

# Accelerating Ray Tracing using Constrained Tetrahedralizations

Ares Lagae & Philip Dutré<sup>†</sup>  
Department of Computer Science  
Katholieke Universiteit Leuven

---

## Abstract

*In this paper we introduce the constrained tetrahedralization as a new acceleration structure for ray tracing. A constrained tetrahedralization of a scene is a tetrahedralization that respects the faces of the scene geometry. The closest intersection of a ray with a scene is found by traversing this tetrahedralization along the ray, one tetrahedron at a time. We show that constrained tetrahedralizations are a viable alternative to current acceleration structures, and that they have a number of unique properties that set them apart from other acceleration structures: constrained tetrahedralizations are not hierarchical yet adaptive; the complexity of traversing them is a function of local geometric complexity rather than global geometric complexity; constrained tetrahedralizations support deforming geometry without any effort; and they have the potential to unify several data structures currently used in global illumination.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

---

## 1 Introduction

Tracing a ray through a scene and finding the closest intersection with the scene geometry is a fundamental operation in computer graphics. During the last two decades, significant efforts have been made to accelerate this operation, with interactive ray tracing as one of the major driving forces. At the heart of a fast method for intersecting a scene with a ray lies the acceleration structure. Many different acceleration structures exist, but research has focused almost exclusively on a few well-trying and well-established techniques: regular and hierarchical grids, bounding volume hierarchies and kd-trees. For an overview we refer to [Gla89, Hav00]. Spectacular advances have been made, which have contributed significantly to making interactive ray tracing a possibility [WSBW01, Wal04, RSH05]. However, despite the success of these acceleration structures, several problems remain open. Handling deforming and dynamic geometry still poses significant challenges [WMG\*07], and the local vs. global complexity of acceleration structures is still not entirely understood. One therefore wonders whether other acceleration structures, that leave the beaten path of efficient grids, bounding volume hierarchies and kd-trees, can provide viable alternatives.

Next to computer graphics, the ray shooting problem is

also studied in computational geometry. For an overview we refer to [dBCvKO08]. Ray shooting queries against a large collection of polyhedra are answered by tracing the ray through a simplicial complex such as a constrained tetrahedralization. This is a well-known technique, see e.g. the chapter *Ray shooting and lines in space* by Pellegrini in *Handbook of Discrete and Computational Geometry* [Pel97]. However, relevant work in computational geometry is usually theoretical, and practical implementations and experimental results are typically not available.

In this paper we explore the idea of accelerating the operation of intersecting a scene with a ray using constrained tetrahedralizations. A constrained tetrahedralization of a scene is a tetrahedralization that respects the faces of the scene geometry. The closest intersection of a ray with a scene is found by traversing this tetrahedralization along the ray, one tetrahedron at a time, until a constrained face is encountered. This is illustrated in figure 1. We show that constrained tetrahedralizations are a viable alternative to state-of-the-art acceleration structures, such as kd-trees, and that constrained tetrahedralizations have a number of interesting and unique properties that set them apart from traditional acceleration structures. Constrained tetrahedralizations are not hierarchical yet adaptive; the complexity of traversing them is a function of local geometric complexity rather than global geometric complexity; constrained tetrahedralizations support deforming geometry without any effort; and they have

---

<sup>†</sup> e-mail: {ares.lagae, philip.dutré}@cs.kuleuven.be

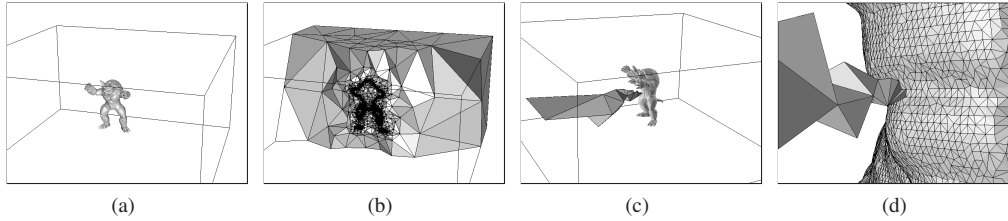


Figure 1: Accelerating ray tracing using constrained tetrahedralizations. (a) A scene consisting of the *Armadillo* model. (b) A tetrahedralization of space that respects the geometry of the scene. (c) A ray is traced through the tetrahedralization. (d) A constrained face is hit.

the potential to unify several data structures currently used in global illumination. Although constrained tetrahedralizations are not a silver bullet, and although they are in general not yet faster than the most optimized kd-trees, constrained tetrahedralizations offer several new perspectives on acceleration structures for ray tracing and deserve attention.

