Bounding Volume Hierarchy Analysis (Case Study)

by

Radek Oslejšek

FI MU Report Series FIMU-RS-2000-13

Bounding Volume Hierarchy Analysis (Case Study)

Radek Oslejsek
oslejsek@fi.muni.cz

Faculty of Informatics
Masaryk University
Brno, Czech Republic

Abstract

Bounding Volume Hierarchies are very popular structures for objects storage of virtual scenes. In this article various OO models of BVH are analysed and described using UML. The first part compares structure of bounding volume hierarchy with inner structure of virtual scene representation. The second part is focused on relationship between a class hierarchy of scene graph and a class hierarchy of BVH.

Analysis of bounding volume hierarchies is a part of a major project dealing with analysis of computer graphics architectures. Ideas originate from experimental architecture under development. Case study shows how the current OO technology and software engineering may bring distinct architectural and design concepts into computer graphics.

Keywords: Bounding volume hierarchy, scene graph, rendering, UML

1 Introduction

An applications of computer graphics often works with wide scenes comprised by a big amount of primitives. For an efficient access to each primitive it is necessary to have stored them in any kind of structure providing quick search. This is required not only by local illumination computation or collision detection but mainly by global illumination methods because during a light distribution we have actually to probe the whole scene for every individual primitive again and again. In the computer graphics structures for object storage are usually called *Space Sorting Structures*.

The most popular and the most often used is just the *Bounding Volume Hierarchy - BVH*. Bounding volumes should help with effective scene search, that is it should optimize object selection based on given strategy. To do that it should cooperate closely with scene graph.

Scene Graph - SG is hierarchical structure representing a virtual scene. It is a tree which nodes are drawable objects (geometric primitives, fog, etc.), special nodes like "link to sub-tree" or "sound" and finally there are group nodes. The last type creates tree ramification. It stores information about its sons and also offers their maintenance and inspection. Typical specialization of **Group** node is **Transform** node which manage geometric transformations of all sons towards upper objects.

Now we have defined basic terms. To be able to describe relationships between bounding volumes and scene graph we summarize requirements for their properties first.

The first requirement to scene objects proceeds from description above: There is a possibility to store them hierarchically using scene graph. Together with geometric transformations hierarchical order allows an easy definition of complex scene with conditions like "a chair is rotated and translated towards a table" and also it facilitates quick look up.

The second requirement to objects is an ability to draw them. Therefore they have to provide informations about their geometry (form) and illumination features (color, texture, transparency, etc.)

The third necessary functionality is intersection computation. On the one hand we need to be able to compute intersection of beam with object (used by ray-tracing for instance), on the other we need to compute intersection with any object in scene during collision detection. Even so in this article both types are called by the single term *intersection*.

Last required feature was already mentioned when we have talked about hierarchical representation of scene. Such a feature is geometric transformation. It is necessary to distinguish between two types of such transformations. The first one may be named "static transformation" and it is used only for a definition of static scene (see example with chair and table above). Then its main property is that objects can be rotated and translated to each other when the scene is created but this transformation is fixed during the scene manipulation. Expressed the same in different way, transformations are fixed inside the scene and they are not available from outside. Second type is “dynamic transformation” which allow sub-scene rotation anytime. They are available from outside the scene and therefore this solution brings a lot of problems with interface design. But such dynamic scene changes are not significant to BVs (see ideas about transformations inside BVH bellow). Therefore they are ignored here.

When we look at the properties required for bounding volume then we can see that they are very similar to properties required for objects of scene. BVH is a tree like scene graph is. Also we would like to draw particular bounding volumes. That can be used for approximation of objects far enough from an avatar. In such case we can replace a big amount of triangles forming object surface by a few volumes from selected level of tree composed by only a few polygons. Also the third functionality is similar to scene objects. BVH is used for quick search of candidates to point of intersection. Therefore it should support computation of intersection with light beam. For a collision detection it also should provide computation of intersection with another bounding volume. BV must be also transformable together with packed objects.

