Bounding Volume Construction using Point Clouds

Wolfgang Stürzlinger

GUP, Johannes Kepler University Linz
Altenbergerstraße 69, A-4040 Linz, Austria/Europe
Tel.: +43 732 2468 9228, Fax: +43 732 2468 9496
Internet: stuerzlinger@gup.uni-linz.ac.at

Abstract. Bounding volumes are used to optimize many graphic algorithms. They permit to decide quickly if a more exact test is likely to succeed or not. The exact test is only applied if the bounding volume test succeeds, which saves time in most cases. Two of the most prominent applications are ray tracing and collision detection.
This paper presents a new method to calculate tight bounding volumes for primitives under arbitrary transformations and hierarchical scene descriptions. An optimized form of storing a convex hull is presented which allows efficient calculation of the bounding volume type used for later processing. Results are shown for the ray tracing of cyclic CSG-graphs used to render plants and other natural phenomena.
Key Words: bounding volume, bounding volume hierarchy, convex hull, ray tracing.

1 Introduction

Bounding volumes are used to optimize and speed up many graphic algorithms. The basic idea is to apply a test first to a simple enclosing object before using a time consuming and exact test for the original object. If the simple test fails the complex test does not need to be performed as it will fail as well. This saves a significant amount of time in most cases. The most common enclosing object is the axes parallel bounding box. Examples of tests which can be sped up are: ray-object intersection and object-object intersection.

One of the most prominent applications of bounding volumes is ray tracing a well established method to produce photo-realistic pictures. Rays are used to determine visibility and to capture important illumination effects such as shadows and reflections by intersecting the rays with the objects of the scene and determining the foremost intersection. Up to ninety percent of the computation time is taken up by ray-object intersections. For a more complete introduction to ray tracing see e.g. the book by Glassner [Gl89]. Other applications include collision detection, animation, and camera frustrum culling.

This work introduces a new method to *calculate* tight bounding volumes for arbitrarily transformed and combined objects. In section 2 the criteria for good bounding volumes are reviewed. Then the new point cloud approach is introduced in section 3. As point clouds cannot be used directly for the tests after construction, the conversion to other bounding volume types is discussed as well. In section 4 the implementation and some results are presented.

2 Bounding Volumes

Testing an arbitrary complex object for inclusion in a volume or for intersection with a ray is generally a time consuming operation. However these operations are fast to perform for simple objects like axes parallel boxes. Enclosing a complex object by a simpler one allows for a quick test if there is a good possibility of the complex test to succeed. Likewise if a ray misses the volume of the enclosing box, it will certainly miss the original object. If the ray intersects the box, there is a high probability that the ray will hit the original object. In general the possibility of quick elimination from further tests rectifies the small amount of extra computations (see e.g. [Gl89]).

More commonly these enclosing objects are known as Bounding Volumes (BV's). Good bounding volumes are chosen according to the following criteria:

- Tightness: The BV should closely fit arbitrary objects to eliminate superfluous tests.
- Minimal storage consumption: To avoid excessive memory consumption, as each object needs an associated BV. This allows to use BV's for every objects.
- Fast methods for tests to keep the extra computations as small as possible. This (almost) implies the use of convex BV's as they are less computationally intensive to test.
- Easy to construct: As BV calculation is mostly done once in a preprocessing phase, this criterion is less significant.

Examples of tests where BV's are applied are:

- testing if a point is inside an object.
- testing an object for intersection with a line (ray).
- testing if an object intersects a plane or lies above/below.
- testing an object for intersection with and/or inclusion within a volume.

Previously used or discussed types of BV's include:

- Axes parallel bounding boxes (Bboxes) are the most commonly used form, because they are the easiest to generate.
- Bounding spheres are used because the intersection procedure is easy and fast.
- Other variants are: bounding slabs [KK86], arbitrarily transformed Bboxes, and parallel-epipeds [BS93].
- Convex Hulls: Have never been used due to the complexity of storing a convex hull and the nonexistence of quick intersection algorithms.

