Binary Space Parititions in Plücker Space[⋆]

David M. Mount[1] and Fan-Tao Pu[2]

[1] Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. `mount@cs.umd.edu`.
[2] Department of Computer Science, University of Maryland, College Park, Maryland. `ftpu@cs.umd.edu`.

1 Introduction

One of the important potential applications of computational geometry is in the field of computer graphics. One challenging computational problem in computer graphics is that of rendering scenes with nearly photographic realism. A major distinction in lighting and shading models in computer graphics is between *local illumination models* and *global illumination models*. Local illumination models are available with most commercial graphics software. In such a model the color of a point on an object is modeled as a function of the local surface properties of the object and its relation to a typically small number of point light sources. The other objects of scene have no effect. In contrast, in global illumination models, the color of a point is determined by considering illumination both from direct light sources as well as indirect lighting from other surfaces in the environment. In some sense, there is no longer a distinction between objects and light sources, since every surface is a potential emitter of (indirect) light. Two physical-based methods dominate the field of global illumination. They are *ray tracing* [8] and *radiosity* [3].

In both of these methods, and important problem that arises is that of determining visibility relationships for a collection of polygonal objects in 3-space. In this paper we propose a data structure to aid in solving this problem. We call this data structure a *visibility map*. Intuitively, a visibility map can be thought of as a sort of computer-generated hologram. Think of this abstract hologram as a piece of transparent plastic. When a viewer looks through the hologram, he sees what appears to be a three-dimensional scene. In particular, this is achieved by associating each ray shot from the eye of the viewer through the hologram with the illumination information associated with this ray. We can remove the viewer and just think of the visibility map as a function that maps rays emanating from the hologram to illumination information.

We show that a visibility map can be represented as a binary space partition tree in projective 5-dimensional space, through the use of a transformation that maps lines in 3-space to points in projective 5-space, called *Plücker coordinates*. An important practical question is how to build such trees and how large they

[⋆] The support of the National Science Foundation under grant CCR–9712379 is gratefully acknowledged.

are. We present an implementation of an algorithm for constructing these trees, and we analyze their size empirically as a function of the number of polygons in the 3-dimensional scene. We also consider methods for pruning the tree and study the effectiveness of these techniques.

1.1 Visibility Maps

Let $\mathbf{P}$ be a set of points and $\mathbf{D}$ be a set of directional vectors. Let $\mathbf{I}$ denote an (application-dependent) domain called the *visibility information*. For example, in rendering applications, the visibility information might be the color of an object, in radiosity applications it might be the radiance of a point. A *visibility map* $\boldsymbol{M}$ is a function

$$\boldsymbol{M} :\ D_{M} \mapsto \mathbf{I} \cup \{\Lambda\},$$

where D_{M}, the domain of $\boldsymbol{M}$ is a subset of $\mathbf{P}\times\mathbf{D}$. Given a point p in space and a direction $\boldsymbol{d}$, $\boldsymbol{M}(p, \boldsymbol{d})$ returns the visibility information arriving at the point p along the direction $-\boldsymbol{d}$. Intuitively, Λ is a special value, called the *null visibility information*, which is returned for rays that hit no objects.

In our application, a visibility map will not necessarily record visibility information coming from every possible object in the scene. A visibility map may have some region of three-dimensional space, called a *scope*, implicitly associated with it. The map records visibility information within the scope. For example, referring to Fig. 1, let $\boldsymbol{M}$ be the visibility map whose domain is the set of rays

Figure omitted.

Fig. 1. Visibility map.

emanating upwards from the shaded disc and whose scope is the region enclosed in the hemisphere. The intersection of each ray with the hemisphere is indicated by a point on the ray. For rays r_1 and r_2, $\boldsymbol{M}(r_1)$ and $\boldsymbol{M}(r_2)$ will return visibility information of objects s_{12} and s_2, respectively. On the other hand, $\boldsymbol{M}(r_4)$ and $\boldsymbol{M}(r_5)$ both return Λ. Although ray r_3 intersects the object s_3, $\boldsymbol{M}(r_3)$ returns Λ because the intersection with object s_3 is outside of $\boldsymbol{M}$'s scope.

