EUROGRAPHICS 2000 / M. Gross and F.R.A. Hopgood
(Guest Editors)

Volume 19 (2000), Number 3

LCTS: Ray Shooting using Longest Common Traversal Sequences

V. Havran and J. Bittner

Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University,
Karlovo nám. 13, 12135 Prague 2, Czech Republic

Abstract

We describe two new techniques of ray shooting acceleration that exploit the traversal coherence of a spatial hierarchy. The first technique determines a sequence of adjacent leaf–cells of the hierarchy that is pierced by all rays contained within a certain convex shaft. This sequence is used to accelerate ray shooting for all remaining rays within the shaft. The second technique establishes a cut of the hierarchy that contains nodes where the hierarchy traversal can no longer be predetermined for all rays contained within a given shaft. This cut is used to initiate the traversal for all remaining rays contained in the shaft. The description of the methods is followed by results evaluated by their practical implementation.

Keywords: ray shooting, ray casting, BSP tree, traversal coherence, hidden surface removal.

1. Introduction

Many modern global illumination techniques are based on discrete sampling of *lighting* within the scene, where the geometrical relationships (visibility) between objects are determined using ray shooting.

A very large amounts of rays are cast by most of the global illumination methods, hence we are giving the ray shooting algorithm special attention. The portion of the rendering time spent by ray shooting is usually quite significant and it is not rare that it takes more than 50% of the total rendering time.

In this paper we present two new methods utilizing spatial coherence of visibility that use the concept of *longest common traversal sequence* (abbreviated to LCTS further in the paper). The presented methods extend the ray shooting acceleration based on a hierarchical spatial subdivision, namely using a rectilinear *binary space partitioning tree* (BSP) tree. A rectilinear BSP tree (all its splitting planes perpendicular to principal axes) is also sometimes called kd–tree.

Let us call a cell of the spatial subdivision *elementary* (*hierarchical*) if it corresponds to a leaf (interior) node of the spatial hierarchy. Rays laying within a certain convex shaft are likely to pierce the same set of hierarchical and elementary cells of the spatial subdivision. We call this phenomenon a *traversal coherence*. The basic concept of traversal coherence for elementary cells is illustrated in Figure 1.

Our first technique determines a LCTS for a given convex region (shaft) R consisting solely of leaf–nodes of the spatial hierarchy. We call the resulting LCTS the *simple* LCTS (SLCTS). The SLCTS (if it exists) can be used for the traversal for all rays contained within R. As it will be shown later, if no intersection is found using the current SLCTS the traversal continues using some conventional traversal technique, such as a neighbour–link scheme for BSP trees.

The second technique uses more elaborate treatment of the information gained during traversal of the spatial hierarchy. It determines a *hierarchical* LCTS (HLCTS), that corresponds to a sequence S of nodes of the spatial hierarchy. These nodes form a cut of the hierarchy at the level where the traversal can no longer be predetermined for all rays located within R. For any ray located in R we can avoid traversal steps from the root of the hierarchy to the nodes in S.

As we show later, this concept can be further extended by pruning adjacent elementary nodes that do not contain any objects. Another extension is to determine a termination object (if it exists) that is hit by all rays located in R.

The rest of the paper is organized as follows: In Section 2

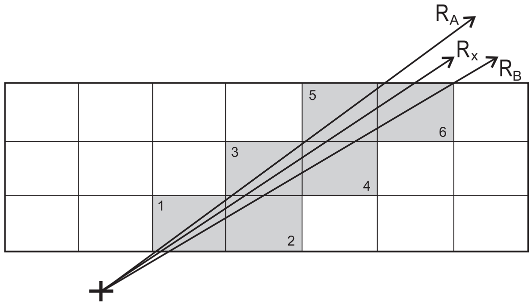


Figure 1: *The concept of traversal coherence in two dimensions. An arbitrary ray R_X lying between rays R_A and R_B pierces the same sequence of elementary cells.*

we describe the previous work related to the approach presented here. Section 3 gives an overview of the algorithm for the construction and traversal of rectilinear BSP trees. In Section 4 we present the LCTS construction algorithm in detail. Section 5 describes utilizing LCTS for ray shooting between two patches and hidden surface removal based on ray casting. Section 6 presents results based on a practical implementation. Finally, Section 7 concludes the paper with several possible directions for future research.

2. Previous Work

The ray shooting problem can be defined easily as to determine the closest intersection given a ray with an object in the scene. In spite of this simple definition the problem itself is difficult to solve efficiently for a large number of objects. The naive algorithm tests all objects for an intersection with a given ray. Its $\Theta(N)$ time complexity makes it unacceptable for number of objects typically used nowadays.

Ray shooting has been studied from different points of view within the computational geometry and computer graphics communities. The approaches of computational geometers are typically aimed at the worst–case time complexity and the algorithms and data structures are mostly restricted to scenes with polygons. There are several known algorithms in $\Re^3$ space according to the certain restrictions of objects. For example, Agarwal and Sharir [1] presented an approach that takes $O(log^2(K))$ query time with $O((M.K)^{2+\varepsilon})$ preprocessing time and space for M possibly intersecting polyhedra with a total K faces. Unfortunately, all known approaches in computational geometry have unacceptable space and preprocessing time complexity for scenes with only hundreds of objects. Szirmay–Kalos and Marton [30] stated the lower bound $\Omega(logN)$ of worst–case time complexity for ray shooting, where N is the number of objects.

The computer graphics community devoted much attention to practical solutions regardless of the worst–case time complexity. Even if the worst–case complexity of these practical techniques is unfavourable, the good average–case complexity is the reason why these methods are commonly used in todays rendering packages. The classification of these techniques is given by Arvo [4] and more recently by Simiakakis [26]. In this paper, we follow the approach of these practical techniques.

The principle of spatial subdivision techniques is the reduction of ray–object intersection tests by decomposing scene space into disjunct cells. The ray–object intersection is then computed using only objects in cells located along the ray path. Although the number of intersection tests is reduced significantly, there are additional time requirements for the *traversal* of the spatial subdivision. One way of ray shooting acceleration is to improve the properties of the spatial subdivision and thus to lower the average number of intersection tests per ray. Another technique is to improve the traversal algorithm by decreasing the number of traversal steps or the cost of a single traversal step. In this paper we deal with decreasing the number of traversal steps.