## 2 Constrained Tetrahedralizations

Triangulations are fundamental geometric structures in computational geometry. It is often useful to require that a triangulation contains specific vertices or edges. Such triangulations are called *constrained triangulations*. Our acceleration structure is based on *constrained tetrahedralizations*, their three-dimensional counterpart.

### 2.1 Constrained Triangulations

Constrained triangulations are constructed starting from a set of constraints. These constraints consist of a set of vertices and a set of edges that must appear in the triangulation, and are formalized as a *planar straight line graph* (PSLG). A PSLG consists of a set of points and a set of segments, such that the endpoints of each segment in the PSLG are also in the PSLG, and the intersection of two segments in the PSLG is either empty or an endpoint.

We introduce three types of constrained triangulations with properties similar to the Delaunay triangulation. The *conforming Delaunay triangulation* [ET92] adds vertices to the PSLG such that all segments in the PSLG are also segments of the Delaunay triangulation. The *constrained Delaunay triangulation* [Che89, She07] does not add vertices but relaxes the Delaunay criterion. The constrained Delaunay triangulation is not strictly a Delaunay triangulation, but retains many of its desirable properties. The *quality Delaunay triangulation* [She98b] is obtained by refining a constrained Delaunay triangulation according to a quality criterion based on the angles, the area, or the radius-edge ratio of the triangles. Quality Delaunay triangulations are frequently used in finite element methods.

Figure 2 shows a PSLG and its conforming Delaunay triangulation, constrained Delaunay triangulation, and quality Delaunay triangulation.

### 2.2 Constrained Tetrahedralizations

Constrained tetrahedralizations are also constructed starting from a set of constraints. These constraints consist of a set of vertices, edges and faces that must appear in the tetrahe-

dralization, and are formalized as a *piecewise linear complex* (PLC) [MTT\*96]. A PLC consists of a set of 0D simplices (vertices), 1D simplices (segments) and 2D simplices (faces), such that the simplices corresponding to the boundary of each simplex in the PLC are also in the PLC, and that the intersection of two simplices of the PLC is either empty or the union of lower dimensional simplices also in the PLC.

The three constrained triangulations introduced in the previous subsection have geometric equivalents in three dimensions. The *conforming Delaunay tetrahedralization* [ET92] adds vertices to the PLC such that all segments and faces in the PLC are also segments and faces of the Delaunay tetrahedralization. The *constrained Delaunay tetrahedralization* [Che89, She07] does not exist for every PLC (a famous example is the Schönhardt polyhedron [Sch28]), and deciding whether a polyhedron is tetrahedralizable is NP-complete [RS92]. However, a condition guaranteeing the existence of the constrained Delaunay tetrahedralizations is available, and that condition can be enforced by inserting additional vertices into the PLC [She98a]. The *quality Delaunay tetrahedralization* [She98b] is obtained by refining a constrained Delaunay tetrahedralization and is frequently used in finite element methods.

Figure 3 shows the constrained Delaunay tetrahedralization and quality Delaunay tetrahedralization of a PLC corresponding to the *Armadillo* model.

## 3 Constructing Constrained Tetrahedralizations

We construct constrained tetrahedralizations from geometric models in two steps. First, a piecewise linear complex is created from the geometric model. Then, a constrained tetrahedralization is constructed from the piecewise linear complex.

### 3.1 Constructing Piecewise Linear Complexes from Geometric Models

Piecewise linear complexes can be seen as very general geometry representations, and are able to represent arbitrary polygons, holes, and non-manifold geometry. The only requirement is that polygons properly intersect. More specifically, polygons are only allowed to touch at shared edges or shared vertices. This is a necessary condition because tetrahedralizations do not allow intersecting faces.