2 BVH Structure Analysis

This chapter is focused on two variants of hierarchical representation of bounding volumes in comparison with scene graph definition. Both discussed solutions differs in a way how a BVH and a scene graph are constructed and how they are managed.

2.1 Self-Managed Bounding Volume Hierarchy

The first solution defines BVH as a structure independent on scene graph. The class **BoundingVolume** depicted in Figure 1(a) contain pointers to sons or in the case of leaf node it contain pointer to list of stored objects. Let me note that object stored in BVH may by any object of scene graph not only a primitive.

Because this arrangement implies that the structure of BVH is under its full control, it is called *Self-Managed Bounding Volume Hierarchy*, shortly *Self-managed Hierarchy* or *Self-Managed BVH*.

This arrangement has three advantages:

1. BVH directs its own structure. BVH decides where a new object of scene is stored or haw a tree depth is managed (for in-

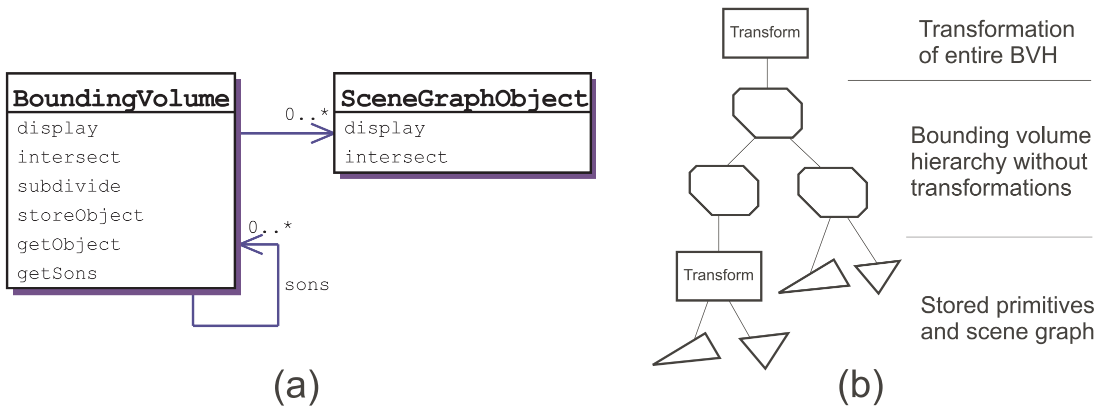

Figure 1: Class diagram of self-managed hierarchy (a) and an example with geometric transformations (b)

stance maximal amount of objects in leaves versus maximal tree depth).

2. Ability to optimize computation with other bounding volume hierarchies. Because BVH know its own structure then it is able to traversing it efficiently and in some cases it is able to cooperate with another BVH which also know its inner structure.

3. Inner nodes of BVH have only a simple structure without geometric transformations and usually they have only a fixed number of sons. Then it is possible to pass through the tree from root to leaf very quickly and then to speed up search of interesting objects.

Absence of transformations inside BVH discussed in third point is not contradict to requirement of transformation of BVH described in previous section. What we really want is to be able to transform stored objects individually - that is their inner functionality, or to be able to transform all stored objects together - that is transformation of the whole tree which is not protected (see Figure 1(b)).

Usage of self-managed model has also some disadvantages:

1. Hierarchy without transformations inside and objects stored only in leaves deplores us to pass through entire tree with no benefit until leaves. Only in leaves is stored any useful information like candidates to intersection.

