors determined by the positive and negative coordinate axes. In our framework, we tested 6-DOPs, 14-DOPs, 18-DOPs and 26-DOPs. An alternative name for this category of bounding volumes is *fixed directions hulls*.

5 IMPLEMENTATION DETAILS

To compare the efficiency of k-DOPs and OBBs to other techniques, we implemented several previously mentioned acceleration methods. Our algorithm mimics the approach presented by Drettakis[Dre94]. We also compute EEE-events, non-emitter EV and non-emitter EEE events.

5.1 Basic Geometry Tests

In order to keep the implementation and tests unified, we decided to base the geometry calculations on a simple test. For EV events, the planar wedges can be described by three planes. For constrained features selection, the space is bounded by a set of planes. In addition, for an EEE event, the quadric surface can be bounded be a set of planes. Thus, the basic operation we implemented is the object-plane intersection.

The planes bounding an EEE event we determine in the following way. Initially, we compute the intersection of the quadric surface with the scene boundary and with the light source. Then we determine bounding boxes of resulting curves and finally connect the bounding boxes into a shaft [HW94]. Planes forming the shaft are used to speedup the computations, see figure 5.

5.2 Voxels

Rasterization is a well-documented topic [Wat93] and we extended this process to 3D. Various mesh densities were manually selected for experiments. A scene was scaled the way that voxel boundaries fall on integer grid, which simplifies the representation and accelerates the computation. Although voxel grids are fine for wedge rasterization, selecting the voxels lying inside the constrained volume is a demanding task compared to other accelerating methods. Furthermore, determining optimal voxel grid density may be tricky.

5.3 Octree

To build an octree, we used a top-down approach described by Hedley[Hed98]. The octree is parameterized by maximal number of polygons allowed in a node and by a minimal node volume to prevent infinite refinement. Hedley proposed to store single object's ID inside a node of corresponding size, which is reasonable if we want to use the octree further e.g. for hierarchical radiosity. However, our experiences have shown that purely for discontinuity meshing it is favorable to copy object IDs into all leaves covered by that object.

When a discontinuity surface is propagated into the scene, a set of candidate polygons is obtained during the octree traversal. The set of selected polygons is then tested for intersection with the discontinuity surface. A pseudocode for getting a list of polygons (for VE events) is presented in figure 6. The code assumes that the wedge is represented by three planes—the plane of the wedge and two perpendicular planes constraining the extent of the wedge's plane.

The test in pseudocode is conservative, it can succeed in nodes that actually do not intersect the wedge. We tested also a non-conservative checking procedure [GH95] and compared its results. In terms of overall speed, the methods of conservative and non-conservative checking are comparable, but for its simplicity we prefer the conservative method just presented.

5.4 BSP-tree

There are many possibilities how to construct a BSP-tree. In our implementation, we chose the heuristics where the polygon with largest area is selected into the current splitting node. The implementation of this strategy is simple and generally gives sufficient results.

```
GetCandidates(Octree $node$, Wedge $w$, List $list$)
{
    if (not intersection($node, w.plane()$))
        return;
    if (inPositiveSpace($node, w.rightPlane()$))
        return;
    if (inPositiveSpace($node, w.leftPlane()$))
        return;

    AddFacesToList($node.faces(), list$);

    foreach $child$ from $node.children()$ {
        if (not $child.empty()$)
            GetCandidateFaces($child$, $w$, $list$);
    }
}
```

Figure 6: VE event: selecting polygons for further processing.

5.5 Bounding Volume Hierarchy of k-DOPs

Various strategies how to build an bounding volume hierarchy (BVH) are described in [KHM$^+$98]. We adopted the approach presented by authors as a "splatter" strategy. The BVH-tree is build in a top-bottom fashion, where at each node a splitting plane perpendicular to axis with greatest variance is chosen.

As presented above, the basic operation in our framework is an object–plane intersection test. Although straightforward for other basic objects, this test is not evident for k-DOPs, besides the $k = 6$ case.

For 14-DOPs, we utilized the approach presented in [ZK97]. This algorithm computes extremal points in a given direction. Selecting the plane normal as the direction of interest we can determine the k-DOP–plane relationship in the same way as for axis aligned boxes. The algorithm precomputes a small amount of data for a given direction that are valid for all stored k-DOPs. These data are used later during the tree traversal to quickly evaluate the extremal points.

For 18-DOPs and 26-DOPs, we tested another strategy. The wedge was wrapped into a DOP volume and then collision of two DOPs was checked. This approach was used also to find out the edges intersecting a pyramid formed from a feature and the light source. Instead of representing the pyramid as an intersection of halfspaces we used the volume occupied by a k-DOP. We remind that comparing two k-DOPs for intersection amounts to k-interval tests.