Many geometry representations in computer graphics satisfy the requirements of piecewise linear complexes: manifold geometry, geometry modeled using constructive solid geometry operations, and geometry obtained by triangulat-

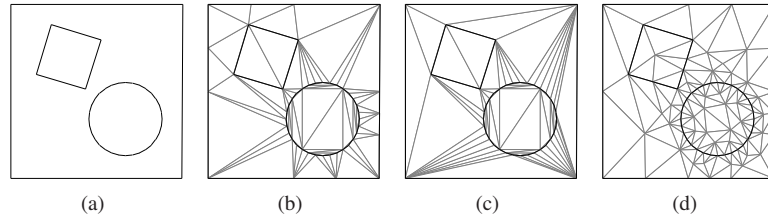


Figure 2: Constrained triangulations. (a) A planar straight line graph representing a room containing a square and a tessellated circle. (b) The conforming Delaunay triangulation. (c) The constrained Delaunay triangulation. (d) A quality Delaunay triangulation.

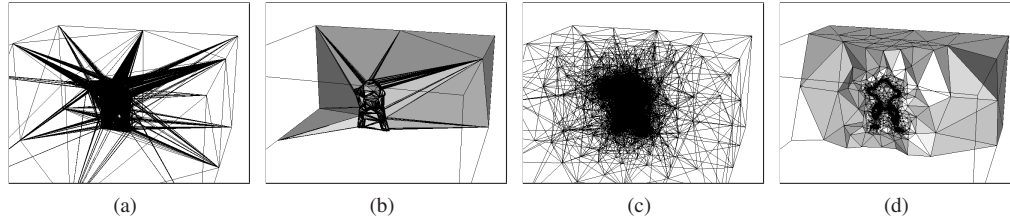


Figure 3: Constrained tetrahedralizations. The constrained Delaunay tetrahedralization (a, b) and the quality Delaunay tetrahedralization (c, d) of a piecewise linear complex representing the *Armadillo* model. Cutaway views expose the interior of the constrained tetrahedralizations.

ing higher order primitives such as NURBS or subdivision surfaces. Polygon soup, an unstructured collection of arbitrary polygons, does not satisfy these requirements, because polygons most probably will intersect. A piecewise linear complex can be constructed from polygon soup by eliminating all self-intersections in the polygon soup by triangulating polygons with respect to their intersection.

### 3.2 Constructing Constrained Tetrahedralizations from Piecewise Linear Complexes

To compute constrained Delaunay tetrahedralizations, we use the method by Si and Gärtner [SG05]. This method works by updating the piecewise linear complex into another geometrically equivalent piecewise linear complex that is guaranteed to have a constrained Delaunay tetrahedralization, and subsequently recovers the missing faces using a cavity re-tetrahedralization algorithm. For computing quality Delaunay tetrahedralizations, we use the method by Si [Si06a], which computes a constrained Delaunay tetrahedralization and then generates an isotropic mesh using a sizing function automatically derived from the geometric data. Both of these methods are implemented in the *TetGen* software package [Si06b], of which the main goal is to generate suitable meshes for solving partial differential equations by finite element methods.

For the remainder of the paper we use constrained Delaunay tetrahedralizations and quality Delaunay tetrahedralizations. We do not use conforming Delaunay tetrahedralizations since they are similar to constrained Delaunay tetrahedralizations, except that they are strictly Delaunay. Since we do not rely on specific properties of these tetrahedralizations, we define a constrained tetrahedralization as any tetrahedralization for which the constrained faces are geometrically equivalent with a given geometric model.

## 4 Traversing Constrained Tetrahedralizations

Traversing a constrained tetrahedralization with a ray is done by locating the tetrahedron containing the ray origin, and traversing the tetrahedralization along the ray, one tetrahedron at a time, until a constrained face is encountered. This face is the closest face of the corresponding geometric model hit by the ray. Figures 4 and 6 illustrate ray traversal in two and three dimensions.

### 4.1 Locating the Ray Origin

Several methods can be used for locating the ray origin. For now, we assume that the ray origin is situated within the domain of the tetrahedralization, but we will later relax this assumption (see subsection 6.4).