1.2 Line Geometry and Plücker Coordinates

Lines in three-dimensional space play an important role in computer graphics. When rendering a patch, we need to know the radiance of every point on the patch. Radiance arrives from every possible direction. To describe radiance in a faithful manner we can define a function that maps each point and each directional vector at the point to a radiance value arriving from this direction. If the patch is planar, it is clear that the domain of this function is isomorphic to the set of lines passing through this patch. A concise and elegant representation of lines in three-dimensional space is important for dealing with such infinite sets of lines. Plücker introduced a method of referencing lines as sets of coordinates, called *Plücker coordinates* [17], which maps a line in projective three-dimensional space to a point in projective five-dimensional space. This creates a new geometry where lines in three-dimensional space are the points of the new geometry.

To define Plücker coordinates, we first define the meet and join operations of two linear subspaces. The *meet* of two subspaces is defined to be their maximum common intersection and the *join* of two subspaces is defined to be the minimum space enclosing both subspaces. For example, in general position, a line meets a plane at a point and a line joins a point into a plane.

Let us consider projective three-dimensional space with homogeneous coordinates. We will present a derivation for line coordinates based on [20] and [10]. Let ℓ be a line and let x and y be two distinct points on ℓ and let ξ and ψ be two distinct planes that contain ℓ. Thus ℓ is the join of x and y and the meet of ξ and ψ. Assume the coordinates for points x, y and planes ξ, ψ are $[x_0, x_1, x_2, x_3]$, $[y_0, y_1, y_2, y_3]$, $[\xi_0, \xi_1, \xi_2, \xi_3]$ and $[\psi_0, \psi_1, \psi_2, \psi_3]$, respectively. Since points x and y both lie on planes ξ and ψ, we have following equations:

$$\begin{cases} \xi_0 x_0 + \xi_1 x_1 + \xi_2 x_2 + \xi_3 x_3 = 0 \\ \xi_0 y_0 + \xi_1 y_1 + \xi_2 y_2 + \xi_3 y_3 = 0 \\ \psi_0 x_0 + \psi_1 x_1 + \psi_2 x_2 + \psi_3 x_3 = 0 \\ \psi_0 y_0 + \psi_1 y_1 + \psi_2 y_2 + \psi_3 y_3 = 0 \end{cases}. \tag{1}$$

Let π_{ij} denote $x_i y_j - x_j y_i$. Eliminating the ξ_0, ξ_1, ξ_2, ξ_3, one at a time, from the first two equations above, we have equations

$$\begin{cases} 0\xi_0 + \pi_{10}\xi_1 + \pi_{20}\xi_2 + \pi_{30}\xi_3 = 0 \\ \pi_{01}\xi_0 + 0\xi_1 + \pi_{21}\xi_2 + \pi_{31}\xi_3 = 0 \\ \pi_{02}\xi_0 + \pi_{12}\xi_1 + 0\xi_2 + \pi_{32}\xi_3 = 0 \\ \pi_{03}\xi_0 + \pi_{13}\xi_1 + \pi_{23}\xi_2 + 0\xi_3 = 0 \end{cases}. \tag{2}$$

There are 16 π_{ij}'s. From the definition of π_{ij}, we know that $\pi_{ij} = -\pi_{ji}$ and $\pi_{ii} = 0$. Therefore, only six different values of π_{ij} determine the remainder. Let these six numbers be π_{01}, π_{02}, π_{03}, π_{23}, π_{31}, and π_{12}. These are called the *Plücker coordinates* of the line ℓ. The six Plücker coordinates derived from different pairs of points of a lines only differ by a non-zero constant factor. Thus the Plücker coordinates of a line are homogeneous coordinates of projective five-dimensional space, which is also called *Plücker space*. We will use the notation

$$\boldsymbol{\pi}(\ell) = [\pi_{01},\, \pi_{02},\, \pi_{03},\, \pi_{23},\, \pi_{31},\, \pi_{12}]$$