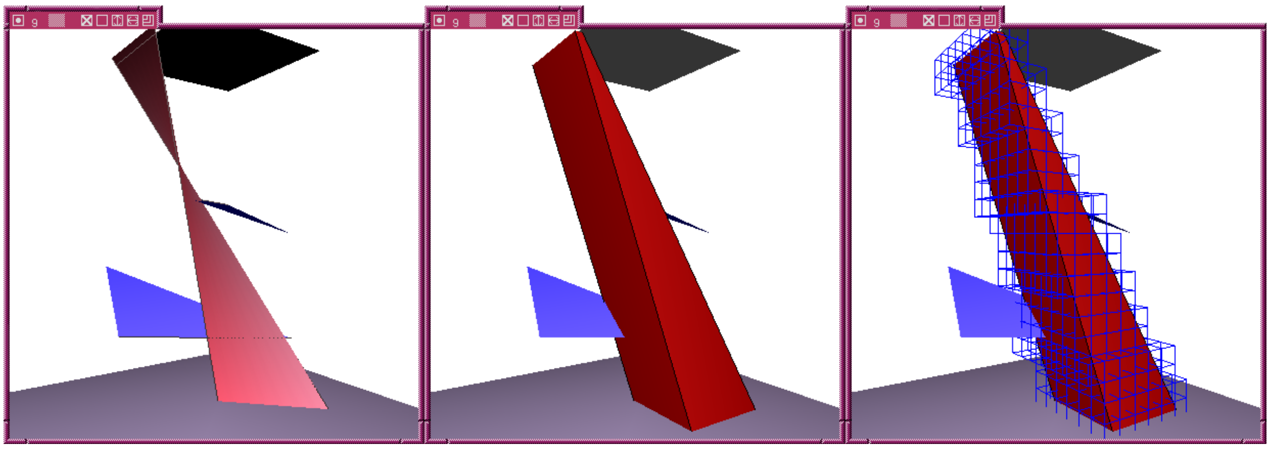

Figure 5: Bounding an EEE event into a shaft

Although not apparent at the first glance, depending on the set of directions selected for k-DOPs the interval test can give only conservative result. Even if the volumes do not intersect in reality the answer from the collision test can be positive. In these cases it is necessary to repeat the test on the lower levels in the tree.

6 TESTS AND RESULTS

For all experiments reported here, we used a Linux PC, (900 MHz, 1GB RAM). The code was compiled with GNU gcc compiler. All timings were obtained by adding the system and user times reported by the C library function ”times”. In order to smooth out minor variations in the timings, all tests have been run repeatedly and we report average times.

6.1 Test Scene and Its Preprocessing

For test purposes, we prepared a set of scenes. A simple interior scene was progressively ”polluted” with objects and mirrored in order to get more complex scenes. For the brevity, we present results from one scene only. The scene consists of 5216 input triangles and its arrangement can be seen on figure 7.

The memory requirements and preprocessing times to create various hierarchies are low. The time for creating accelerating structures is 0.5s for a BSP-tree and always less then 0.2s for other methods. Thus, the preprocessing time is negligible compared to overall discontinuity meshing computation. The memory requirements start at 160kB for larger voxels and end up with 1.7MB for an OBB tree. For exact values, see the table 1.

During the preprocessing step, we also tagged vertices and edges of the scene if they lied in a silhouette position to the light source. As edges and vertices are shared between multiple faces, we do not have to repeatedly evaluate silhouette information later.

Method	Memory [kB]
Voxels 20^3	161.4
Voxels 30^3	358.7
Voxels 50^3	1250.5
6-DOP	366.7
14-DOP	692.7
18-DOP	855.7
26-DOP	1181.6
Octree	1261.8
OBB tree	1711.3
BSP tree	771.4

Table 1: Memory requirements - scene with 5216 triangles

6.2 Casting VE and EV Events

The processing of VE and EV events represent the easiest part of all algorithms. The features are easily located and the resulting events are planar surfaces. The geometric configuration of faces in the test scene gives rise to 10080 VE events, which results in 8115 discontinuity curves. There are 34313 faces intersected by an VE event.

We get similar statistics for EV events. There were 11814 events cast, which resulted in 5600 discontinuity curves. The number of faces that collided with an EV event was 46054. These numbers are common for all algorithms tested.

However, the amount of faces that were examined for an intersection with an event is different for each particular acceleration technique. The results from tests are presented in table 2. The time that was actually needed to process the events is included in the table as well.