2. It is possible to reasonably work only with one type of bounding volume in one hierarchy because it is rather hard to propose

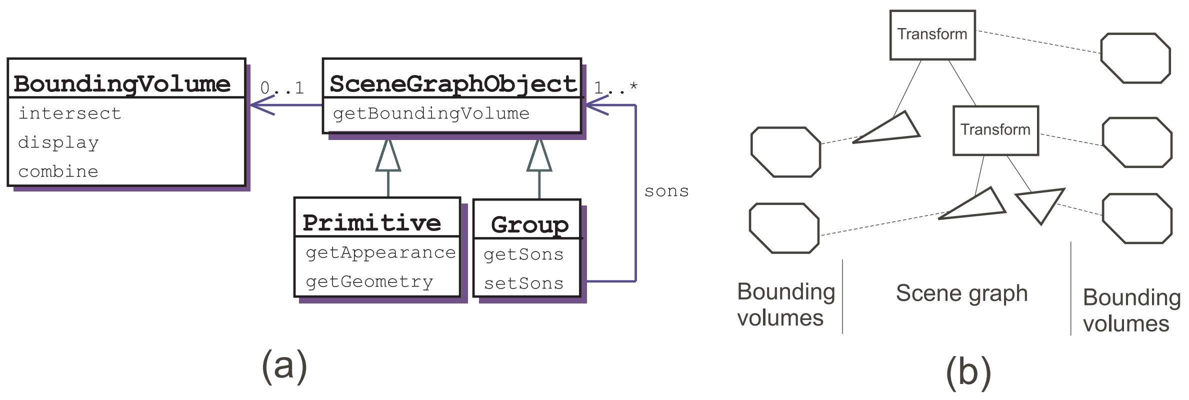

Figure 2: Class diagram of outside-managed hierarchy (a) and an example of scene graph (b)

enough general OO model of hierarchy combining for example box and sphere as its BV. The trick how to reduce such restriction is to store in leaves another BVH instead of only object of scene. But this is quite brute adaptation.

Operations over BVH can be implemented in two ways. Firstly they can be fixed in interface. But this solution brings some problems when we want to append new operation into existing class hierarchy. The second way is to append general methods for tree traversing into the interface and to use them for children inspection. More detailed discussion is in section 3.1.

Conclusion is that a structure of self-managed model is mainly useful in situation when we want to store mutually static objects like any complex solid approximated by a big amount of planar polygons. In such case there is a free space for various optimizations and speedups. But we can use discussed arrangement also as a repository of entire scene graphs or a parts of scene graphs managing transformations internally and therefore we are not restricted to only static scenes without transformations. Moreover, the next way haw to add geometric transformation to self-managed hierarchy is to rotate and translate BVH as a whole.

2.2 Outside-Managed Bounding Volume Hierarchy

The second view to bounding volume hierarchy definition is based on the fact that both the scene graph and the BVH are trees. The basic idea is to use structure of scene graph also as a structure of bounding volume hierarchy. Every node of scene graph contain a pointer to bounding volume packing itself (and all its sons), see Figure 2(a) and (b). Hence now we have to understand the term *Bounding Volume* as a really single BV rather then entire bounding volume hierarchy like in self-managed model. Therefore in the following text it is necessary to distinguish between the shortcut *BV* denoting single volume and the shortcut *BVH* used for entire hierarchy.

Contrary to self-managed BVH, hierarchy defined by this second way is completely dependent on the structure of scene graph and is also managed together with scene graph management. This is the reason why it is called *outside-managed BVH* .

Using outside-managed hierarchy has advantage that there is not necessary to define separate structure and methods for its administration. That may save a lot of work and it also brings simple unified interface usable for both structures. This interface only offers methods for tree structure management (setup of children) and methods for its traversing. All other operations like collision detection are applied externally. Finally it is possible to use different volumes for different nodes.

On the other hand, the tight interconnection between scene graph and bounding volumes, the structure of BVH managed outside and finally the unified interface for structure traversing are very restrictive for optimizations of methods working with volumes. Additional disadvantage is that using this model brings troubles with usage of another space sorting structures like *Unified Space Sorting* because almost of such non tree structures cannot be associated with single scene graph node in this tree context.