Several papers related to the approach presented here have been published. Concepts of generalized rays were introduced; cone tracing [2], beam tracing [17], and pencil tracing [25]. Arvo and Kirk [3] presented a ray classification method, which subdivides the five dimensional ray space. For each cell of this subdivision, a sorted list of objects is constructed and a ray r is tested for intersection only with objects corresponding to the elementary cell that is intersectedT by r. Simiakakis and Day presented a technique that improves the space complexity of ray classification by adaptively subdividing the ray space [27]. The memory complexity of this approach was improved by Kwon et al in [21] by reducing the ray space from five to four dimensions. The ray coherence theorem [23] is a generalization of the light buffer [11] approach. It uses directionality of rays and a binary search. Haines and Wallace [12] utilized the concept of *shaft*; the ray–object intersection tests are restricted only to objects that intersect a shaft connecting two patches. Teller and Alex [31] subdivide the viewing frustum by combining Warnock's visibility algorithm and beam tracing. Similar use of coherence was presented by Gonzáles and Gisbert for an octree [8]. Pyramid clipping of a spatial subdivision aimed at a parallel implementation for ray traversal was presented by van der Zwaan et al [32]. The technique of directed safe zones utilizing free adjacent elementary cells within a uniform grid was published in [24]. Genetti et al [7] presented recently an approach for adaptive supersampling in object space, using pyramidal rays. All the methods mentioned utilize some concepts of *coherence*, that were surveyed by Gröller [9].

The method presented in this paper is a combination of directional techniques with a hierarchical spatial subdivision. We have based our work on BSP trees, since the recently presented results [28] show that adaptively constructed rectilinear BSP trees are more efficient for ray tracing acceleration than nowadays popular hierarchical grids [20, 5].

3. Classical Algorithm Overview

In order to describe the proposed techniques we first briefly review the construction of rectilinear BSP trees for ray shooting as well as the classical traversal algorithm.

3.1. BSP Tree

A rectilinear *Binary Space Partitioning tree* (abbrev BSP tree) is a higher dimensional analogy to the binary search tree. A BSP tree for a set S of objects is defined as follows: Each node v in the BSP tree corresponds to a non–empty axis–aligned box B_v, we call this box a *cell*. The cell associated with the root of the tree is the bounding box of all objects from S. Each interior node v of BSP tree is assigned a cutting plane H_v, that divides B_v into two cells. Let H_v^+ be the positive halfspace and H_v^- the negative halfspace bounded by H_v. The cells associated with the left and the right child of v are $B_v \cap H_v^+$ and $B_v \cap H_v^-$, respectively. The left subtree of v is a BSP tree for the set of objects $S_v^- = \{s \cap H_v^- \neq \emptyset | s \in S_v\}$, the right subtree is defined similarly. Each leaf node l contains a list of objects S_l that intersect the node bounding box B_l. The leaves of the BSP tree are either occupied by objects or vacant. A two dimensional example of a BSP tree is depicted in Figure 2.

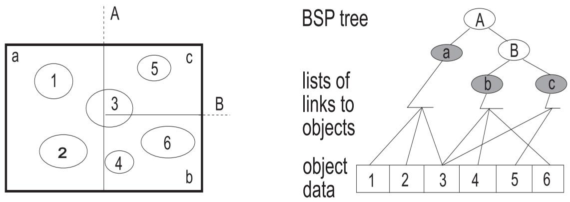


Figure 2: *A two dimensional example of a BSP tree.*

The orthogonality of the cutting planes of a BSP tree significantly simplifies the computation of ray shooting queries. The computation cost of the signed distance t of intersection point of a ray with the cutting plane is roughly three times lower than for an arbitrary positioned plane.

A BSP tree is constructed hierarchically in top down fashion. At a current leaf l a cutting plane is selected that subdivides B_l into two cells. The leaf then becomes an interior node with two new leaves. The objects of l are distributed into new descendants of l. The process is repeated recursively until certain termination criteria are reached.

An important feature of the rectilinear BSP tree is its adaptability to the scene geometry that is induced by possibility to position the cutting plane. Traditionally, the cutting plane is positioned in the mid–point of the chosen axis, and the order of axes is regularly changed on successive levels of the hierarchy [19]. Another method uses adaptive positioning of cutting planes when the position of the cutting plane is chosen along the whole range [22] using surface area heuristics. We use the latter approach that can improve the performance of BSP tree for ray shooting queries over the mid–point subdivision scheme by orders of magnitude [13], especially for sparsely occupied scenes.

3.2. BSP Tree Traversal

Given a ray and a BSP tree we need to identify elementary cells along the ray path. This task is solved by a *traversal algorithm* [15, 18, 29].

Here we only recall the basic idea of the BSP tree traversal. At each node of the BSP tree we determine one of four possible cases: to traverse only the left child, only the right child, the left child first and then the right one, or the right child first and then the left one. When both child nodes have to be traversed, the farther node is stored on the *traversal stack* and the nearer one is traversed first. This algorithm proceeds recursively until a leaf node is encountered. The objects in the leaves are tested for intersection with the ray. If there is an intersection of ray with object(s) lying in the leaf's bounding box, the object closest to the ray origin is selected and the traversal terminates. Otherwise, a node is popped from the traversal stack and ray traversal continues as described above until an intersection is found or the traversal stack is empty (no intersection is found).

It is obvious that the ray traversal includes the determination of traversal order of the nodes of the hierarchy starting always from the root node. This behaviour can be eliminated by extending the BSP tree by neighbour–links (ropes) [14, 22]. The modified traversal algorithm uses these links to determine the next elementary cell pierced by the given ray. Since these links usually point to rather deep nodes of the hierarchy or even leaf nodes [14], the number of traversed interior nodes can be decreased significantly. On the other hand the cost of the traversal step from the leaf nodes is slightly higher.

The proposed approaches aim to combine advantages of both the hierarchical and link based traversal of the BSP tree for a restricted set of rays.

4. Construction of LCTS

The LCTS is constructed for a convex *shaft* defined by a set of *boundary rays*. Typically, these rays form edges of a *frustum* (if they share the origin) or edges of a *tunnel* (if the rays are parallel). For each of the boundary rays a *traversal history* is stored. This information is used to construct the LCTS, that is common to all rays belonging to the shaft. We distinguish between two types of LCTS. The first type – SLCTS (*simple LCTS*) exploits coherence of traversal using only leaf nodes of the hierarchy. The second one – HLCTS (*hierarchical LCTS*) allows traversal coherence to be used for hierarchical nodes as well, but requires more computational effort for its construction.

4.1. SLCTS

The concept of SLCTS is depicted in Figure 1. Assume a convex shaft defined by several rays that traverse the same sequence S of elementary cells of a BSP tree. Then an arbitrary ray lying within the shaft traverses sequence S as well. The origin of the ray has to be positioned in the shaft (if the shaft is a tunnel). There are some potential problems to be solved for SLCTS as depicted in Figure 3:

1. No common sequence of leaf nodes exist (Figure 3, case 1).
2. Having some initial sequence of elementary cells S for the rays defining a LCTS, then the last common cell for them is known. If a ray does not hit any object in S, which cells have to be traversed then ? (Figure 3, case 2).