The simplest method is a linear search over all tetrahedra using a point-in-tetrahedron test. This test substitutes the ray origin into the plane equations of the faces of the tetrahedron, and uses the sign of the results to determine whether the point is inside the tetrahedron. More efficient methods for point location in triangulations and tetrahedralizations are available both in computational geometry (e.g. monotone subdivisions) and in computer graphics (e.g. grids).

In general, we will avoid locating the ray origin by exploiting ray connectivity, and only locate the position of the camera once (see subsection 4.3). Therefore, we use a simple method such as a linear search or a coarse grid. This already suffices for reducing ray origin location to a fraction of the total ray tracing time (see subsection 5.3).

### 4.2 Traversing the Tetrahedralization

After the tetrahedron containing the ray origin is located, the tetrahedralization is traversed, one tetrahedron at a time, until a constrained face is hit. The next tetrahedron traversed by the ray is obtained by determining the face where the ray exits the current tetrahedron, given the face where the ray enters the current tetrahedron. This can easily be done

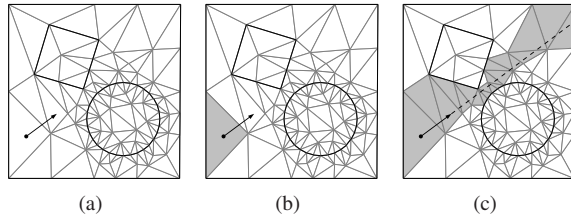


Figure 4: Ray traversal in 2D. (a) A constrained triangulation of the scene of figure 2(a) and a ray. (b) The triangle containing the ray origin is located. (c) The triangulation is traversed along the ray. Ray traversal stops when a constrained edge is encountered.

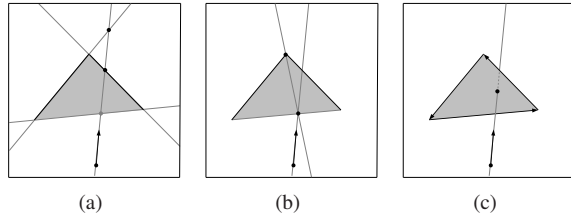


Figure 5: Triangle traversal in 2D. (a) The 2D equivalent of the plane intersection method. (b) The 2D equivalent of the half space classification method. (c) The 3D method based on Plücker coordinates or scalar triple products.

using several ray-triangle intersection tests or a full ray-tetrahedron intersection test [PT03]. However, we present several more efficient alternatives.

**Plane Intersections** The planes of a tetrahedron intersect the line corresponding to the ray at four different points. The exit face corresponds to the smallest ray parameter larger than the ray parameter of the entry face. The two-dimensional equivalent of this method is illustrated in figure 5(a). A special case occurs for the tetrahedron containing the ray origin, for which the entry face is not available. However, since the ray origin is inside the tetrahedron, the smallest positive ray parameter indicates the exit face. A similar method was used by Garrity [Gar90] (see subsection 6.8).

**Half Space Classification** The half space classification method has a two-dimensional version, for traversing a triangulation, and a three-dimensional version. In two dimensions, the exit edge can be determined by classifying the ray direction against the normal or direction vector of a single line, which is determined by two vertices. The first vertex is the intersection point of the ray and the entry edge. The second vertex is the vertex opposite the entry edge. This method is illustrated in figure 5(b). In three dimensions, the exit face can be determined by classifying the ray direction against the normals of three planes. Each of these planes is determined by three points. Two of these points are common for all planes: the entry point of the ray, and the vertex opposite to the entry face. The third point is one of the other vertices of the tetrahedron.

**Plücker Coordinates** A directed line in three-dimensional space can be expressed by six Plücker coordinates [TH99]. The permuted inner product of two pairs of Plücker coordi-

nates determines the relative orientation of the lines corresponding to these coordinates. More specifically, the sign of the permuted inner product determines whether their relative orientation is clockwise or counterclockwise. A ray intersects a triangle if the relative orientation of the ray direction and each of the three consistently oriented directed edges is the same. This is illustrated in figure 5(c). A ray-tetrahedron intersection test can be computed as four ray-triangle intersection tests [PT03], requiring 12 permuted inner products. However, the permuted inner product flips sign if the direction of one of the lines is reversed. This means that 6 permuted inner products can be reused.

