VIS–RT: A VISUALIZATION SYSTEM FOR RT SPATIAL DATA STRUCTURES

Vlastimil Havran **Libor Dachs** **Jiří Žára**

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University, Karlovo nám. 13
121 35 Prague, Czech Republic
{havran,xdachs,zara}@fel.cvut.cz

ABSTRACT

In this application paper we describe a system for visualization of spatial data structures, that are used to accelerate ray shooting in global illumination. The system has been implemented and further used to verify the correctness of spatial data structures implementation, since for large and complex scenes the verification is difficult or even impossible.

Keywords: ray shooting, visualization, spatial data structures.

1 INTRODUCTION

Many image synthesis algorithms use ray shooting as sampling method of three–dimensional space to obtain geometrical properties of the scene. The visibility determination is essential particularly for modern photorealistic methods including stochastic ones: Monte Carlo radiosity, Instant Radiosity, Metropolis Light Transport etc. The time devoted to ray shooting can present a significant portion of total rendering time, since a large amount of rays is needed to synthesize an image, particularly for complex scenes.

Ray shooting problem is defined as follows: given a ray find its nearest intersection with objects in the scene. The trivial solution would test every object for intersection with the ray in $\Theta(n)$ time, where n is the number of objects. Since this trivial algorithm is unacceptable for any application, many methods based on some type of coherence reducing this time complexity have been developed. This includes mostly *ray* and *spatial coherence* [Arvo89]. At the present, it has not been proven that any of this method is of prevalent performance. The ultimate goal of research in this area is to find out a method with best performance and acceptable space complexity, and prove this fact both theoretically and experimentally.

When implementing the existing methods we found out that verifying the correctness of the implementation in accordance with required state is a difficult problem. Since new methods are developed particularly for large number of primitives, the resulting data structures are very huge. The implementation of any method has to process any set of objects in the scene and return correct result. If implementation is incorrect, it can result in several fault types:

- *time complexity is worse* than expected – difficult to detect, since comparison with a reference solution is even impossible, or there is no reference implementation available.
- *incorrect results* are returned – usually easier to detect, but the fault can appear on a particular scene only, that is usually unpredictable.
- *memory complexity is worse* than expected one – sometimes difficult to detect.
- any combination of points above.

All these reasons have led us to a development of a new system for visualization of spatial data data structures for ray shooting. The system enables us to visualize a spatial data structure regardless its type in a uniform way. We are not aware of existence of any similar system developed for such a purpose, that is main contribution of this paper.

The paper is organized as follows: Section 2 briefly recalls spatial data structures for ray shooting. In Section 3 we describe the concept of visualization system in detail. Section 4 presents several outputs of the system for particular scenes and concludes the paper.

2 SPATIAL DATA STRUCTURES

Many spatial data structures were developed for ray shooting [Arvo89], however, survey including the development within last ten years is missing. Usually, we need a minimum bounding box enclosing all the objects, that is used at the beginning of the spatial data structure construction. Let us recall commonly used spatial data structures together with original references:

- *uniform grid* [Fujim86] – the bounding box is subdivided in all three axes. The subdivision can be either uniform or non–uniform. The number of voxels created is usually derived from the number of objects in the scene.
- *BSP tree* [Kapla85] – it is given a box in each step of construction, that is split into two child boxes of identical size, then construction continues recursively. The splitting plane orientation is regularly changed for all three axes.
- *kd–tree* [Glass84] – similar to BSP tree, but the splitting planes can be positioned arbitrarily, mostly using some cost function. This results in child boxes of different sizes.
- *octree* [Samet88] – the box is subdivided into eight child boxes, then construction continues recursively.
- *hierarchy of grids* – the basic principle is that the cell of the uniform grid is subdivided with another uniform grid. Three approaches were published – RecGrid [Jevan89], AdaGrid [Klima97], and HUG [Cazal97].
- *hierarchy of bounding volumes* – each node of hierarchy contains the list of references to objects or its descendants.