3 Structural Models of BV Definition

In previous chapter there was described two variants of coexistence between bounding volume hierarchies and scene graphs from point of view of their management. Contrary, following paragraphs are fo-

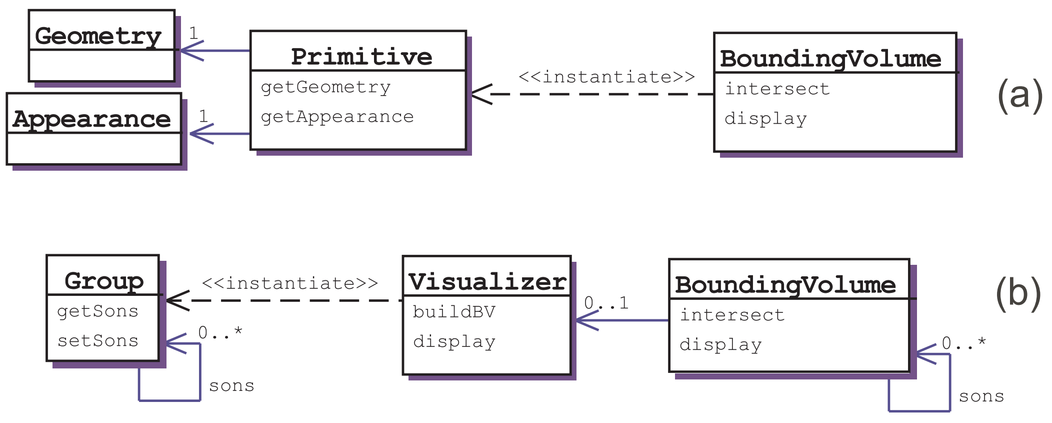

Figure 3: Bounding volume hierarchy defined separately

cused on possible bounding volume definitions in comparison with class diagram of scene graph.

3.1 Independent Definition of BVH

In the first case, the class hierarchy of bounding volumes and the class hierarchy of scene graph objects are defined completely independently. Using such a solution bounding volumes may be seen as a something special from visible objects of virtual scene. Event so, bounding volumes still have to provide two methods common for both the BV and the drawable object. That is visualization and intersection computation. Let look at them more closely.

The main meaning of bounding volume is to approximate any generally complex object by surface with easy mathematical manipulation. Therefore the bounding volume should have its geometrical properties prepared for effective computation of intersection with beam and with other volumes. Hence it should be easy to implement such methods.

A little bit complicated is visualization because BV can be rendered using various shading techniques from Phong's shading through wire-frame model up to any kind of global illumination. But every such technique requires different description of material and geometry (for instance radiosity require polygonal approximation for its job). Therefore there must be used quite complex model denoting illumination and geometry properties for every BV. But the same complex model must be created also for depictable objects of

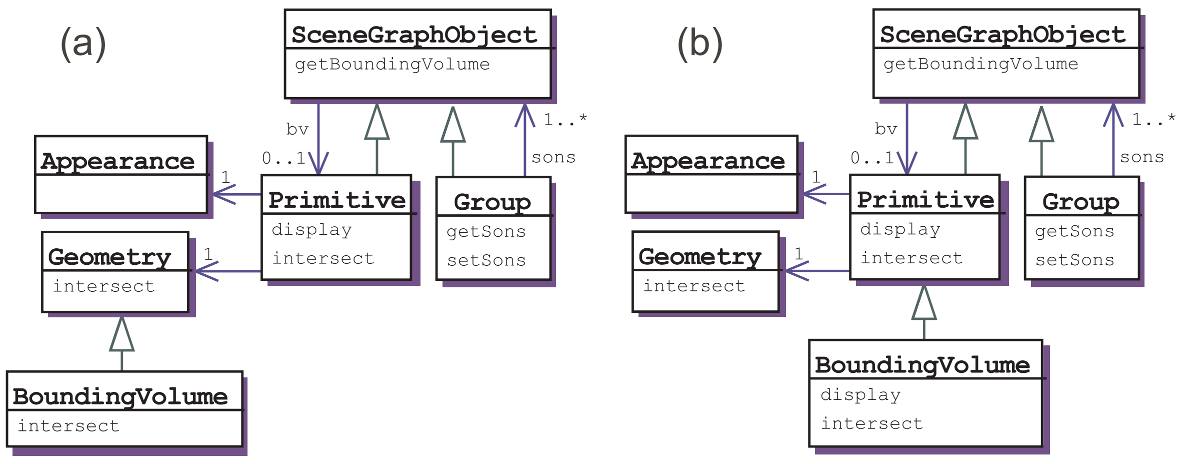

Figure 4: Bounding volume defined as a special geometry and a special primitive respectively