to denote the Plücker coordinates of line ℓ. Since the system of homogeneous (linear) equations (2) has a nontrivial solution, the determinant

$$\begin{vmatrix} 0 & \pi_{10} & \pi_{20} & \pi_{30} \\ \pi_{01} & 0 & \pi_{21} & \pi_{31} \\ \pi_{02} & \pi_{12} & 0 & \pi_{32} \\ \pi_{03} & \pi_{13} & \pi_{23} & 0 \end{vmatrix}$$

vanishes. This gives the following constraint on the π_{ij}'s,

$$\pi_{01}\pi_{23} + \pi_{02}\pi_{31} + \pi_{03}\pi_{12} = 0. \tag{3}$$

The set of points satisfying this equation are said to lie on the *Grassmann manifold*.

Define a binary operator, *cross product* [9] ($\times$), on two Plücker points $\boldsymbol{\pi}(\ell_1)$ and $\boldsymbol{\pi}(\ell_2)$ as

$$\boldsymbol{\pi}(\ell_1) \times \boldsymbol{\pi}(\ell_2) = \pi^1_{01}\pi^2_{23} + \pi^1_{02}\pi^2_{31} + \pi^1_{03}\pi^2_{12} + \pi^1_{23}\pi^2_{01} + \pi^1_{31}\pi^2_{02} + \pi^1_{12}\pi^2_{03}. \tag{4}$$

The concept of *directed line*, line with one direction, is used to resolve the ambiguity of two opposite directions a line represents. The homogeneity condition for Plücker coordinates is modified to allow multiplication by any strictly positive constant. This operator returns 0 if ℓ_1 and ℓ_2 are incident, and otherwise its sign can be used to determine the relative orientation of the (directed) lines.

1.3 Binary Space Partitions

The *binary space partition* (BSP) tree [6] is an example of a data structure based on a recursive hierarchical space partitioning. A BSP tree is a binary tree, which encodes a hierarchical subdivision of d-dimensional space. The cell associated with each node v of a BSP tree is a convex polytope R_v. Each internal node v in a BSP tree is associated with a $(d-1)$-dimensional *cutting hyperplane* H_v. Let H_v^+ and H_v^- denote the two halfspaces introduced by hyperplane H_v. Then cutting hyperplane H_v cuts polytope R_v into two polytopes $R_v \cap H_v^+$ and $R_v \cap H_v^-$ which are associated with the left and right subtrees of node v, respectively.

The splitting procedure is performed recursively at each node until either all objects are separated or some application-dependent termination conditions are satisfied. Such termination conditions may be designed to avoid an excessive fragmentation of object or to bound the maximum tree depth. A node v in a BSP tree stores the objects that intersect the interior of the polytope R_v. If the cutting hyperplane H_v intersects an object, the object is split by H_v and each portion will be stored in the corresponding subtree.

There is no special rule for selecting the cutting hyperplanes for a BSP tree. However, the choice of cutting hyperplanes affects the size and maximum depth of the BSP tree and the number of object fragments that arise. Several works [1, 2, 4, 14, 15] have been devoted to the problem of selecting the cutting hyperplanes so as to minimize the complexity of the resulting BSP tree. For example, if there is a facet of an object lying on a hyperplane which does not intersect the interior of any other objects of this node, then this is a good candidate for the cutting hyperplane.

2 Data Structures for Visibility Maps

Before discussing the representation of visibility maps, we begin by discussing how to represent complex functions of line-space by simpler piecewise functions.

Henceforth, we assume that lines in three-dimensional space are represented as directed lines using signed Plücker coordinates.