The common concept is subdivision of space, either elementary or hierarchical. All the listed methods were developed in our rendering system named GOLEM [GOLEM]. The hierarchy of

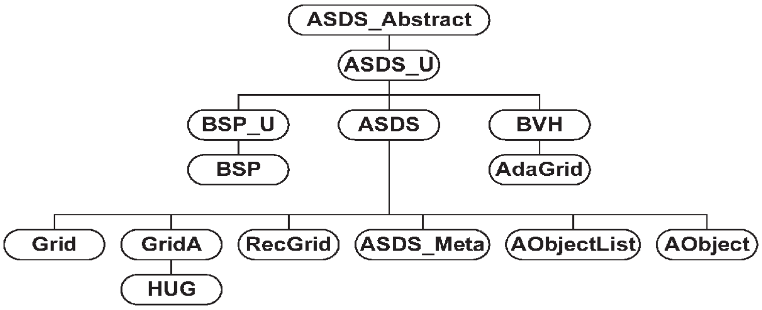


Figure 1: Hierarchy of spatial data structures in GOLEM rendering system

these methods as implemented within the GOLEM system is depicted in Figure 1.

There are several important aspects of spatial data structure influencing the performance of ray shooting algorithm in global illumination applications:

- *the subdivision density* – how many spatial cells should be created to achieve the best performance ? What should be arity of tree data structure, the termination criteria for building such hierarchies ?
- *data representation* – either explicit representation, hash tables, or other special approaches for sparsely engaged data structures.
- *traversal algorithm* – the traversal code is essential for the performance as well. Sequential, DDA, recursive, and neighbour–link algorithms were developed.

Although it could seem that in general there is no difference between different implementations, practically these details are very important for application and resulting performance. For example, there have been about 18 papers written about different octree algorithms for ray shooting purposes [Havra99].

Our goal was to developed such a visualization system with graphical user interface and portable to different operating systems, that enables us to visualize efficiently any spatial data structure in the uniform framework in real–time or interactive way.

3 VISUALIZATION SYSTEM

The requirements posed in the previous section have many aspects. Let us now introduce the designed system, named VIS–RT, that tries to cover them all.

3.1 Uniform Framework

The concept of uniform visualization framework is strongly connected with the hierarchy of spatial data structures. Mostly, advanced data structures(hierarchical grids) use simpler data structures (uniform grids). Generally, such a concept is called meta–hierarchy and it was introduced in [Arvo90], but we are not aware of any real implementation in the past. We implemented such a hierarchy of different spatial data structures within GOLEM system as depicted in Figure 1.

Analysis of common properties of spatial data structures has lead us to *general node* properties:

- the node represents an *axis–aligned bounding box* with non–zero volume, the geometric extent of the node is stored in the node.

- the node contains *list of references to its child nodes*, that have common intersection with the box. This is required when node is interior node in a hierarchy.
- the node contains *list of references to objects* that are assigned to the node. It is needed when node is a leaf of a hierarchy.
- the node contains information about *the depth of the node* in the original hierarchy.

To fulfill all the requirements posed on such a hierarchical node, we have used general n–ary tree of bounding boxes, that actually represents a hierarchy of bounding volumes.

There are no further restrictions to the node. Firstly, it means that children's bounding boxes of a given node can overlap. Secondly, the node can contain the references to the objects and other hierarchical nodes at the same time. Such a general node is not suitable for ray shooting algorithm, since its ray traversal is rather inefficient due to its generality. We use this concept of general node just to emulate all the spatial data structures for visualization purposes.

System Description

The GUI interface of system was implemented using Qt Library [QtLib], that is portable GUI library running on different UNIX and Windows platforms. OpenGL was used for rendering of primitives in three dimensions. The VIS–RT application is built over GOLEM library, that contains the spatial data structure algorithms. The VIS–RT application enables us to examine details of spatial data structures with selected level of detail.