**Scalar Triple Products** The scalar triple product of three vectors is defined as the dot product of the first vector with the cross product of the second and third vector. The scalar triple product is equal to the signed volume of the parallelepiped determined by the vectors. Scalar triple products and Plücker coordinates are closely related. The sign of the permuted inner product of two pairs of Plücker coordinates corresponding to two directed lines is the same as the sign of the scalar triple product of three vectors determined by the direction vectors of the lines. Determining the exit face given the entry face can be done using 3 to 6 scalar triple products. This method is more efficient than the method based on Plücker coordinates since Plücker coordinates are relatively expensive to compute.

Each of these methods is more efficient than several ray-triangle intersection tests or a full ray-tetrahedron intersection test. In our ray traversal algorithm we have opted for the scalar triple product method, which is simple to implement and efficient. However, additional performance gains are most likely possible using low-level optimizations, or by exploiting data level parallelism through SIMD instructions.

#### 4.3 Ray Connectivity

Ray traversal consists of locating the ray origin, which depends on the global number of tetrahedra in the tetrahedralization, and traversing the tetrahedralization, which depends on the local geometric complexity of the tetrahedralization. Locating the ray origin for every ray is potentially costly. Therefore we will avoid locating the ray origin by exploiting ray connectivity. When using a camera model for which all primary rays have the same origin, such as the traditional perspective camera, then the camera position has to be located only once. All primary rays have the same origin. Shadow rays and secondary rays start where the previous ray in the path ended. The cost for locating the camera position is amortized over all rays, and the complexity of tracing a single ray now only depends on the local complexity of the tetrahedralization.

#### 4.4 Data Structure

The data structure for the constrained tetrahedralization used during ray traversal is a static data structure, since both the topology of the tetrahedralization and the position of the ver-

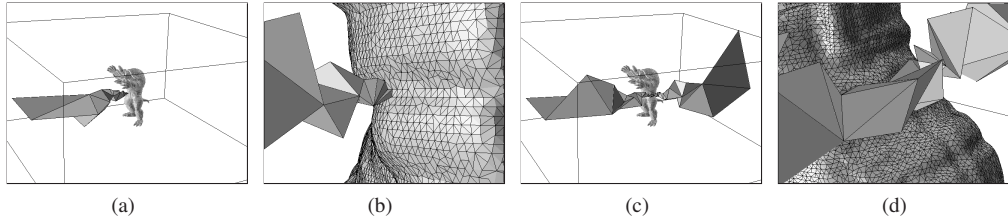


Figure 6: Ray traversal in 3D. (a) All tetrahedra traversed by a ray hitting the *Armadillo* model. (b) A close-up of (a). (c) All tetrahedra traversed by a ray just missing the *Armadillo* model. (d) A close-up of (c). The ray is not visible since it is entirely inside the tetrahedra.

tices is static. The data structure consists of an array of tetrahedra and an array of vertices. Each tetrahedron stores the indices of its vertices, the indices of its neighboring tetrahedra, and a boolean value for each of its faces indicating whether the face is constrained. Each vertex only stores its position. Edges and faces are not explicitly represented.

## 5 Experimental Results

In this section we compare constrained tetrahedralizations to state-of-the-art kd-trees, which are generally considered among the fastest acceleration structures currently available. We use the local greedy surface area heuristic kd-tree builder of Wald and Havran [WH06], including optimizations such as perfect splits. We have verified our implementation by accurately reproducing their results [WH06, table 1]. We use standard single-ray iterative kd-tree traversal. We did not use low-level optimizations or SIMD instructions in our constrained tetrahedralization traversal nor in our kd-tree traversal. Table 1 shows various statistics. All timings are obtained on a single core of a 3 GHz Intel Xeon X5365 CPU. All images are rendered at a resolution of  $512 \times 512$ .

### 5.1 Ray Tracing Cost

Figure 7 and table 1 compare the ray tracing cost of constrained Delaunay tetrahedralizations, quality Delaunay tetrahedralizations and kd-trees.