Let $\boldsymbol{\pi}(\ell) = [\ell_{01}, \ell_{02}, \ell_{03}, \ell_{23}, \ell_{31}, \ell_{12}]$ denote the Plücker coordinates of a directed line associated with some ray in the domain of the visibility map. Let $\boldsymbol{\pi}(e) = [e_{01}, e_{02}, e_{03}, e_{23}, e_{31}, e_{12}]$ denote the Plücker coordinates of the line supporting an edge of some convex patch in the scene. For a discontinuity to occur when ℓ intersects e, it must be that these two directed lines are incident, implying that

$$(\boldsymbol{\pi}(\ell) \times \boldsymbol{\pi}(e)) = \ell_{01}e_{23} + \ell_{02}e_{31} + \ell_{03}e_{12} + \ell_{23}e_{01} + \ell_{31}e_{02} + \ell_{12}e_{03} = 0.$$

This is a linear equation in $\boldsymbol{\pi}(\ell)$. We can use the sign of the orientation to determine whether ℓ passes to the left or right of e. Given a convex polygonal patch, P, let $\ell_{e_1}, \ell_{e_2}, \ldots, \ell_{e_m}$ denote the directed lines supporting the counterclockwise oriented edges of the patch. Then ℓ intersects, or *stabs*, P if all of these orientations are of the same sign, that is, if

$$\begin{array}{cc} \boldsymbol{\pi}(\ell) \times \boldsymbol{\pi}(\ell_{e_i}) \geq 0 & \text{for } 1 \leq i \leq m \\ \text{or} & \\ \boldsymbol{\pi}(\ell) \times \boldsymbol{\pi}(\ell_{e_i}) \leq 0 & \text{for } 1 \leq i \leq m. \end{array}$$

Thus the set of lines in three-dimensional space that stab a convex polygonal patch from one side or the other is a closed polyhedron in the projective five-dimensional space or Plücker space. See Fig. 2.

Figure omitted.

Fig. 2. The orientation between a patch and its stabbing line.

2.1 Plücker Space Partition Trees

Suppose that we want to represent the visibility map associated with some convex polygonal patch P. Let us assume that we are interested in visibility information arriving from only one side of P, indicated by an outward pointing normal vector. Take the set of directional vectors for the map to be the set of vectors whose angle with respect to P's normal vector is at most $\pi/2$. Each point on P and each directional vector corresponds to a directed line passing through the point and having this direction. As the point and/or direction vary continuously, the visibility information varies continuously as well, until the visibility ray strikes a different object of the scene. Thus discontinuities arise only when the line supporting the visibility ray intersects an object edge. Hence the visibility map is a piecewise continuous function where discontinuities occur along hyperplanes in Plücker space. We can think of this function as a subdivision of five-dimensional Plücker space, where each cell of the subdivision is associated with one visibility information function. We will concentrate on showing how to represent this subdivision through the use of a hierarchical space partition, which we call a *Plücker space partition tree* or *PSP tree* for short.

Recall the definition of a binary space partition from the Section 1. The space to be partitioned is Plücker space. Each node is implicitly associated with a convex polyhedral region of space. The root of the PSP tree implicitly represents all of Plücker space. (However we will modify this below.) Each internal node of the PSP tree is associated with a directed *splitting line* in three-dimensional space, or equivalently, a four-dimensional splitting hyperplane in the Plücker space. Each internal node has two children, one for lines positively oriented with respect to the splitting line and the other negatively oriented.

We will be storing visibility maps for axis-aligned rectangles in three-dimensional space. Each PSP tree will be implicitly associated with the set of directed lines passing through an axis-aligned rectangular "window" in three-dimensional space. Let ℓ_1, ℓ_2, ℓ_3, and ℓ_4 denote the directed lines supporting the edges of this rectangle, oriented counterclockwise around the rectangle so that the directed lines passing through the rectangle are positively oriented with respect to these lines. These four lines correspond to four halfspaces in Plücker space. Since we will only be interested in directed lines that lie in this region of Plücker space, rather than storing these four lines in the PSP tree (e.g. as the top four nodes), we store them separately as a *header* for the PSP tree.