Input Data

There are two modes to obtain the input data for the VIS–RT application. In *offline* mode the spatial data structures are generated within GOLEM system and saved to a special format file. Later, the data are loaded into the VIS–RT application. In *online* mode, when the VIS–RT is started, user can specify types and parameters for GOLEM library and the VIS–RT generates spatial data structures without restarting the VIS–RT application. Similarly, the spatial data structures can be then also saved to a file.

Viewing Modes

There are several viewing modes of spatial data structures, that follow from a general node requirements. The visualization program uses three different, simultaneously displayed *windows*.

- *node window* – just one general node of the hierarchy is displayed and highlighted in a list box. Such a node is called *active* node in the VIS–RT application, since it is connected with other nodes. User can traverse the hierarchy down to a selected node, up, and directly to the root. The geometrical extent of the node and its depth in the original hierarchy is displayed.
- *graphical window* – displays the spatial data structures with or without the scene objects in perspective or parallel projection, active node is highlighted.
- *tree window* – the hierarchy is visualized as acyclic graph, active node is again highlighted. A user can select active node of hierarchy in this window.

The node and graphical window are always displayed. The tree window is displayed on a user's request.

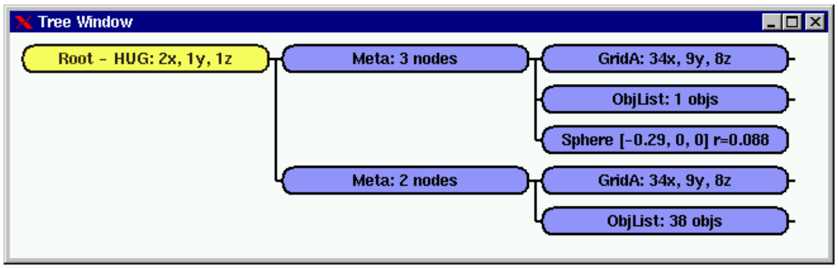


Figure 2: An example of *tree window* in the VIS–RT application

There are five *display modes* to visualize/traverse spatial data structures:

- *tree mode* – a user traverses the spatial hierarchy in node or tree window; the traversal can influence the visualization in graphical window according to the selected camera mode described below. This mode is basic traversal mode, that has been shown the most useful.
- *path mode* – a user can view only the nodes on the path from the root node to an active node. Active node is selected in the node or the tree window.
- *levels mode* – a user can view the extent of details of selected level just to make an initial estimate about the spatial data structure and its complexity. In this mode all the bounding boxes up to the selected levels are displayed.
- *object mode* – this mode serves to get information about the object. A user can select object in all the modes, then system highlights all the nodes where object is directly referenced. It is especially suitable to verify the correctness of spatial data structures.
- *scene mode* – the mode is similar to tree mode, but it displays only the part of the hierarchy and objects in region of interests. This region is a bounding box specified by a user, so all the bounding boxes having common intersection with the region of interests are displayed.

We have found out the tree/scene mode the most useful in general, the object mode for debugging the correctness of spatial data structures.

Camera positioning in space for graphical window has been shown also very important. Right camera setting can greatly improve user's orientation. A user arbitrarily selects from three *camera modes*:

- *center* – camera is oriented to the center of the scene. The position of the camera can be changed by a user interaction. The whole scene is always displayed.
- *adaptive* – similar to center; the active node is always zoomed to cover the whole window and the camera is centered to the center of active node.
- *local* – general positioning of camera position, orientation, and zoom.

It is also possible to reset the viewing position to the initial viewing position, that is specified in the scene description.

A user arbitrarily selects the type of rendering for both spatial data structures and the objects in the scene. This selection is identical with the OpenGL features: *flat* shading, *Gourad* shading, *wireframe* mode, and *disabled* rendering. Additionally, it has been shown practical that the rectangles of the bounding boxes can be also rendered *transparently* with alpha channel.

Other Features

To further improve visualization of required features of spatial data structures extending features were implemented.