The *Neptune* scene is rendered using one primary ray per pixel using the constrained Delaunay tetrahedralization, a quality Delaunay tetrahedralization and a kd-tree. Figures 7(b) and figure 7(c) show the ray tracing cost for each pixel in the image, measured as the number of tetrahedra traversed by the corresponding primary ray. Figure 7(d) show the cost measured as the number of kd-tree nodes accessed by the ray, using the same quantitative color scale. Although the number of tetrahedra cannot be compared to the number of nodes, these false color images give an impression of the relative cost of the different parts of the image.

The quality Delaunay tetrahedralization is better suited for ray tracing than the constrained Delaunay tetrahedralizations. This is because quality Delaunay tetrahedralizations are more adaptive to the geometry than constrained Delaunay tetrahedralizations, which often contain many large tetrahedron fans. We will later present theoretical evidence for this behavior (see subsection 6.3).

Both the quality Delaunay tetrahedralization and the kd-tree are adaptive to the geometry. Rays that do not pass near

complex geometry have a lower cost than rays that do. The behavior of the quality Delaunay tetrahedralization is even slightly better than that of a kd-tree since it is more localized. For example, the hand of *Neptune* forces the kd-tree to introduce splitting planes that span a portion of the scene larger than the hand. Quality Delaunay tetrahedralizations are more localized than kd-trees because they are not hierarchical.

The ray tracing time using constrained Delaunay tetrahedralizations is within a factor 2 to 3 of the ray tracing time using state-of-the-art kd-trees. This is a surprising result, considering that constrained Delaunay tetrahedralizations were not primarily designed for ray tracing, and that efficient kd-trees are the result of two decades of research. We present will later more insight into this behavior in (see subsection 6.3).

### 5.2 The Teapot-in-a-Stadium Problem

The teapot-in-a-stadium problem consists of a setup where a small high-polygon-count object (the teapot) is placed in a large low-polygon-count environment (the stadium). The cost of a ray should only depend on the geometry in the neighborhood where the ray passes through.

Figure 8 shows three scenes constructed to test the teapot-in-a-stadium problem. The high-polygon-count *Chair* model is placed into a low-polygon-count *Forest* scene. For each of these scenes the ray tracing cost of the constrained Delaunay tetrahedralization, a quality Delaunay tetrahedralization and a kd-tree is shown.

The quality Delaunay tetrahedralizations are not subject to the teapot-in-a-stadium problem. Rays that do not pass near the complex geometry of the chair have the same cost as when the chair is omitted. Constrained Delaunay tetrahedralizations on the other hand suffer from the teapot-in-a-stadium problem. Although kd-trees are generally assumed not to be subject to the teapot-in-a-stadium problem, the influence of the chair is less localized than with the quality Delaunay tetrahedralization.

### 5.3 Ray Connectivity

Exploiting ray connectivity is important for reducing the time needed for ray traversal. Figure 9 shows the *Cornell Box* scene rendered using a point light source and shadow rays, and using an area light source and path tracing. The cost for locating the camera position is typically in the sub-millisecond range, even for large scenes such as the *Nep-*

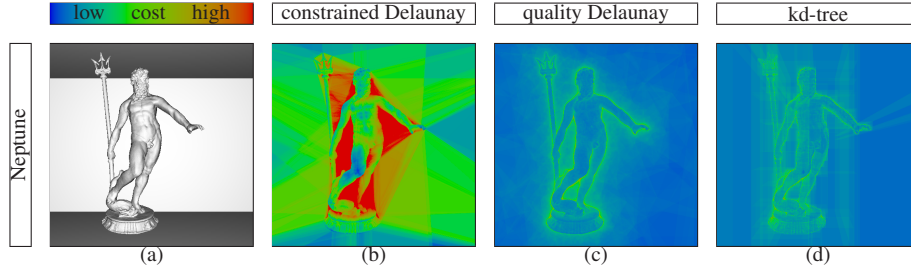


Figure 7: Ray tracing cost. (a) The *Neptune* scene. (b-d) False color images representing the ray tracing cost for ray tracing the scene using (b) the constrained Delaunay tetrahedralization, (c) a quality Delaunay tetrahedralization and (d) a kd-tree. The ray tracing cost is measured as (b, c) the number of tetrahedra or (d) the number of nodes traversed by a ray. The color scale used for all false color images is the same.