The algorithm for constructing the rest of the Plücker space partition tree is as follows. Starting with the root, we insert each of the lines that bounds each of the three-dimensional patches of the scene as splitting lines into the PSP tree. For example, if we assume that all of the polygonal patches from the scene are convex, then we could insert the directed lines supporting the edges of each polygonal patch into the PSP tree one by one. The insertion of each splitting line corresponds to the insertion of a four-dimensional hyperplane into the tree. This is done done by an appropriate generalization of the standard insertion scheme for standard BSP trees (see, for example, Patterson and Yao [14]) but in dimension five. Each leaf of the final tree corresponds to a set of directed lines that are equally oriented with respect to all the edges of the patches and thus stab the same set of polygonal patches. Hence, all the lines in this cell share the same visibility information function.

General Structure Each node in a PSP tree, called a *PSPNode*, denotes a region in Plücker space. This region may be further subdivided by a Plücker hyperplane into two subregions. The subregion resides on the positive side of the cutting hyperplane is represented by its positive child and the one on the negative side is represented by its negative child. A node without a cutting hyperplane is a leaf node. Every internal node in this data structure has two children. Since for rendering we are only interested in the Plücker regions represented by leaf nodes of the tree, we adapted the idea of a *threaded tree* [11] by chaining all leaf nodes into a doubly linked list. This list provides a direct way to enumerate these leaf nodes.

Figure omitted.

Fig. 3. Structure of Plücker space partition tree.

Insertion A Plücker space partition tree is constructed by inserting the polygons of the scene one-by-one. Each polygon is inserted into a Plücker space partition tree by inserting all of the supporting lines for its edges. The basic update of the Plücker space partition tree is the insertion of a line (as a Plücker hyperplane) into the tree. Recall that each node of the Plücker space partition tree represents a convex polyhedral region of Plücker space. A line is inserted recursively into a node's two children provided that the line's corresponding Plücker hyperplane cuts the region represented by the node. The recursion stops when the node is a leaf. This line (the Plücker hyperplane) becomes the leaf node's cutting plane and two new leaf nodes are created as its children. This algorithm is given in Fig. 4.

```
Insert(history,iplane)
    if (is_leaf)
        make iplane as its cutting_plane
        create two leaf nodes as its children
    else
        PosPolytope = Polytope(history, halfspace(cutting_plane,+1))
        NegPolytope = Polytope(history, halfspace(cutting_plane,-1))
        if (intersect(PosPolytope, iplane))
            new_history = union(history, halfspace(cutting_plane,+1))
            Insert(new_history, iplane)
        endif
        if (intersect(NegPolytope, iplane))
            new_history = union(history, halfspace(cutting_plane,-1))
            Insert(new_history, iplane)
        endif
    endif
end Insert
```

Fig. 4. Insertion.

3 Trimming the Plücker Space Partition Tree

The size of a Plücker space partition tree grows rapidly as a function of the number of triangles. See Section 4.3. In fact, it is too large to deal practically with any non-trivial scene. We investigate methods for eliminating redundant nodes from the tree. We consider two simple ways in which redundant nodes may arise: (1) Two siblings encode the same visibility information. (2) Leaf nodes do not encode any line. The former suggests merging sibling leaf nodes having the same visibility information. The latter suggests deleting leaf nodes which do not interest Grassmann manifold. They are described next.

3.1 Merging by Constant Visibility Information

When two sibling leaf nodes carry the same *constant visibility information*, they can be merged by deleting them and making their parent a new leaf node. Under the assumption we have made earlier, constant visibility information means either transparency (no polygon is stabbed by the lines of the node) or the same single polygon is stabbed from the same direction. If sibling leaf nodes stab the same set of polygons and the size of the set is more than two, then they cannot be merged because the visible polygon in the set is view-dependent and may differ.

The merging process is performed by a recursive postorder traversal of the Plücker space partition tree. The recursion stops at a leaf and returns the number of stabbing polygons. Each internal node compares the values returned from its two children. If both children are leaves and either they stab no polygon, or they both stab the same single polygon then the internal node merges information of its children and deletes them. See Fig. 5.

```
ConstVisInfoMerge(node)
    if (is_leaf(node))
        return number of stabbed polygon
    else
        num_pos_stab = ConstVisInfoMerge(positive_child)
        num_neg_stab = ConstVisInfoMerge(negative_child)
        if ((num_pos_stab==0)&&(num_neg_stab==0))
            merge children
        else if ((num_pos_stab==1)&&(num_neg_stab==1))
            if the two stabbing polygons are the same then
                merge children
            endif
        endif
    endif
End
```

Fig. 5. Constant visibility information merging algorithm.