First, the VIS–RT application contains specific support for level of detail. It is performed with the two thresholds. One threshold is for the size of a bounding box, one is for the level within the hierarchy.

Second, we have found out interesting to compare the different spatial data structures for the same scene. For this purpose, the visualization of spatial data structures can be run in the tracking mode. The application run as first one is in master mode. All of the successive application can run in the slave model; they track the camera positioning of the master window. User can then view several spatial data structures on the same scene with the same camera setting.

Third, a user can specify arbitrary color settings for all the important features of graphical window. This setting can be saved/loaded to/from a file.

Fourth, a user can select arbitrary object in the scene and hide it from the graphical window.

4 RESULTS AND CONCLUSION

We implemented the VIS–RT and verified that it is useful for designed purposes. Several results of visualization are depicted on Figures 2, 3, 4, 5, and 6. The VIS–RT enables us to examine effectively local and global properties of spatial data structures for ray shooting. Several implementation bugs in the spatial data structures were really found out. For example, there have been singularities of intersection an object with the node geometric extent for octree; the leaves of octree only touching the objects also contained the references to the objects, that made worse performance for ray shooting.

Application of VIS–RT application to spatial data structures is restricted only by properties of general node; it covers all the spatial hierarchies introduced up to now. At the present, the results of visualization are used to design new and more efficient spatial hierarchies with good performance for huge number of primitives and sparsely occupied scenes.

Acknowledgment

This project has been supported by Czech–Austrian scientific cooperation grant Aktion number 1999/17.

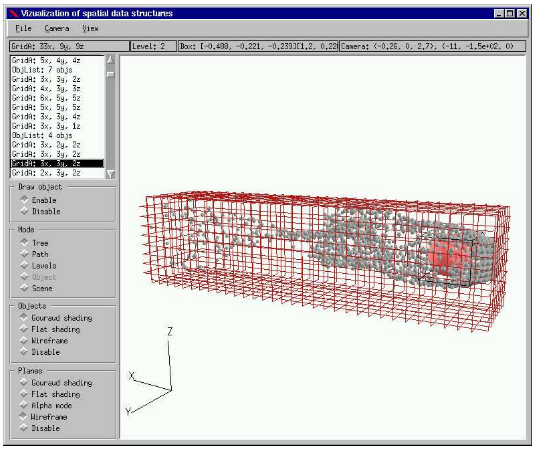
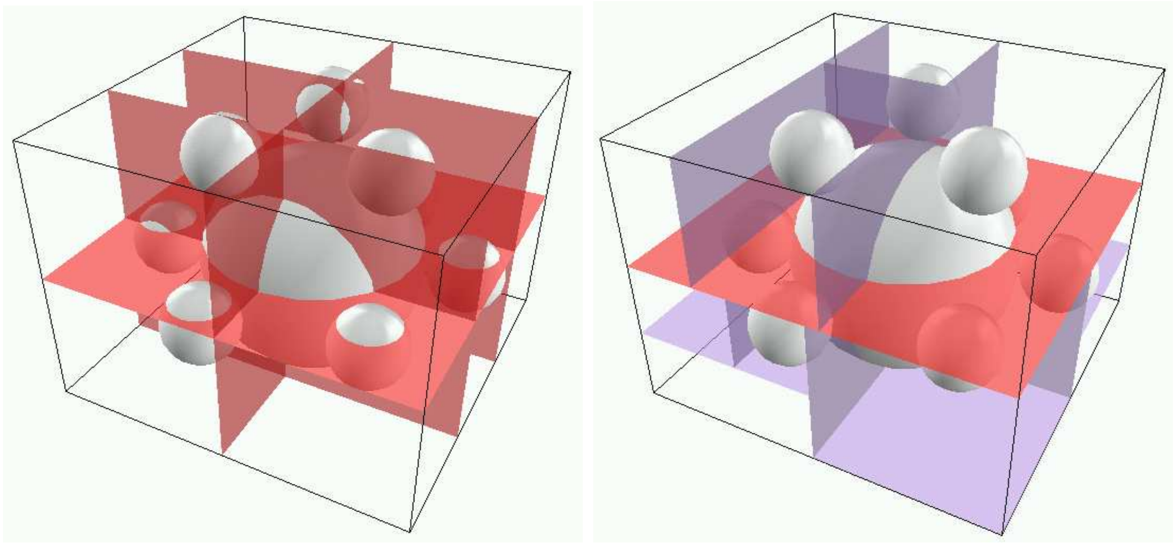