For a scene consisting of thousands of objects even testing all BV's is an expensive operation. A hierarchy of BVs allow even quicker elimination of objects from consideration. Assume that the scene has been defined hierarchically. After calculating the BV's for all objects stored in the leaves, the BV's for intermediate nodes are calculated as the union of the children's BV's which makes the root node a BV for the whole scene. This bounding volume hierarchy (BVH) is used in a top down fashion: If a ray does not intersect the BV of a node, it will miss all objects beneath that node and the test stops. Otherwise the test is applied recursively to the children of the node. When reaching a leaf it's BV is tested and if this test is successful the corresponding object is tested. Other types of optimization techniques (regular grid, octree, etc.) also use the BV of the objects to find which object lies in which region of space during the setup phase. For a discussion of these techniques and automatic BVH construction see [Gl89].

Trumbore [Tr92] discusses the generation of optimal Bboxes for several transformed quadrics and tori. For other objects under arbitrary transformations it is not easy to find an optimal BV. This holds especially for rotated and sheared objects. CSG operations such as intersection are complicated to handle as it is difficult to construct an optimal BV for the remaining object just from the two BV's of the child nodes. Scene hierarchies with embedded transformation nodes (as commonly used in animation) also pose problems as each transformation node can only work on the BV of it's successor.

The situation is even worse for cyclic CSG-graphs [GT96] where repeated transformations *and* combinations are applied to the primitives. The use of Bboxes can result in unnecessary enlargement due to repeated rotations (see figure 1a). The worst case for 2d-Bboxes is a rotation by 45 degrees, here the increase is 41% for each rotation. In three dimensions the increase can be up to 73%. Bounding spheres and arbitrarily aligned bounding slabs are rotation invariant, but cannot represent an optimal BV for a combination of two rotated instances of child nodes (see figure 1b,c). As bounding slabs cannot represent more than a limited set of plane orientations, they do not have enough degrees of freedom to represent the optimal result (see figure 1c for an example with six planes).

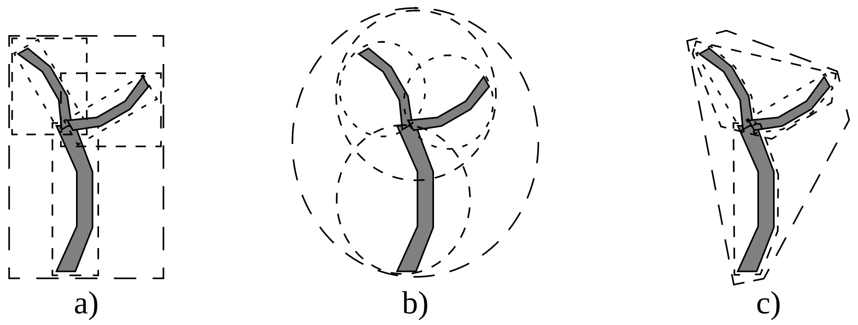

Fig. 1: Suboptimal bounding volumes for cyclic CSG-graphs.

All the above methods assume (implicitly) that the type of BV used during processing must be the same as used during setup time. However the BV representation during setup time does not depend on the type of BV used later on. During setup a better suited type of BV can be used, if it is converted to another form type before processing, which is easier or faster to test.

3 Point Clouds

A convex hull (CH) of an object represented by a convex polyhedron is an optimal bounding volume. For curved objects the CH can be chosen as tight as convenient by varying the number of points defining the hull. The CH is invariant to rotations. The union of two CH's increases the total volume minimally and can be performed by convex hull algorithms known from computational geometry. The intersection of two CH polyhedra can be used to calculate the CH for CSG intersection. The major disadvantage of the CH is the complexity of intersection algorithms. Another disadvantage is high storage requirement, as not only the vertices, but also the edges and faces along with topological information must be stored.

Using the CH during setup phase to construct the BV's and the BVH will provide an optimal result. To avoid the use of the CH during the processing phase, the CH's should be converted to more conventional BV's at the end of the setup phase, resulting in optimal BV's and BVH's. The CH's may then be discarded before the processing phase if they are not needed any more.