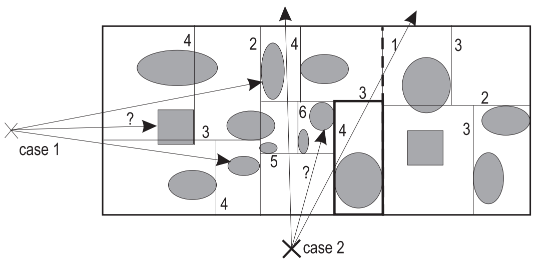


Figure 3: *Two potential problems of SLCTS to be solved. The numbers mark the depth of the cutting planes in the hierarchy.*

We use a simple solution to the first problem; we apply any traditional traversal algorithm for BSP tree. The second problem is solvable for BSP trees with the use of neighbour–links. When no object is intersected using the sequence S, then the ray traversal is performed further using the links (ropes and rope trees) to neighbour–leaf cells. It is worth mentioning the SLCTS is applicable not only to BSP tree, but for any spatial subdivision, which enables us to continue the traversal without the down–traversal phase, e.g. uniform grids.

4.2. HLCTS

The second proposed method uses the HLCTS and exploits also traversal coherence of interior nodes of the hierarchy. We describe the details of HLCTS construction and traversal below.

4.2.1. Traversal Trees

A traversal history for a given ray can be stored by means of a *traversal tree*. The traversal tree is a binary tree, where each node N of the tree corresponds to a node B in the scene BSP tree that was visited in the scope of the traversal. Additionally, the node contains the information about the further traversal (traversal decision) that reaches one of the following five states: LEFT, RIGHT, LEFT/RIGHT, RIGHT/LEFT, and TERMINATION. The traversal state TERMINATION corresponds either to pierced leaf–nodes of the BSP tree or interior nodes that were pushed on the traversal stack, but as the ray has been terminated, these nodes were not used for further traversal. Other traversal states express the order of the traversal of the BSP tree "below" the node B. Additionally, node N contains a pointer to an *exit–plane*, that is a plane bounding the node B cell along the direction of the ray (see Figure 4). The use of exit–plane pointer will be described further in the text.

If the node N is not a leaf, then the left child corresponds to the BSP tree node B_1 that was visited first during the traversal. The right child (if any) corresponds to the BSP tree node B_2 pushed on the traversal stack and thus visited later or unvisited (if the ray has been terminated before reaching this node). See Figure 4 that depicts an example of the traversal tree structure.

4.2.2. Constructing Initial HLCTS

The *initial HLCTS* is constructed using n traversal trees ($n > 1$) determined for n boundary rays of a given convex shaft. Using convexity, it can be shown that the traversal decision for a given node of the BSP tree does not change for all rays within the shaft if the corresponding traversal decisions for all boundary rays are equal. The hierarchical traversal of such nodes can be avoided by descending the hierarchy and constructing an ordered sequence of nodes where the traversal state is no longer equal. The HLCTS can be seen as a cut on the BSP tree at the level where the traversal can no longer be precomputed from the traversal histories of the boundary rays. Figure 5(a) depicts the boundary rays of a frustum.

The HLCTS construction algorithm performs a constrained depth first search in parallel on all n traversal trees. If the traversal states associated with all n currently reached interior nodes are equal, the algorithm is applied recursively first on the left child and then on the right child (if any). If the reached nodes are leaves of the traversal trees (state=TERMINATION) or the traversal states are not equal, the HLCTS is enlarged using the BSP tree node associated with the reached nodes. Additionally, each HLCTS entry contains n pointers to the associated nodes of the traversal trees (see Figure 5(b)). Their use will be explained further in the text.

Once the initial HLCTS has been constructed, it can be used to initiate the traversal for all rays within the corresponding shaft. The traversal stack can be filled using all nodes of the HLCTS. Note that commonly used traversal algorithm usually assumes that the entry and exit points are known for the current node. Using a HLCTS these must be computed explicitly for each visited node of the HLCTS, since they have not been determined recursively as in the classical traversal algorithm. The pointers to exit planes are used to solve this problem.

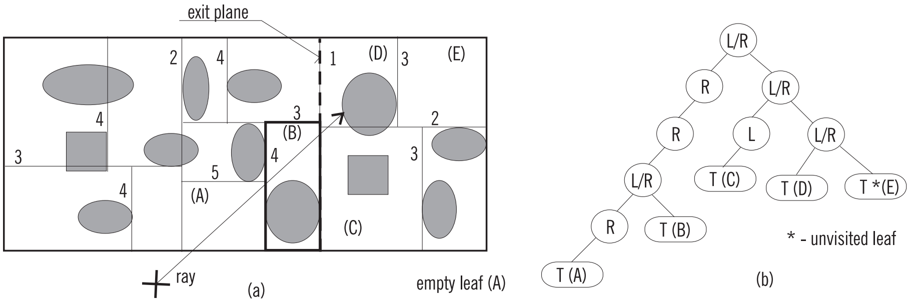


Figure 4: *(a) Exit plane – the ray leaves the marked leaf node at the face in the exit plane that is formed by the cutting plane of the root node. (b) Traversal tree corresponding to the ray path. Notation for nodes of traversal tree: L – LEFT, L/R – LEFT/RIGHT, R – RIGHT, T - TERMINATION.*

4.2.3. Constructing General HLCTS

The HLCTS for a given shaft can be refined further by constructing HLCTS for its sub–shafts. Nevertheless, if the traversal of the hierarchy does not originate at the root node of the BSP tree, the traversal history no longer corresponds to a single tree. Instead, it is stored as a sequence of traversal trees, with their roots corresponding to the nodes of the HLCTS, that was used to initiate the traversal.

We consider constructing HLCTS for the n traversal histories with the following properties:

1. all traversal histories have been generated from the same HLCTS and thus they correspond to sequences of traversal trees of the same length.
2. k traversal histories have been generated from the same HLCTS, say L. Other $n-k$ histories have been generated before, but they have been used to establish L.

The first case can be solved simply, by applying the previously mentioned algorithm on all n–tuples of root nodes. If the algorithm is implemented using a stack, these n–tuples can be initially pushed on the stack in the reverse order.