Figure 4: The VIS–RT application with *node* and *graphical window*

REFERENCES

[Arvo89] Arvo, J., Kirk, D.: A survey of ray tracing acceleration techniques,



(a) (b)

Figure 3: Visualization of a simple scene (a) uniform grid (b) kd–tree

In *An Introduction to ray tracing*, pp. 201–262, Academic Press, 1989.

[Arvo90] Arvo, J.:*Ray Tracing with Meta–Hierarchies*, Course Notes, SIGGRAPH'90, 1990.

[Cazal97] Cazals, F. and Puech, C.:Bucket–like space partitioning data–structures with applications to ray–tracing. In *13th ACM Symposium on Computational Geometry*, pp. 11–20, Nice, 1997.

[Fujim86] Fujimoto, A. et al: ARTS: Accelerated ray tracing system. In *IEEE Computer Graphics and Applications*, Vol.6, No.4, pp. 16–26, 1986.

[Glass84] Glassner, A.S.: Space subdivision for fast ray tracing. In *IEEE Computer Graphics and Applications*, Vol.4, No.10, pp. 15–22, 1984.

[GOLEM] GOLEM rendering system, http://www.cgg.cvut.cz/GOLEM, 1996–99.

[Havra99] Havran, V.: A Summary of Octree Ray Traversal Algorithms, in *Ray Tracing News*, http://www.acm.org/tog/resources/RTNews/html/rtnv12n2.html, Vol.12, No.2, 1999.

[Jevan89] Jevans, D. and Wyvill, B.: Adaptive voxel subdivision for ray tracing. In *Proceedings of Graphics Interface '89*, pp. 164–172, 1989.

[Kapla85] Kaplan, M.: Space-Tracing: A Constant Time Ray-Tracer. In *SIGGRAPH'85 State of the Art in Image Synthesis seminar notes*, 1985.

[Klima97] Klimaszewski, K.S., and Sederberg, T.W.: Faster ray tracing using adaptive grids. In *IEEE Computer Graphics and Applications*, Vol.17, No.1, pp. 42–51, 1997.

[QtLib] Qt Library, GUI C++ portable library, http://www.troll.no, 1994–999.

[Samet88] Samet, H., Webber, R., E.: Hierarchical data structures and algorithms for computer graphics, *IEEE Computer Graphics and Applications*, Vol.8, No.4, pp.59–75, 1988.