3.2 Merging by Empty Grassmann Intersection

Recall that a Plücker polyhedron does not represent any line in three-dimensional space if it does not intersect the Grassmann manifold. Any leaf node whose associated region does not intersect the Grassmann manifold may be deleted as redundant.

To test whether a given node in the Plücker space partition tree intersects the Grassmann manifold we consider the polyhedron it represents. In [18] we showed that such a polyhedron does not intersect the Grassmann manifold if and only if its one-dimensional boundary (1-skeleton) does not intersect the manifold. Therefore, the emptiness test involves the following steps:

1. Enumerate the vertices of the Plücker region in projective five-dimensional space as well as their adjacency relation,
2. Interpolate every pair of adjacent vertices to construct the 1-skeleton, and
3. Test whether each edge of the 1-skeleton intersects the Grassmann manifold, and return empty-intersection if no intersection is found.

The basic structure of the merging algorithm is similar to the one for the constant visibility information merge, see Fig. 6. We discuss enumeration below.

```
EmptyGManifoldMerge(node)
    if (is_leaf(node))
        return EmptinessTest(node)
    else
        pos_emptiness = EmptyGManifoldMerge(positive_child)
        neg_emptiness = EmptyGManifoldMerge(negative_child)
        if ((pos_emptiness==EMPTY) && (neg_emptiness==EMPTY))
            merge children
            return EMPTY
        else if ((pos_emptiness==EMPTY) || (neg_emptiness==EMPTY))
            merge children
            return NON_EMPTY
        else
            return NON_EMPTY
        endif
    endif
End
```

Fig. 6. Trimming empty Grassmann intersection algorithm.

3.3 Auxiliary Routines

Two nontrivial tasks have been ignored in the above description. One is used when inserting a hyperplane into a Plücker space partition tree and another is used when testing whether a node in Plücker space partition tree does not intersect the Grassmann manifold. They are stated more formally below. Let C be a given Plücker polyhedron defined as the intersection of a set of Plücker halfspaces. Note that the polyhedron C is a cone in Euclidean six-dimensional space.

1. For a Plücker hyperplane h, determine whether h intersects C. (See Section 2.1.)
2. What are the extremal rays of cone C? (See Section 3.2)

The first is a linear-programming problem and the second is a polytope enumeration problem.

A convex polyhedron can be defined in two ways. It can be described either as an intersection of a set of halfspaces, or as a convex combination of its vertices and/or extremal rays. Note that extremal rays arise if the polyhedron is not closed, i.e., unbounded. The former is called the *H-representation* and the latter is called the *V-representation* [7] of the polyhedron. An application program called *cdd/cdd+* by Fukuda [7] utilizes linear programming techniques to implement an algorithm called the *double description method* [13], which converts one representation to another for a given polyhedron. We have adapted and modified the routines in the cdd/cdd+ program for solving the following tasks:

1. Find the existence of a feasible region (except the origin) of a given set of homogeneous six-dimensional inequalities, and
2. Find the extremal rays of a cone defined by a set of homogeneous six-dimensional inequalities.

Note that we perform computation for projective five-dimensional space in Euclidean six-dimensional space.

4 Experiments

To explore various properties of the Plücker space partition tree we ran a number of experiments which generates Plücker space partition trees from randomly generated inputs and than measure these properties. Each experiments involves generating 100 different random inputs, each consisting of a set of nonintersecting triangles in three-dimensional space. The generating program is described in Section 4.1. In Section 4.2, we discuss the result of these experiments.

4.1 Random Triangle Generator

A small program which generates triangles in a bounding box randomly is used as data generator for the experiment. This program is given the number of triangles n, and outputs n nonintersecting triangles of various sizes distributed through out a given bounding box.