In the second case the information stored within L is used. Although the $n-k$ "old" traversal histories correspond to sequences of traversal trees of length $x \leq |L|$, each entry of L contains pointers to the traversal tree nodes, that correspond to this HLCTS entry. Using these pointers the "old" traversal histories can be accessed directly at the "level" corresponding to L. Thus the algorithm is applied on n–tuples of traversal history nodes, where k entries correspond to roots of the traversal trees generated using L. Other $n-k$ entries are determined using appropriate pointers stored within entries of L. The problem and its solution are illustrated in Figure 5(c).

4.3. Further Improvements

In this section we present several improvements that can be used in the scope of the construction of both SLCTS and HLCTS.

4.3.1. Unification of Empty Leaves

Although the BSP tree is built adaptively with respect to the scene geometry, it can happen that the LCTS contains a subsequence S of entries corresponding to empty leaves of the scene BSP tree. This is depicted in Figure 6. The LCTS construction algorithm can be modified to detect such situation and to replace S with a single entry. This approach is analogous to directed safe zones [24] that would be constructed on the fly.

The modification of the LCTS construction algorithm is straightforward. If a new LCTS entry E that corresponds to an empty leaf node is to be added, it is first checked if the last entry L of the constructed LCTS corresponds to an empty leaf of the BSP tree. If this is the case, the LCTS is not enlarged by E. Instead the "exit–plane–node" of E is used to replace the exit–plane–node of L. In such a way, the spatial extent corresponding to L is enlarged properly for all rays within the shaft. It is necessary for the correct behaviour of the ray traversal algorithm.

4.3.2. Termination Object

A remarkable reduction of traversal steps can be obtained by determining a single convex *termination object* that is hit by all rays in a given convex shaft. It is possible only if the origin of the rays is somehow restricted. This holds for example for a pyramidal shaft where all rays originate at the same point.

If there is a termination object, the ray traversal can be eliminated completely. Using LCTS we can perform a simple test that optionally determines the termination object for the LCTS. The presented approach is *conservative*, since it does not always determine the termination object even if it exists, but it never gives a wrong answer if no termination object exists for a given shaft.

The termination object exists if all of the following conditions hold:

- Condition 1 – all boundary rays of the shaft hit the same object O and are terminated in the same cell C.

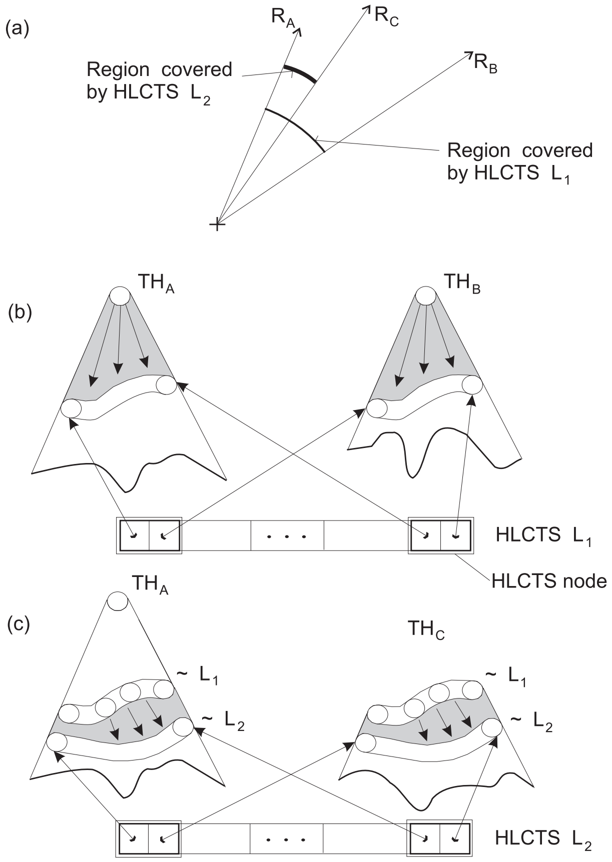


Figure 5: *HLCTS construction and use: (a) Underlying geometry, two boundary rays R_A and R_B, the ray R_C between boundary rays, and the regions defined by the rays. (b) HLCTS L_1 generated from two traversal histories TH_A and TH_B corresponding to boundary rays R_A and R_B. (c) HLCTS L_2 generated from traversal histories of different number of roots, that is using TH_A and TH_C traversal histories and HLCTS L_1. TH_A is accessed using L_1, TH_C contains four root nodes generated from L_1. Notation: $\sim X$ – cut of the traversal history corresponding to HLCTS X. TH – traversal history; TH_A, TH_B for boundary rays, TH_C for a ray between boundary rays.*

- Condition 2 – object O is convex and it is the only object intersecting the cell C.
- Condition 3 – the cells visited before reaching the cell C are empty.

If the unification of empty leaves described above is applied, the last condition reduces to one of the following cases:

- Condition 3a – the cell C corresponds to the first entry of the LCTS.
- Condition 3b – the cell C corresponds to the second entry

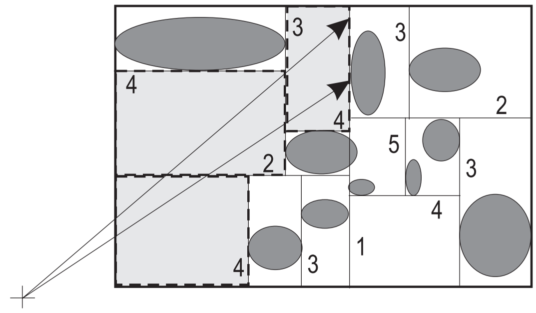


Figure 6: *Unification of empty leaves. For two rays with common origin three empty leaves can be found.*

and the first one corresponds to an empty leaf (unification of empty leaves).

4.3.3. Initial Leaf Sequence for HLCTS

This improvement is applicable to HLCTS only. If the HLCTS corresponds to a shaft that forms a pyramid, it can be expected that the first entries of the HLCTS correspond to leaves of the BSP tree. In such a case this *initial leaf sequence* of HLCTS forms a SLCTS.

With each HLCTS we keep a single value k that expresses the number of leaves at the beginning of the HLCTS. If $k = |HLCTS|$, the sequence corresponds to a sequence of leaf nodes of the BSP tree. In this case it forms a SLCTS and cannot be refined any longer.

The HLCTS construction algorithm can be modified to copy the first k leaf nodes from the "parental" sequence L without performing any matching. The previously mentioned HLCTS construction algorithm is applied starting at the node $(k+1)$-th node of the HLCTS. Similarly, the index k can be exploited in the traversal algorithm where the first k nodes can be visited without using any traversal stack. If the traversal is terminated before reaching the $(k+1)$-th node, the stack initialization is completely avoided.