scene. Then it is rational to define geometry and appearance only once and to use it simultaneously by both classes. The most straightforward seems to associate the light properties with **Primitive** and in the **BoundingVolume** class to redefine method *display()* in the way that it instantiates colored primitives - see Figure 3(a).

Described approach assumes existence of a single primitive with geometry suitable for bounding volume. To get around this restriction the method *display()* can instantiate any objects of scene graph i.e. **Group** rather than primitive only. This solution allows creation of complex geometry composed by more primitives. But such processing may be quite complex. To became this solution a little bit clear and flexible it seems to be better to move visualization from single method *display()* into the special class **Visualizer** which encapsulates this functionality (Figure 3(b)).

Separate definition is applicable for single volume used by outside-managed hierarchy as well as for BVH used in self-managed model.

3.2 BVH as a Special Scene Object

Next three OO models works with BV like with object of scene graph having some special features. Therefore all three models defines bounding volumes as a specialization of various classes from scene graph.

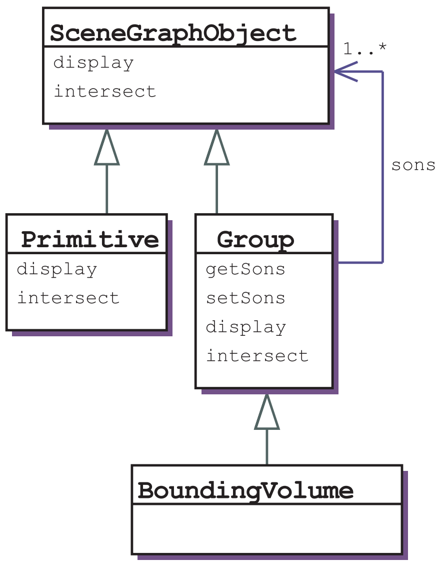

Figure 5: BVH as a specialization of **Group** node

The first two possibilities depicted in Figure 4 have very similar class diagrams. One of them looks at a bounding volume as at a special geometry. The second one uses BV as a special primitive. Both the principles are exactly the same. The only difference is that the scene designer instantiates appropriate bounding volume directly or he must instantiate some general primitive first and then to assign valid geometry to it..

It is easy to imagine the usage of described models for definition of single volume in outside-managed hierarchy. But because the classes **Geometry** and **Primitive** are understood as a simple non-separable objects it is wasteful to define entire BVH in such a way.

The third approach is based on the fact that both the bounding volume hierarchy and the **Group** node are mainly used for object storage. Accordingly a BVH is defined as specialization of just the **Group** class (Figure 5).

It is folly to define single volume inherited from class representing container and then this approach is not suitable for outside-managed model.

To be able to exploit power of entire BVH defined by this approach we must use existing methods for scene traversing in a little different way or we must fix all operations over BVH into its interface. The reason for that is based of the fact that standard methods for scene graph traversing works directly with stored sub-nodes (*get-*

Sons(), setSons()) but the inner structure of class keeps hidden. If we want to exploit knowledge of inner structure from outside then we have to adapt each operation onto the actual structure or we must be able to work with each inner nodes of BVH.

For instance, let assume that we want find candidates to intersection with ray. Using only the *getSons()* operation working in the standard way we must test all children sequentially. But when we fix the function *getCandidates()* into the interface then the function can return the most probable candidates first. Another already depicted way is to use existing traversing methods not for look up of stored children but for traversing of BVH inner structure. It means that the method *getSons()* returns children nodes in bounding volume hierarchy instead of really stored objects of scene.