To generate nonintersecting triangles we first decompose space hierarchically using a k-d tree [19]. The k-d tree splitting rule guides this distributing. A recursive algorithm directs the process. Let n be the number of triangles to be generated in a bounding box b. If $n > 1$ then an axis is chosen at random. A plane h perpendicular to this axis is then randomly generated to split the bounding box b into boxes b' and b''. The number n is partitioned according to the ratio between b' and b'' into n' and n'' ($n = n' + n''$), respectively. The process is recursively applied separately to box b' with number of triangle n', and to box b'' with number of triangles n''. If $n = 1$ then a triangle is generated by randomly picking three points on the walls of box b.

4.2 Results

Plücker Space Partition Trees without Pruning To explore the properties of the Plücker space partition tree, we build Plücker space partition trees for scenes consisting of a number of triangles ranging from 1 to 14. Each input size is tested over 100 randomly generated scenes. We presents the plots for the size of tree (Fig. 7), the height of tree (Fig. 8) and the time (measure in seconds) for building the tree (Fig. 9). We use the average number of leaf nodes for showing the size of Plücker space partition tree. Leaf nodes are further classified as being either transparent (stabbing no triangles) or not transparent as denoted by "Trans." and "Non-Trans." in the legend of Fig. 7.

Plücker Space Partition Trees with Pruning The effect of trimming Plücker space partition tree is presented in Fig. 10 in regular scale and Fig. 11 in logarithm scale for comparing the average number of nodes in a tree when without trimming, with merging by constant visibility information only, with merging by empty Grassmann intersection only, or with both merging heuristics. They are denoted as "No TRIM", "C TRIM", "G TRIM" and "CG TRIM" in the legend box, respectively.

4.3 Analysis

The Plücker space partition tree is a variation of binary space partition tree in higher (five) dimensions. We know of no nontrivial complexity analysis of binary space partition trees in these dimensions. The Plücker space partition tree defines a subdivision of space which is a coarsening of the arrangement of the set of hyperplanes of the five-dimensional space generated by all the lines that were inserted into the tree. It follows from standard results in combinatorial geometry that, the worst case complexity of Plücker space partition tree is $O(n^5)$ [5] where n is the number of lines. McKenna and O'Rourke [12] established that the number of distinct isotopy classes for n given lines is $O(n^4\alpha(n))$. The number of leaves in the Plücker space partition tree can generally exceed this amount, since some leaves of the tree might not intersect the Grassmann manifold, and hence may not correspond to any isotopy class. However, after pruning away leaf cells that do not intersect the Grassmann manifold, this bound should apply in our case. Pellegrini and Shor [16] showed that for a given set of convex polyhedra, the complexity of lines that stabs these polyhedra, a subset of isotopy classes induced by these lines, is $O(n^3 2^{c\sqrt{\log n}})$, where n is the number of facets of the given set of polyhedra and c is a constant. However, this bound does not apply immediately, because the Plücker space partition tree computes leaf cells that might not intersect any of the objects.

It follows that the size of a Plücker space partition tree is at most $O(n^5)$, where n is the number of input triangles. Let $s(n)$ be the size of a Plücker space partition tree of input size n. We conjecture that the size of the Plücker space partition tree is of the following form

$$s(n) \approx an^c \qquad \text{or equivalently} \qquad \log s \approx c \log n + \log a,$$

for some constants a and c. Based on this conjecture, the values of constants a and c can be estimated from our experimental results by fitting a line to a log-log scale plot of tree-size versus n.