5. Application of LCTS

The ray shooting with LCTS concept can be used in many global illumination techniques based on discrete sampling of space via rays as ray tracing, photon tracing, Monte Carlo methods, shadow determination, form factor computation etc. We discuss here two techniques that can be used in a more general way, namely patch–to–patch visibility and hidden surface removal.

5.1. Patch–to–patch Visibility

For the purpose of computing patch–to–patch visibility factors, the HLCTS is more suitable, since it can occur that there is no SLCTS for the given two patches.

The task is to determine mutual visibility using ray shooting for a given two patches in the scene. We create a convex hulls of both patches, then we determine the set S_R of rays that form boundary rays of a convex shaft between these convex hulls. We then construct the traversal trees for each ray in S_R and the corresponding HLCTS, that is subsequently used for *any ray* between the patches.

This application of HLCTS is similar to the concept of shaft culling [12], but there are major differences in the way of obtaining the desired set of cells intersecting the shaft. In the classical shaft culling shaft–cell intersection tests are performed, which can be costly. The HLCTS technique uses only ray shooting and then only non–geometric computations for HLCTS construction, which is less expensive. It is worth mentioning that the results of applying these two techniques need not be the same. Generally, the HLCTS determines a superset of cells determined by classical shaft culling, but in much more efficient way.

5.2. Hidden Surface Removal

Hidden surface removal using ray shooting is usually called *ray casting*. For this purpose, both HLCTS and SLCTS are suitable; significant reduction of traversal steps can be achieved by using a termination object as mentioned in the previous chapter.

The common origin of rays (viewpoint) that induces the initial sequence of the first common nodes in LCTS is likely to be a sequence of leaves (SLCTS). The more paraxial rays, the longer the initial SLCTS. Assuming the cells are farther from the viewpoint, the rays are more likely not to generate the same sequence of BSP tree leaves. Thus scene geometry, BSP tree properties, and the image resolution, influence the level of utilization of traversal coherence.

There are several possibilities how to exploit the concept of LCTS for hidden surface removal. We can deal with an image as with a two–dimensional array or as with an array of one–dimensional arrays (scanline approach). Since both SLCTS and HLCTS techniques can be applied, we have exactly four cases:

- **SLCTS–1D** Scanline with SLCTS: this approach is basically undersampling on a scanline, creating SLCTS for two adjacent samples and using this SLCTS to compute samples between them. This scheme is depicted in Figure 7(a).
- **HLCTS–1D** Scanline with HLCTS: this approach can be implemented as above, but a better utilization of traversal coherence combines HLCTS with bisection. The initial HLCTS is incrementally refined. The scheme is depicted in Figure 7(b).
- **SLCTS–2D** Two–dimensions with SLCTS: the sampling can be performed as undersampling $m \times n$ pixels. Having the sequence of traversed leaves for the four corners of a rectangle, the SLCTS is created and used for samples inside the rectangle. The SLCTS-1D can be seen as special case, when $n = 1$. The scheme is depicted in Figure 7(c).
- **HLCTS–2D** Two–dimensions with HLCTS: the bisection is applied in two dimensions; the axis for splitting is regularly changed. At the beginning four rays corresponding to the pixels in the image corners are cast. In one bisection step, four rays are cast again and two new rectangles are created. See Figure 7(d).

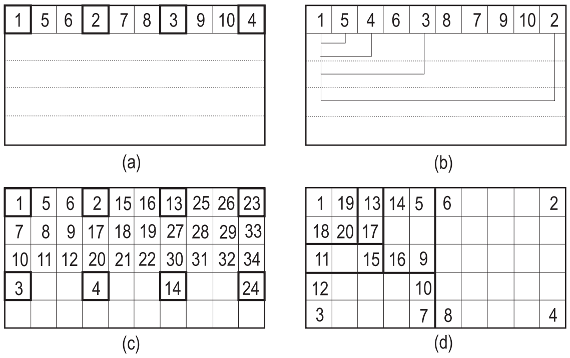


Figure 7: *Hidden surface removal sampling patterns for LCTS: (a) SLCTS–1D (b) HLCTS–1D (c) SLCTS–2D (d) HLCTS–2D. The numbers mark the order how the rays are cast.*

We shall point out that all these sampling schemes can be applied more successfully when the image resolution is high with respect to the space subdivision projected to the image plane. In this case, many areas in the image have a common traversal sequence, that often form SLCTS.

Note that SLCTS could also be used with bisection, but since the rays corresponding to corner pixels are far from being paraxial, these rays do not generate any SLCTS. As we expected, we have found that the undersampling method is more efficient for the SLCTS concept.

6. Results

We implemented all the sampling techniques mentioned in the previous chapter for hidden surface removal and the HLCTS approach for ray shooting between two patches. For testing, we used scenes from *Standard Procedural Database* introduced by Haines [10] and scene *fluid*, which depicts the simulation of liquid flow. BSP trees for all the scenes were built using the surface area heuristics [22].

The basic scene properties and the number of leaves of the constructed BSP trees are listed in Table 1. The termination criteria for BSP tree construction were: the maximum depth of BSP tree equal to 16, the threshold for a node to become a leaf equal to 2 objects.

Scene	balls	fluid	gears	lattice	mount	rings	teapot	tetra	tree
objects	7382	2515	9345	8281	8196	8392	9264	4096	8191
spheres	7381	2514	9345	2197	4	4195	-	-	4095
polygons	1	1	-	-	8192	1	9264	4096	1
cones	-	-	-	-	-	1	-	-	4095
cylinders	-	-	-	6084	-	4195	-	-	-
BSP tree leaves	5253	1917	13830	25689	8554	12924	2387	2972	3426

Table 1: *Testing scenes properties*

6.1. Patch–to–patch Visibility

We have observed that it is difficult to predict the time reduction in advance. If only a low number of rays between the patches is cast, then the HLCTS construction time is not usually recovered later. For tens and hundreds of rays between the patches the HLCTS construction is worthwhile, particularly, when the patches are incrementally refined as in the hierarchical radiosity algorithms. The reduction of the number of traversal steps depends on the shape and the positioning of the constructed shaft in the scene. The more elongated shaft and deeper the BSP tree, the better results can be achieved.

6.2. Hidden Surface Removal

We tested hidden surface removal using ray shooting with hierarchical traversal, rope traversal, SLCTS–1D, SLCTS–2D, HLCTS–1D, and HLCTS–2D. The number of intersection tests per ray (the same for all the traversal methods on the same scene) and the number of traversal steps for primary rays for the 1024×1024 image resolution and for all the traversal methods is shown in Table 2. The hierarchical traversal [15] was used as a reference for comparison. A *rope traversal* designates the method, where the rope tree technique [14] is used without LCTS. The running times in the both tables include the construction of LCTSs, that are built on the fly, so no additional preprocessing is performed.