Method	**VE events**		**EV events**	
	Tested faces	time[s]	Tested faces	time[s]
Voxels 20^3	877221	1.2	1035076	1.7
Voxels 30^3	609026	1.1	705535	1.6
Voxels 50^3	426277	1.1	475979	2.0
6-DOPs	278848	0.9	260222	1.1
14-DOPs	230961	1.1	246941	1.5
18-DOPs	755163	1.7	531398	1.6
26-DOPs	540035	1.4	391862	1.5
Octree	402227	2.5	412869	2.1
OBB tree	217892	1.1	266703	1.5
BSP tree		3.4		3.6

Table 2: Results for planar events that do not insist on a features selection phase.

We remind that for 6-DOPs and 14-DOPs we compute intersection with a plane exactly, but in the case of 18-DOPs and 26-DOPs, we wrap an event into another DOP and a check for intersection between two DOPs. We do not present the number of tested polygons for a BSP-tree because the scene is subdivided into smaller polygons and the result is not comparable to other methods.

The timings from table 2 are very tight and we can not prefer any particular accelerating technique for planar surface processing.

6.3 Processing EEE, EV$_n$ and EEE$_n$ Events

All events discussed in this section require a features selection step. During the selection phase, a bounding volume is build and features for further processing are taken from that volume only. The faces that were caught in selection step are labelled as pyramid faces in the table 3. Then, events are cast into the scene and the number of faces tested for intersection is also stored in the results table.

In the current testbed, we compute EEE events in a combinatorial way, which is slower than the approach presented by Drettakis [DF94]. After finishing the mesh insertion part of the discontinuity meshing we can switch to optimized enumeration of EEE events too.

For non-emitter events, the results indicate that these events are manageable with current hardware too. We think it is because architectonic-like scenes show high horizontal complexity but vertical complexity is usually low. With vertical/horizontal complexity we mean how many objects can be found in a vertical/horizontal belt of the scene.

For EEE events, there were 28686 events cast which resulted in 8477 discontinuity curves. The number of faces that really intersected an event was 55213. For EV$_n$ events, there were 12596 events cast which resulted in 3789 discontinuity curves. There were 11633 faces colliding with an EV event. For EEE$_n$ events, there were 3210 events cast which resulted in 591 discontinuity curves. There were 2934 faces colliding with an EV event.

7 CONCLUSION AND FURTHER WORK

The BSP-tree and voxels are not particularly suitable for accelerating a complete discontinuity meshing algorithm. Both of these methods suffer when it comes to the features selection part of the algorithm. As voxels size goes smaller, the number of processed faces goes down. But on the other hand, the selection and event tracing phase become more demanding, bringing no advantage at all. A BSP-tree does not help us when processing non-planar events either.

The 6-DOPs (axis aligned bounding boxes) seem to be the most powerful acceleration technique. Even though they are not the tightest bounding volumes, they win in speed.

Unfortunately, our hope that examined k-DOPs and/or OBBs can bring a breakthrough in accelerating methods for discontinuity meshing was not fulfilled. For moderately sized scenes we tested, these accelerating methods give similar performance as the axis aligned bounding boxes. We got remarkable speedup against the axis aligned boxes solely for unconventional artificial scenes.

In the near future, we are going to test the accelerating techniques on large scenes. When developing the framework, the next step is to implement discontinuity curves insertion into the scene elements. As already noted, that should help us to enumerate EEE events in a faster way.

For further development, we would also like to switch the order of computations and to compute all events that arise in a common space progressively. E.g. we could cache the objects situated in a given region and then evaluate all events from that region regardless of the type of event. Such ordering could avoid unnecessary tree traversals and could help us to discard unimportant events in one step if we employ an importance-based generation of discontinuity curves.

ACKNOWLEDGEMENTS

The authors wish to thank to Petr Konečný for sharing his ideas and code. The work presented here was supported by Grant Agency of Czech Republic, Contract No. GACR 201/98/K041 and by Czech Ministry of Education, Contract No. VZ MSM143300003.

Method	**EEE events**			**EV$_n$ events**			**EEE$_n$ events**		
	Pyramid faces	Tested faces	Time [s]	Pyramid faces	Tested faces	time [s]	Pyramid faces	Tested faces	time [s]
Voxels 20^3	998730	3416760	38.9	331925	1042929	3.6	429145	425460	7.0
Voxels 30^3	673076	2427820	37.0	234125	730153	3.8	271929	318769	7.8
Voxels 50^3	487047	1693231	57.8	162983	526333	6.6	195252	218985	14.0
6-DOPs	333098	1265357	17.8	95905	391445	2.0	134873	172134	4.0
14-DOPs	322656	1254506	18.7	93600	346432	2.7	128365	170430	4.3
18-DOPs	637263	1991163	25.0	217185	570730	2.8	318457	264246	5.3
26-DOPs	429888	1335761	18.6	147339	438723	2.3	203308	188680	4.2
Octree	446641	1660396	22.6	440552	1313049	4.1	572431	590502	7.7
OBB tree	334879	1298364	19.8	98918	310617	2.3	130791	165108	4.3

Table 3: Results for events that require a features selection phase

.