Still, constructing CH's is not an easy operation due to the necessary topological computations. Analysing the operations which are needed in the preprocessing phase shows that it is easier to use a list of points as the representation of the BV. The CH of this point list is than defined as the respective bounding volume. The operations introduced later on imply no ordering in the list. Also there is no need to remove points lying inside the CH of the point list immediately for the proposed algorithms. These properties led to the use of the name point clouds (PC's) for the newly introduced BV type.

3.1 Point Clouds for Primitives

For all convex polyhedra and polygons the point cloud (PC) can be generated trivially by taking the list of all vertices. Non-convex polyhedra can also be handled in exactly the same way, because PC's do not have to form a CH. Quadrics, quartics, superquadrics, etc. can be handled by computing the intersection points of appropriate tangent planes (depicted in figure 2). Increasing the number of tangent planes allows to improve the tightness of the convex hull. As all primitives are normally defined in an object coordinate system the points forming a convex hull can be precalculated and reused for all instances.

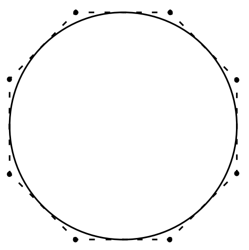

Fig. 2: Principle of point cloud construction for curved objects

Translational sweeps are handled by constructing a PC for the two dimensional contour and then duplicating the PC for the upper cap. For conical sweeps it suffices to add the top point. For rotational sweeps the two dimensional contour point list is scaled radially outwards and duplicated in appropriate rotation steps (e.g. scaled by a factor of $\sqrt{2}$ for 90 degree rotations). Free form surfaces such as Bezier, B-Spline and NURBS[1] surfaces obey the convex hull property and are handled by reusing the control point list.

3.2 Transforming Point Clouds

A PC can be transformed with an affine transformation matrix by appling the transformation to all its points (which maintains the CH property).

More general transformations such as bending, twisting, tapering, and other free form deformations are handled by constructing the CH of the PC and computing the new PC for the transformed CH. One way to perform this CH transformation is to split all faces of the CH into small enough pieces, and use the transformed vertices as the new PC.

3.3 Point Clouds for the Nodes of a Hierarchy

For hierarchical scene descriptions such as CSG-trees or BVH's the construction of a

1. The convex hull property holds for NURBS with positive weights only.

node's PC from the PC's of the node's children proceeds as follows:

Union (OR).

The most natural choice is to employ a CH algorithm [PS85][BDH93][Or94] to the combined list of the children's PC's and construct the new PC from the points of the CH. All inner points can be discarded. Due to the high computational demand of the CH algorithms this is a time consuming operation. A potential optimization is to use just one pass of the gift wrapping algorithm [PS85], if the two PC's do not intersect each other.

Instead of constructing the CH for the combined PC at each union node it is faster to just append the PC's of the children to form the new PC. But this allows the point lists to grow unnecessarily large. To prevent this the CH algorithm is used only if the number of points in the new PC exceeds a predefined threshold. This scheme provides a balance between the computation time of the CH algorithm and the processing time for the PC lists.

Intersection (AND).

To construct the PC of an intersection node, the convex hulls of the two children are computed. The list of vertices of the intersection polyhedron of the two CH polyhedra is then used to construct the new PC. One major disadvantage of this approach is the associated computational requirements, as two CH and a polyhedra intersection have to be computed.

In typical scenes intersection nodes are rare compared to the number of union nodes. Therefore, a faster but sub-optimal algorithm can be used. It computes the Bboxes for the two PC's of the children (see section 3.4), intersects the two Bboxes, and uses the eight vertices of the resulting Bbox to construct the new PC.

Difference (SUB).

Due to the properties of this node optimization is impractical and the resulting PC is identical to the PC of the first operand.

3.4 Constructing Bounding Volumes from Point Clouds

Constructing the axes parallel Bbox for a list of points is done by finding the minimum and maximum extent in the x-, y-, and z-directions respectively.

To compute a bounding sphere for a list of points several algorithms have been proposed [Ri90][Wu92]. The second algorithm employs eigenvectors to find the principal axes of the point distribution which yields optimal bounding spheres. It can also be used to find optimal arbitrarily oriented bounding boxes.