Table 3 shows the sensitivity of different traversal methods to the resolution of the image for the scene *teapot*.

Note that the pure rope tree traversal method is not efficient compared to a hierarchical traversal. The first reason is the cost of one traversal step is slightly higher than for the hierarchical traversal. The second reason is that for hidden surface removal for tested scenes rays are cast from outside the scene, so many empty voxels have to be traversed before hitting an object. The rope tree traversal is more suitable for higher order rays, that originate on the surfaces of objects [14].

Figure 8* visualizes the traversal coherence for the scene *mount* and the SLCTS–2D method. It is obvious that most pixels in the projection have at least one common initial leaf node.

All the experiments were conducted on the SGI O^2, MIPS R10000, 180 MHz, 256 MBytes RAM running the *Irix 6.3* operating system. The traversal algorithms were implemented within the GOLEM rendering system [6].

6.3. Discussion

The successful use of the LCTS concept for patch–to–patch visibility is conditioned by the number of rays shot between the patches. Similarly, the application of LCTS for hidden surface removal via ray casting depends on the image resolution.

Let us discuss the properties of hidden surface removal using ray shooting with LCTS in detail. Time savings are scene–dependent, nevertheless, this is the case for all the practical ray shooting techniques. It follows from the results that hierarchical traversal steps to the first leaf are successfully avoided, total number of traversal steps is decreased typically by more than 60 % (for scene lattice even by 78 %). This corresponds to the time reduction of 20 % on average, since then most of the computation is then devoted to ray–object intersections. The total time does not include the shading, the ratio of time devoted to the ray shooting to the time of the whole ray tracing is scene dependent and reaches from 40 % to 75 % for used test scenes [13].

Note that the hierarchical traversal algorithm [15] for an arbitrary ray used as reference solution is highly optimized and improves the previously published traversal algorithms in terms of optimal number of decisions per one traversal step. Our preliminary results [16] show that the solution using this traversal algorithm is very close to the optimal one that can be ever achieved for hidden surface removal casting individual rays (please, see the number of ray–object intersection tests per ray and the number of traversal steps per ray in Table 2). It means that improving performance of hierarchical ray shooting algorithm by another 20 % using the LCTS concept is significant.

A special note should be given to the setting of undersampling resolution for SLCTS approaches. We can observe the tradeoff between the undersampling resolution and the possible existence of SLCTS or/and its properties. If we take

Scene	balls	fluid	gears	lattice	mount	rings	teapot	tetra	tree
Intersections/ray	7.40	5.26	4.07	8.75	3.52	10.9	2.96	1.44	6.49
					hierarchical traversal				
$TSPR_{1024}[-]$	30.4	20.5	16.6	49.9	30.7	41.3	22.2	13.7	23.2
$\tau_{1024}[s]$	15.4	12.1	48.1	25.6	12.9	28.6	11.5	7.46	16.1
$\alpha_{1024}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\eta_{1024}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\eta_{4096}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
					rope traversal				
$TSPR_{1024}[-]$	26.2	18.5	15.6	19.1	26.8	34.2	19.0	11.6	16.9
$\tau_{1024}[s]$	16.0	13.2	48.2	23.5	13.4	31.9	12.2	8.65	16.7
$\alpha_{1024}[-]$	0.862	0.902	0.940	0.383	0.873	0.828	0.856	0.847	0.728
$\eta_{1024}[-]$	1.04	1.09	1.00	0.918	1.03	1.12	1.06	1.16	1.03
$\eta_{4096}[-]$	1.02	1.09	1.00	0.913	1.02	1.01	1.06	1.15	1.02
					SLCTS–1D window 5×1 pixels				
$TSPR_{1024}[-]$	13.2	8.19	7.13	11.6	11.9	15.7	8.46	5.70	9.83
$\tau_{1024}[s]$	14.7	11.2	47.7	20.8	11.0	28.6	10.6	8.32	15.9
$\alpha_{1024}[-]$	0.434	0.400	0.430	0.234	0.388	0.380	0.381	0.416	0.424
$\eta_{1024}[-]$	0.958	0.925	0.990	0.812	0.853	0.999	0.925	1.16	0.988
$\eta_{4096}[-]$	0.846	0.891	0.977	0.785	0.787	0.947	0.871	0.987	0.947
$\gamma_{1024}[-]$	0.964	0.993	0.871	0.999	0.938	0.981	0.833	0.560	0.999
					HLCTS–1D				
$TSPR_{1024}[-]$	10.1	6.01	5.58	13.8	8.52	11.0	6.35	4.65	7.66
$\tau_{1024}[s]$	14.7	11.0	48.1	22.3	11.8	27.5	10.6	7.39	15.6
$\alpha_{1024}[-]$	0.332	0.293	0.336	0.277	0.278	0.266	0.286	0.339	0.33
$\eta_{1024}[-]$	0.958	0.907	0.998	0.872	0.913	0.960	0.929	0.990	0.970
$\eta_{4096}[-]$	0.820	0.820	0.972	0.775	0.752	0.868	0.816	0.851	0.895
					SLCTS–2D window 5×5 pixels				
$TSPR_{1024}[-]$	12.4	7.23	6.46	10.8	10.7	13.8	7.73	5.45	9.13
$\tau_{1024}[s]$	12.6	10.3	46.7	19.7	9.99	27.2	9.69	7.24	15.1
$\alpha_{1024}[-]$	0.408	0.353	0.389	0.216	0.349	0.334	0.348	0.398	0.394
η_{1024}[-]	0.819	0.845	0.971	0.768	0.772	0.949	0.846	0.97	0.939
$\eta_{4096}[-]$	0.749	0.800	0.956	0.730	0.691	0.890	0.796	0.892	0.896
$\gamma_{1024}[-]$	0.919	0.980	0.780	0.999	0.916	0.971	0.798	0.525	0.990
					HLCTS–2D				
$TSPR_{1024}[-]$	11.3	6.56	6.37	14.8	9.35	12.7	7.31	5.32	8.61
$\tau_{1024}[s]$	13.6	10.3	47.5	21.7	10.4	26.4	9.60	6.31	14.6
$\alpha_{1024}[-]$	0.372	0.32	0.384	0.297	0.305	0.308	0.329	0.388	0.371
η_{1024}[-]	0.887	0.849	0.987	0.846	0.805	0.921	0.838	0.846	0.903
η_{4096}[-]	0.800	0.784	0.964	0.782	0.691	0.857	0.751	0.719	0.855