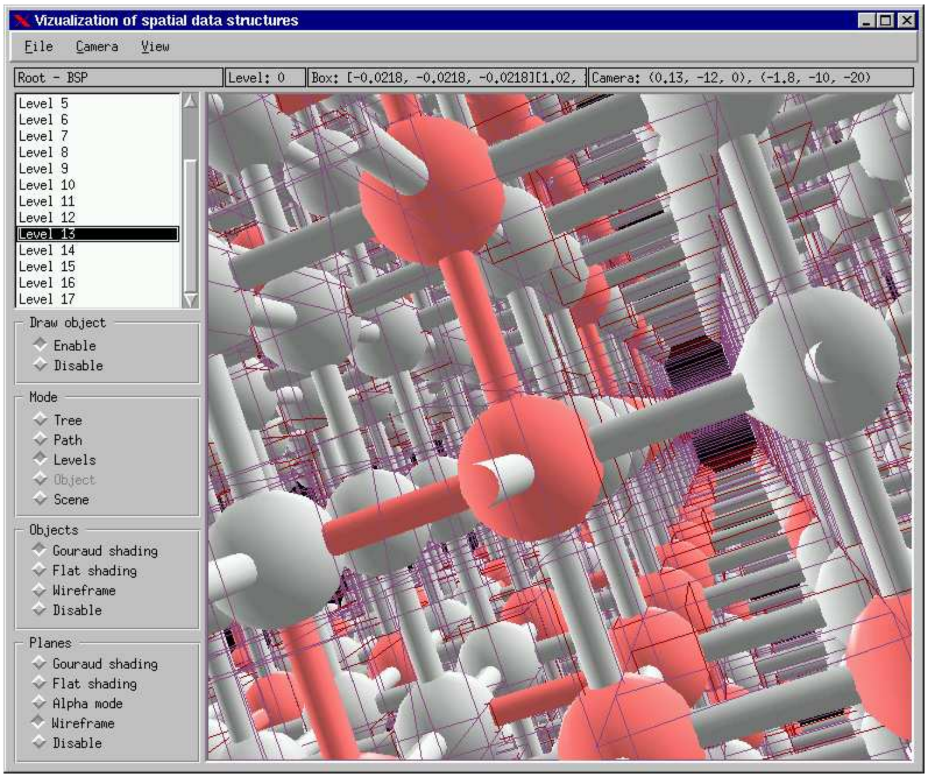


Figure 5: Perspective projection for benchmark scene *lattice*

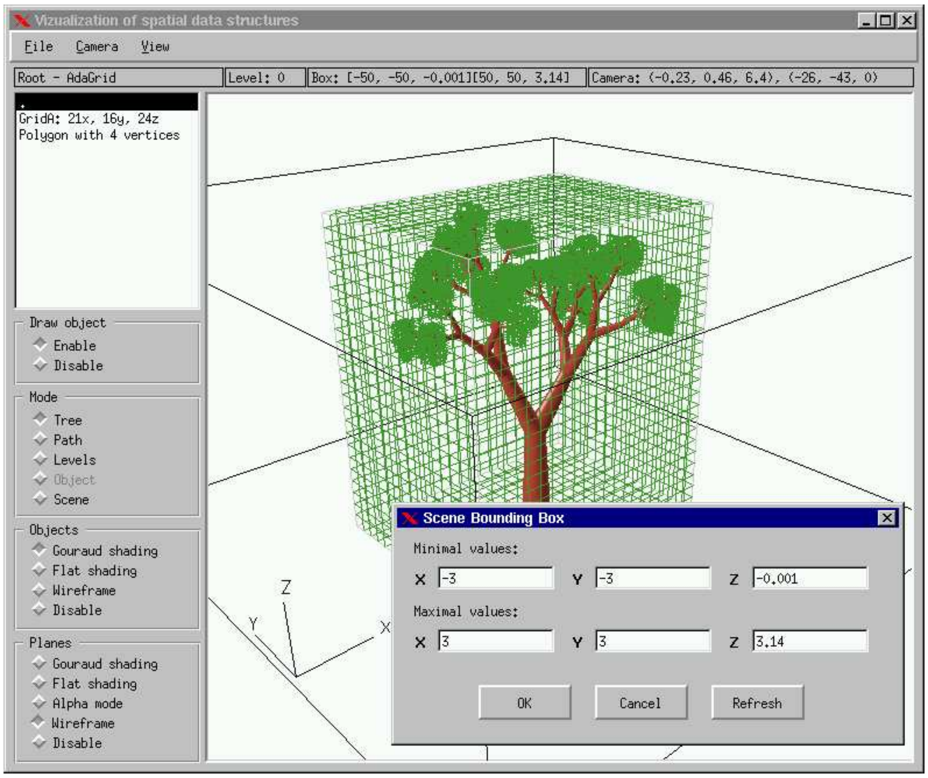


Figure 6: Tree mode for benchmark scene *tree*, and the dialog for level of detail.