We consider the un-trimmed Plücker space partition tree first. Figure 12 (a) shows the curve obtained by connecting the 14 points whose x-coordinate is the logarithm of the number of triangles and y-coordinate is the logarithm of the average tree size. We deleted the leftmost two points (since low values are less likely to provide good estimation of asymptotic values) and applied a least squares line fit. See Fig. 12 (b). We found that the line of fit is $(\ln s) = 5.17(\ln n) - 1.16$, i.e.,

$$s(n) \approx 0.315 n^{5.17}. \tag{5}$$

The exponent is quite close to 5, which suggests that the $O(n^5)$ upper bound may be tight. The fact that the exponent exceeds 5 is likely due to the fact that the input sizes are not large enough to accurately gauge asymptotic growth rate. Of course, the input sizes are quite small, and so our extrapolations should be taken with a grain of salt. Next is the trimmed (by both heuristic methods) Plücker space partition tree. Figure 13 (a) presents the curve by connecting points of the average trimmed size of Plücker space partition tree versus the number of triangles in log-log scale. Again, after deleting the leftmost two points in this figure, least square fit returns the line, see Fig. 13 (b), $(\ln s) = 4.31(\ln n) - 0.129$, i.e.,

$$s(n) = 0.879 n^{4.31}. \tag{6}$$

Again, extrapolations to asymptotic bounds is risky with such small input sizes, but it does bear a similarity to the $O(n^4\alpha(n))$ bound of McKenna and O'Rourke.

In summary, (5) and (6) suggest the complexities for the un-trimmed and trimmed Plücker space partition tree are roughly $O(n^{5.17})$ and $O(n^{4.31})$, respectively. These are similar the upper bounds on the complexity of an arrangement in five-dimensional space and the complexity of isotopy classes, respectively.

References

1. Pankaj K. Agarwal, Leonidas J. Guibas, T. M. Murali, and Jeffrey Scott Vitter. Cylindrical static and kinetic binary space partitions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 39–48, 1997.

2. Pankaj K. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 382–384, 1997.
3. Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, Cambridge, Mass., 1993.
4. M. de Berg, M. de Groot, and M. Overmars. New results on binary space partitions in the plane. *Comput. Geom. Theory Appl.*, 8:317–333, 1997.
5. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.
6. H. Fuchs, M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, July 1980. Proc. SIGGRAPH '80.
7. Komei Fukuda and Alain Prodon. Double description method revisited. *Lecture Notes in Computer Science*, 1120, 1996. The URL for cdd+ package is: http://www.ifor.math.ethz.ch/ifor/staff/fukuda/cdd_home/cdd.html.
8. Andrew S. Glassner, editor. *An Introduction to Ray Tracing.* Academic Press, San Diego, CA, 1989.
9. Václav Hlavatý. *Differential Line Geometry.* P. Noordhoff Ltd., Groningen, Holland, 1953. Translated by Harry Levy.
10. C. M. Jessop. *A Treatise of Line Complex.* Combridge, Combridge, England, 1903.
11. D. E. Knuth. *The Art of Computer Programming.* Addison Wesley, second edition, 1978.
12. M. McKenna and J. O'Rourke. Arrangements of lines in 3-space: a data structure with applications. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 371–380, 1988.
13. T. S. Motzkin, H. Raiffa, G. L. Tompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tuker, editors, *Contributions to theory of games*, volume 2. Princeton University Press, Princeton, RI, 1953.
14. M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5:485–503, 1990.
15. M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.
16. M. Pellegrini and P. Shor. Finding stabbing lines in 3-space. *Discrete Comput. Geom.*, 8:191–208, 1992.
17. J. Plücker. *Neue Geometrie des Raumes.* Leipzig, 1868.
18. F.-T. Pu. *Data structures for global illumination computation and visibility queries in 3-space.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, March 1998.
19. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.
20. D. M. Y. Sommerville. *Analytic Geometry of Three Dimensions.* Cambridge University Press, Cambridge, England, 1934.

Figure omitted.

Fig. 7. Leaf nodes.

Figure omitted.

Fig. 8. The height of Plücker space partition tree.

Figure omitted.

Fig. 9. Building time of Plücker space partition tree.

Figure omitted.

Fig. 10. The average tree size for different trimming.

Figure omitted.

Fig. 11. The average tree size for different trimming (log scale).

Figure omitted.

Fig. 12. Average size of un-trimmed tree in log-log scale.

Figure omitted.

Fig. 13. Average size of trimmed tree in log-log scale.