Table 2: *Comparison of traversal algorithms for hidden surface removal based on ray casting. Intersections/ray – number of ray–object intersections per ray on average. $TSPR_{1024}$ – number of traversal steps per ray on average for specific traversal method and the resolution 1024×1024. τ_{1024} – the time for ray shooting only (without shading) for a specific traversal method for the resolution 1024×1024. α_{1024} – the ratio between the number of traversal steps for the specific method and the number of traversal steps of hierarchical traversal algorithm for the resolution 1024×1024. η_{1024} – the ratio between the time of the method and the time of hierarchical traversal algorithm for the resolution 1024×1024. η_{4096} – the ratio between the time of the method and the time of hierarchical traversal algorithm for the resolution 4096×4096. γ_{1024} – the ratio of number of SLCTS sequences that contains at least one leaf to the number of all possible sequences for the resolution 1024×1024.*

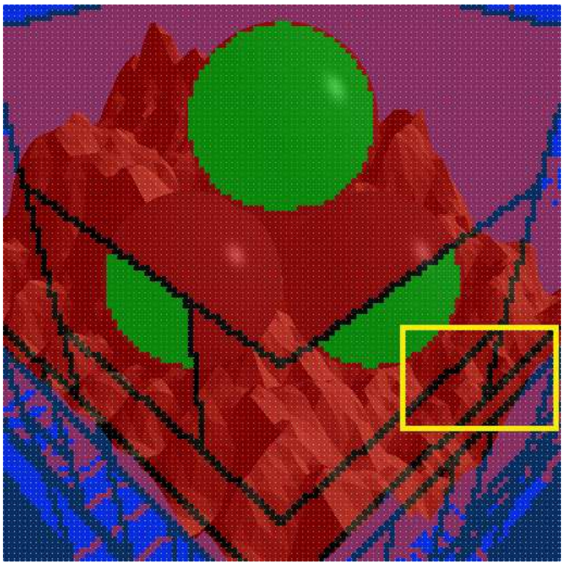
Resolution	256 × 256	512 × 512	1024 × 1024	2048 × 2048	4096 × 4096
			hierarchical traversal		
$TSPR[-]$	22.1	22.2	22.2	22.2	22.2
$\tau[s]$	0.734	2.89	11.5	45.6	182
$\alpha[-]$	1.00	1.00	1.00	1.00	1.00
$\eta[-]$	1.00	1.00	1.00	1.00	1.00
			rope traversal		
$TSPR[-]$	18.9	19.0	19.0	19.0	19.0
$\tau[s]$	0.792	3.08	12.2	48.4	193
$\alpha[-]$	0.855	0.856	0.856	0.856	0.856
$\eta[-]$	1.080	1.066	1.064	1.060	1.057
			SLCTS–1D window 5 × 1 pixels		
$TSPR[-]$	11.4	9.87	8.46	7.31	6.41
$\tau[s]$	0.741	2.77	10.6	40.6	159
$\alpha[-]$	0.516	0.445	0.381	0.329	0.289
$\eta[-]$	1.010	0.959	0.924	0.889	0.871
			HLCTS–1D		
$TSPR[-]$	10.7	8.30	6.35	4.91	3.94
$\tau[s]$	0.782	2.87	10.6	39.6	149
$\alpha[-]$	0.484	0.374	0.286	0.221	0.177
$\eta[-]$	1.065	0.990	0.929	0.868	0.815
			SLCTS–2D window 5 × 5 pixels		
$TSPR[-]$	11.6	9.65	7.73	6.09	4.75
$\tau[s]$	0.683	2.55	9.69	36.9	145
$\alpha[-]$	0.525	0.435	0.348	0.274	0.213
$\eta[-]$	0.930	0.883	0.846	0.809	0.796
			HLCTS–2D		
$TSPR[-]$	12.4	9.70	7.31	5.43	4.09
$\tau_t[s]$	0.711	2.60	9.60	35.8	137
$\alpha[-]$	0.561	0.437	0.329	0.244	0.184
$\eta[-]$	0.969	0.90	0.838	0.784	0.751

Table 3: *Comparison of traversal algorithms for primary rays for the scene teapot at different resolutions. $TSPR$ – number of traversal steps per ray on average for a specific traversal method. τ – the time for ray shooting only (without shading) for a specific traversal method. α – the ratio between the number of traversal steps for the specific method and the number of traversal steps of the hierarchical traversal algorithm. η – the ratio between the time of the specific method and the time of the hierarchical traversal algorithm.*

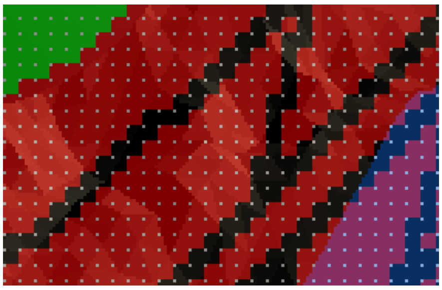
less samples for whole SLCTS, there is lower probability of possible traversal steps reduction for constructed SLCTS. The number of samples needed to construct one SLCTS was always constant, either two (SLCTS–1D) or four (SLCTS–2D). Let us have pixels on a scanline and construct SLCTS–1D for two pixels. Let n be the distance between the two pixels. We can then use the constructed SLCTS for $N' = n-2$ pixels if such a SLCTS exists. The undersampling resolution $n = 5$ pixels is a reasonable compromise. The same holds for SLCTS–2D, when we set the undersampling resolution to 5 × 5 pixels. In general, SLCTS–2D enables to use SLCTS (if SLCTS exists) for $N'' = n.m - 4 - (m-2) - (n-2) = n.m - n - m$ pixels. For high resolution images, n (and m) can be set even to a higher value.

Obviously, the two–dimensional LCTS methods can better exploit coherence properties than one–dimensional ones. The HLCTS reaches better time reduction only for high resolution, mainly because the cost of HLCTS construction is higher than for SLCTS.

(a)



(b)



(c)

Figure 8: *Visualization of the traversal coherence of SLCTS–2D for the scene mount (a) Normal ray tracing. (b) Blended with the color for pixels:* white *– sampling pixels,* red *– the pixels for which exists a common traversal sequence with at least one leaf,* green *– the pixels for which the terminating object was found,* blue *– the pixels for which is known that no object can be hit,* black *– the pixels for which no LCTS was found. (c) The zoomed in part of image (b), where sampling pattern is more visible.*

7. Conclusion and Future Work

In this paper we have studied a new way of exploiting coherence in ray shooting with BSP trees. We tried to avoid as much hierarchical traversal steps within the spatial hierarchy as possible and at the same time to preserve all the advantages of using the hierarchy. Two concepts for longest common traversal sequence have been introduced; SLCTS and HLCTS. The presented techniques enable us to decrease the number of traversal steps for hidden surface removal typically by more than 60%. For high resolution images the reduction of traversal steps is even more remarkable.