3.5 Constructing the Bounding Volume Hierarchy

Constructing the BV's for a BVH proceeds recursively through the hierarchy. First each node constructs the children's PC's. Then it applies the corresponding operator to the children's PC's and computes it's own PC (section 3.3) which is then used to construct and store the node's BV (section 3.4).

In the case of Bboxes as BV's a possible optimization for intersection and difference nodes performs another recursive pass over the children's Bboxes before returning to the calling procedure. This recursive procedure uses the parent's Bbox to minimize and update the children's Bbox by intersecting the two Bboxes. For a difference node the right sub-tree can be optimized (see figure 3b). If the intersection is empty the right sub-tree is redundant and can be optimized away (figure 3c).

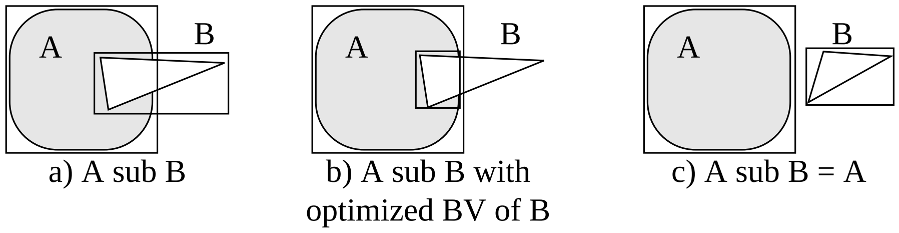

Fig. 3: Optimized bounding volume construction for difference nodes

In the case of an intersection node the Bboxes of both sub-trees can be optimized. If the Bbox intersection is empty the intersection node describes an empty volume and the whole sub-tree can be removed.

4 Implementation and Results

The new method was implemented in the FLIRT CSG ray tracing system [STS93], which was extended with an earlier version of the cyclic CSG-graph system [TG95]. The ray tracer uses Bboxes as BV's for primitives and a BVH for ray casting optimization. All tests were performed on an Indy R4600PC. The optimal threshold for the PC's maximum length (see section 3.3) was found to be 256 for the implementation used.

A non-contractive cyclic CSG-graph for a flower was used to show the efficiency of the new method. The extent of the uppermost Bbox of the scene was approximately [±31700.0; ±31300.0; ±8400.0] due to the repeatedly rotated Bboxes for the unoptimized implementation. The new method produced the following Bbox: [-4.3,5.0; -3.0,4.9; 0.0,12.9] which is smaller by several orders of magnitude and is a very tight enclosure. The memory consumption and timings for a 512 by 512 picture (see figure 4) are given in the following table:

Table 1: Statistics for flower cyclic CSG-graph

	setup time (sec)	rendering time (sec)	max. memory usage (kB)
without PC	1	7355	226
with PC	115	1499	362

Above results show that the new BV construction method leads to a performance increase by almost a factor of five for a moderate increase in storage use. A recent comparison with the optimized BV construction method described by [TG95] which uses a stack of Bboxes and transformations showed that the method presented here constructs a slightly smaller Bbox (1.5%) due to the fact that Bboxes are not optimal BV's for all primitives (e.g. spheres). The setup time for the method of [TG95] was seven seconds; the rendering times for the two methods were equal.

Additionally the implementation of the cyclic CSG-graph system was improved by the introduction of a new node consisting of a combination of calculation (C) and multiple transformation (T) nodes which was called CT-node. This allows to shorten the grammar