Fixed operations are essential mainly in cases when also other types of space sorting may be used because not all space sorting structures are tree-based. But this approach have one disadvantage. When it is necessary to append some new operation into existing class hierarchy then all its classes must be properly changed. However the solution is to use ITERATOR pattern described in [GoF95]. More about several patterns may be also found in [Alex77], [Folw97], [Niem81], [Risi98], etc.

4 Conclusion

In previous chapters there was discussed two basic views on relationship between bounding volume hierarchy and scene graph. The first view have described two variants of BVH definitions: by separate structure independent of scene graph and secondly as a part of scene graph. The second part was focused on definition of bounding volumes in comparison with class hierarchy of scene graph.

Various combinations of models taken from both views are applicable to various tasks. Suitability of their usage strongly depends on requirements to scene manipulation and rendering techniques. The main goal of this article was not to explicitly name which combination are best for any concrete application. There was only mentioned restrictions and advantages of some combinations found during implementation of experimental architecture.

References

[Alex77] Alexanderi Ch., Ishikawa S., Silverstein M., Jacobson M., Fiksdahl-King I., Angel S.: A Pattern Language. Oxford University Press, New York, 1977.

[Folw97] Fowler M.: Analysis patterns reusable object models. Addison-Wesley, Menlo Park, 1997.

[Lea97] Lea D.: Concurrent programming in JAVA : design principles and patterns. Addison Wesley, Reading, 1997.

[Malv97] Malveau R.C., Mowbray T.J.: Corba design patterns. John Wiley & Sons, New York, 1997.

[GoF95] Gamma E., Helm R., Johnson R., Vlissides J.: Design patterns elements of reusable object-oriented software. Addison-Wesley, Reading, 1995.

[Coad97] Coad P.: Object models : strategies, patterns and applications. Yourdon Press, Upper Saddle River, 1997.

[Niem81] Niemann H.: Pattern Analysis. Springer-Verlag, Berlin, 1981.

[Vlis98] Vlissides J.: Pattern hatching : design pattens applied. Addison-Wesley, Reading, 1998.

[Buch96] Buschmann F. [et al.]: Pattern-oriented software architecture : a system of patterns. John Wiley & Sons, Chichester, 1996.

[Risi98] Rising L.: Patterns handbook : techniques, strategies, and applications. Cambridge University Press, Cambridge, 1998.

[Zika97] K. Zikan a P. Konecny, Lower Bound of Distance in 3D, Technical Report of FI MU, FIMU-RS-97-01, 1997.

[Sowi97] Sowizral H., Rushforth K., Deering M.: The Java 3D API Specification. Addison-Wesley, 1997.

[Ames97] Ames A.L., Nadeau D.R., Moreland J.L.: VRML 2.0 Sourcebook. John Wiley & Sons, 1997.

[Brow98] Brown W.J.: AntiPattern : refactoring software, architectures, and projects in crisis. John Wiley & Sons, 1998.

[Chen95] Chen S.E., Turkowski K., Turner D.: An Object–Oriented Testbed for Global Illumination. In: Laffra et al. (Eds.) Object–Oriented Programming for Graphics. Springer–Verlag, 1995, pp.155–166

[Slus95] Slussalek,P., Seidel,H.P.: Vision - An Architecture for Global Illumination Calculations. In: IEEE Trans. *Visualization & Computer Graphics* 1(1), 1995

[Fell95] Fellner,D.W.: MRT - An Extensible Platform for 3D Image Synthesis. Computer Graphics Lab., Dept. of Computer Science, University of Bonn, Germany, Dec. 1995

[Fell96] Fellner,D.W.: Extensible Image Synthesis. In: *Object-Oriented and Mixed Programming Paradigms*, Wisskirchen P., (Ed.), Focus on Computer Graphics, Springer, Feb. 1996

[Schr96] Schroeder W., Martin K., Lorensen B.: The Visualization Toolkit : An Object-Oriented Approach to 3D Graphics, Prentice-Hall, 1996.