There are several topics for future research based on the LCTS concept. It could be used for higher order rays in various global illumination algorithms similarly to beam tracing. It can be applied if the rays of the first order (primary rays) have the same termination object and create a shaft, that is then completely reflected or refracted. Further, other image space sampling patterns suitable for LCTS application than those mentioned in this paper should be studied. The automatic setting of the SLCTS undersampling resolution based on an estimation of scene properties and the use of LCTS in rendering of animation sequences are also possible topics of research.

Acknowledgement

We would like to thank Jan Přikryl and Eduard Gröller from Vienna University of Technology for carefully reading the previous version of this paper and their comments. This project has been partially supported by Czech–Austrian scientific cooperation grant Aktion number 1999/17.

References

1. P.K. Agarwal. and M.Sharir Ray Shooting Amidst Convex Polytopes in Three Dimensions. In proceedings of *4th ACM-SIAM Sympos. Discrete Algorithms*, pp. 260–270, 1993.

2. J. Amanatides. Ray Tracing with Cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, pp. 129–135, July 1984.

3. J. Arvo and D. Kirk. Fast Ray Tracing by Ray Classification. In M. C. Stone, editor, *(SIGGRAPH '87 Proceedings)*, Vol. 21, pp. 55–64, July 1987.

4. J. Arvo and D. Kirk. *A Survey of Ray Tracing Acceleration Techniques*, pp. 201–262, in book An Introduction to Ray Tracing, A. Glassner editor, Academic Press, Redding, 1989.

5. F. Cazals and C. Puech. Bucket-like Space Partitioning Data-structures with Applications to Ray–Tracing. In *13th ACM Symposium on Computational Geometry*, pp. 11–20, Nice, 1997.

6. GOLEM rendering system, `http://www.cgg.cvut.cz/GOLEM`, 1997–2000.

7. J. Genetti, D. Gordon, and G. Williams. Adaptive Supersampling in Object Space using Pyramidal Rays. In *Computer Graphics Forum*, Vol. 17, No. 1, pp. 29–54, 1998.

8. P. Gonzalez and F. Gisbert. Object and Ray Coherence in the Optimization of the Ray Tracing Algorithm. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pp. 264–267, Hannover, Germany, IEEE, NY, June 1998.

9. E. Gröller and W. Purgathofer. Coherence in Computer Graphics. Technical research report TR-186-2-95-04, Institute for Computer Graphics, Technical University Vienna, 1995.

10. E. A. Haines. A Proposal for Standard Graphics Environments. *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, pp. 3–5, Nov. 1987.

11. E. A. Haines and D. P. Greenberg. The Light Buffer: A Ray Tracer Shadow Testing Accelerator. *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, pp. 6–16, Sept. 1986.

12. E. A. Haines and J. 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 2nd Eurographics Workshop on Rendering)*, New York, Springer-Verlag, pp. 122–138, 1994.

13. V. Havran and J. Žára. Evaluation of BSP Properties for Ray–Tracing. In *Proceedings of 12th Spring Conference on Computer Graphics*, pp. 155–162, Budmeřice, June 1997.

14. V. Havran, J. Bittner, and J. Žára : Ray Tracing with Rope Trees. In proceedings of *13th Spring Conference on Computer Graphics*, Budmerice, pp. 130–139, 1998.

15. V. Havran, T. Kopal, J. Bittner, and J. Žára : Fast Robust BSP Tree Traversal Algorithm for Ray Tracing, in *Journal of Graphics Tools*, AK Peters Ltd., Vol. 2, No. 4, pp. 15–24, 1998.

16. V. Havran. Methodology for comparison of practical ray shooting acceleration methods. Under preparation.

17. P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *Computer Graphics (SIGGRAPH'84 Proceedings)*, Vol. 18, No. 3, pp. 119–127, July 1984.

18. F. W. Jansen. Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. van Lierop, editors, *Data Structures for Raster Graphics*, pp. 57–73. Springer-Verlag, New York, 1986.

19. M. Kaplan. *Space-Tracing: A Constant Time Ray-Tracer*, SIGGRAPH'85 *State of the Art in Image Synthesis Seminar Notes*, Vol. 18, pp 149–158. July 1985.

20. K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, Vol. 17, No. 1, pp. 42–51, Jan./Feb. 1997.

21. B. Kwon, D. S. Kim, K.-Y. Chwa, and S. Y. Shin. Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 3, pp. 193–201, July/Sept. 1998.

22. J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, Vol. 6, No. 6, pp. 153–65, 1990.

23. M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. L. Kunii, editor, *Computer Graphics 1987 (Proceedings of CG International '87)*, pp. 303–314. Springer-Verlag, 1987.

24. S. Semwal and H. Kvarnstrom. Directional safe zones & dual extent algorithms for efficient grid traversal. In proceedings of *Graphics Interface 97*, pp. 76–87, 1997.

25. M. Shinya, T. Takahashi, and S. Naito. Principles and applications of pencil tracing. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, pp. 45–54, July 1987.

26. G. Simiakakis. *Accelerating Ray Tracing with Directional Subdivision and Parallel Processing*. PhD thesis, University of East Anglia, October 1995.

27. G. Simiakakis and A. M. Day. Five–dimensional Adaptive Subdivision for Ray Tracing. *Computer Graphics Forum*, Vol. 13, No. 2., pp. 133–140, June 1994.

28. F. Sixta. Data Structures for Ray Tracing. Master Thesis supervised by Vlastimil Havran, Czech Technical University in Prague, February 1999. Available in Czech only.

29. K. Sung and P. Shirley. Ray Tracing with the BSP Tree. In D. Kirk, editor, *Graphics Gems III*, pp. 271–274. Academic Press, San Diego, 1992.

30. L. Szirmay-Kalos and G. Marton. Analysis and Construction of Worst–Case Optimal Ray Shooting algorithms. *Computers and Graphics*, Vol. 22, No. 2, pp. 167–174, Jan. 1998.

31. S. Teller and J. Allex. Frustum casting for progressive, interactive rendering. Technical Report MIT LCS TR–740, MIT, Jan. 1998.

32. M. van der Zwaan, E. Reinhard, and F. W. Jansen. Pyramid clipping for efficient ray traversal. In *Proceedings of the Sixth Eurographics Rendering Workshop*, pp. 1–10, Springer Verlag, New York, 1995.