REFERENCES

[CS95] Yiorgos Chrysanthou and Mel Slater. Shadow volume BSP trees for computation of shadows in dynamic scenes. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 45–50. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[DF94] George Drettakis and Eugene Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 223–230, 1994.

[Dre94] George Drettakis. Simplifying the Representation of Radiance from Multiple Emitters. In *Fifth Eurographics Workshop on Rendering*, pages 259–272, Darmstadt, Germany, 1994.

[GCS91] Ziv Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE PAMI*, 13(6):542–551, 1991.

[GH94] Neil Gatenby and W. T. Hewitt. Optimizing Discontinuity Meshing Radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 249–258, Darmstadt, Germany, June 1994.

[GH95] Daniel Green and Don Hatch. Fast polygon-cube intersection testing. In *Graphics Gems V*, pages 375–379. 1995.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[GM90] Ziv Gigus and Jitendra Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE PAMI, February 1990*, 12(2):113–122, 1990.

[Got98] S. Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis, University of North Carolina. Department of Computer Science, 1998.

[Hec92] P. Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[Hed98] David Hedley. *Discontinuity Meshing for Complex Environments*. PhD thesis, Department of Computer Science, University of Bristol, August 1998.

[HW94] Eric A. Haines and John R. Wallace. Shaft Culling for Efficient Ray-Traced Radiosity. In P. Brunet and F. W. Jansen, editors, *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, New York, NY, 1994. Springer-Verlag.

[KHM$^+$98] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In David C.

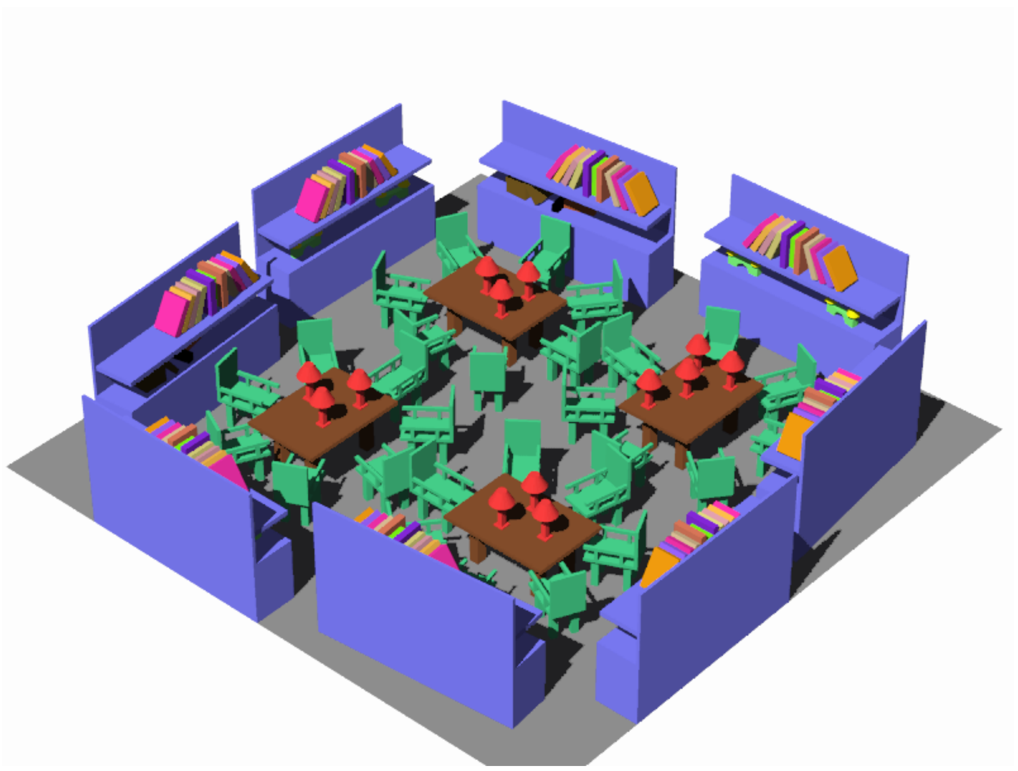

Figure 7: Test scene with 5216 triangles.

Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986.

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[Wat93] Alan Watt. *3D Computer Graphics*. Addison-Wesley, 1993. ISBN 0-201-63186-5.

[Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, March 1998.

[ZK97] Karel Zikan and Pert Konecny. Lower bound of distance in 3d. In *WSCG '97 (The Fifth International Conference in Central Europe on Computer Graphics and Visualization)*, pages 640–649, Plzeň-Bory, Czech Republic, 1997. University of West Bohemia.