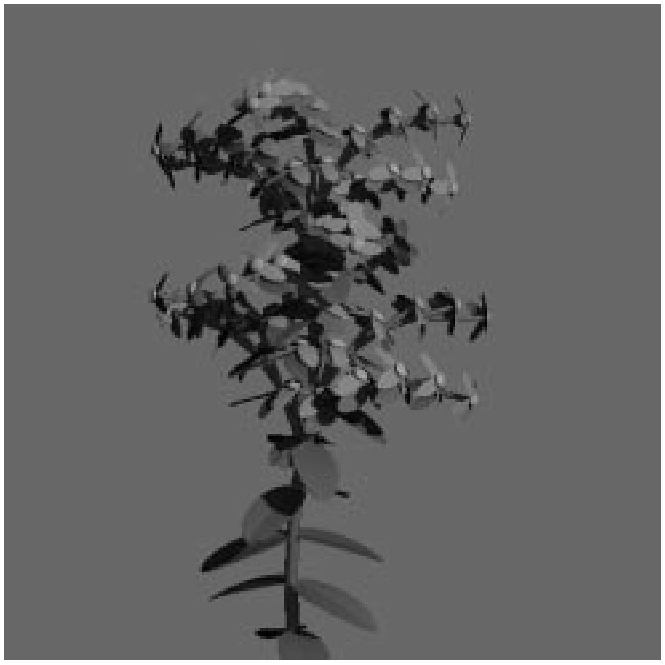

Fig. 4: Example image

underlying the cyclic CSG-graph and saves unnecessary computational overhead.

Table 2: Statistics for improved flower cyclic CSG-graph

	setup time (sec)	rendering time (sec)	max. memory usage (kB)
without PC	1	4166	164
with PC	75	940	283

5 Conclusion and Further Extensions

A new method to construct BV’s has been presented. The construction of point clouds (PC) for primitives, transformation nodes, and combination nodes was discussed. The used data structures facilitate the construction of tight Bboxes (or other BV types) and bounding volume hierarchies for the processing phase. The given example shows that even for repeated transformations optimal BV’s can be constructed resulting in major time savings in a ray tracing application. A threshold used to limit the growth of the PC’s can be used to tune relative performance of the PC construction and the CH algorithm.

Other possible applications of this new method include the optimization of collision detection algorithms [Ra94] and the use in animation systems.

6 References

[BDH93] C. Barber, D. Dobkin, H. Huhdanpaa, *“The Quickhull Algorithm for Convex Hull”*, Geometry Center Technical Report GCG53, Univ. of Minnesota, MN, 1993.

[BS93] W. Barth, W. Stürzlinger, *“Efficient Ray Tracing for Bezier and B-Spline Surfaces”*, Computers & Graphics, vol. 17, no. 4, pp. 423-430, July 1993.

[Gl89] A. Glassner, *“An Introduction to Ray Tracing”*, Academic Press, San Diego, CA, 1989.

[GT96] M. Gervautz, C. Traxler, *“Representation and Realistic Rendering of Natural Phenomena with Cyclic CSG-Graphs”*, to appear in Visual Computer.

[KK86] T. Kay, J. Kajiya, *“Ray Tracing complex scenes”*, Computer Graphics (SIGGRAPH 86), vol. 20, no. 4, pp. 269-278, 1986.

[Or94] J. O’Rourke, *“Computational Geometry in C”*, Cambridge University Press, Cambridge, UK, 1994.

[PS85] F. Preparata, M. Shamos, *“Computational Geometry: An Introduction”*, Springer Verlag, New York, 1985.

[Ra94] R. Rabbitz, *“Fast Collision detection of Moving Convex Polyhedra”*, in Graphics Gems IV, edited by P. Heckbert, Academic Press, pp. 83-109, 1994.

[Ri90] J. Ritter, *“An Efficient Bounding Sphere”*, in Graphics Gems, edited by A. Glassner, Academic Press, pp. 301-303, 1990.

[STS93] W. Stürzlinger, R. Tobler, M. Schindler, *“FLIRT - Faster than Light Ray Tracer”*, Technical Report, Institut für Computergraphik, Technical University of Vienna, Aug. 1993.

[Tr92] B. Trumbore, *“Rectangular Bounding Volumes for Popular Primitives”*, in Graphics Gems III, edited by D. Kirk, Academic Press, pp. 295-300, 1992.

[TG95] C. Traxler, M. Gervautz, *“Calculation of Tight Bounding Volumes for Cyclic CSG-Graphs”*, Proceedings of Spring Conference on Computer Graphics (Bratislava, SV), pp. AP40-49, 1995.

[Wu92] X. Wu, *“A Linear-Time Simple Bounding Volume Algorithm”*, in Graphics Gems III, edited by D. Kirk, Academic Press, pp. 301-306, 1992.