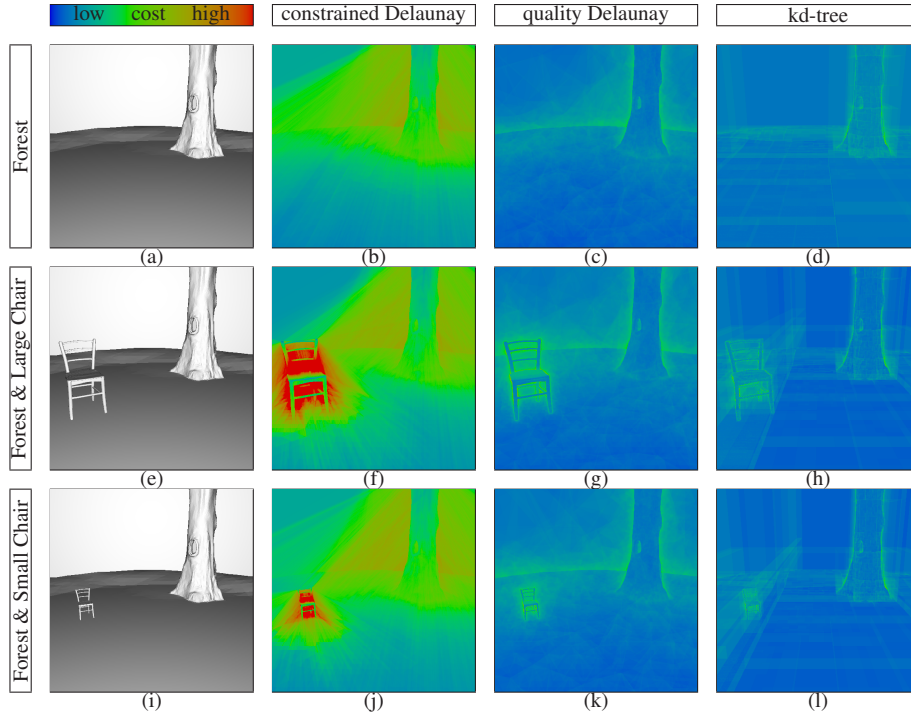


Figure 8: Teapot-in-a-stadium problem. (row 1) The low polygon count *Forest* scene, (row 2) the *Forest & Large Chair* scene, and (row 3) the *Forest & Small Chair* scene, obtained by adding a large and a small version of the high-polygon-count *Chair* model to the low-polygon-count *Forest* scene. (col 2, col 3, col 4) False color images representing the ray tracing cost for ray tracing these scenes using the constrained Delaunay tetrahedralization (col 2), quality Delaunay tetrahedralizations (col 3) and kd-trees (col 4).

*tune* scene, and is amortized over millions of rays. When ray connectivity is not exploited, the total rendering time for the path tracer image increases by about 30%, even when a grid is used to speed up ray origin location.

## 6 Discussion

In this section, we discuss several interesting and unique properties of constrained tetrahedralizations.

### 6.1 Optimal Constrained Tetrahedralizations

The average cost of a ray shooting query in a simplicial complex such as a constrained tetrahedralization is proportional to the weight of the simplicial complex [AF99], which is defined as the sum of the surface area of all faces of the tetrahedralization. This provides a heuristic for building good

tetrahedralizations, similar to the surface area heuristic for kd-trees [MB92]. The difference in performance between a kd-tree built using this heuristic and a kd-tree built using an ad-hoc method is often a factor 2 or more [WH06].

Table 1 shows experimental evidence that supports this theory. Quality Delaunay tetrahedralizations have a much smaller weight than constrained Delaunay tetrahedralizations and result in better ray tracing performance. Moreover, for every scene the ray tracing times are roughly proportional with the weight of the tetrahedralizations. Note that this relation does not necessarily hold between different scenes since the heuristic ignores effects due to locality of reference.

This is an important observation since it provides a heuristic similar to the surface area heuristic for kd-trees for build-


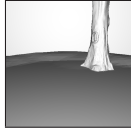
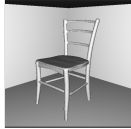
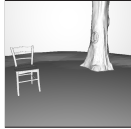
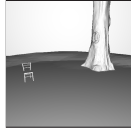

	neptune	forest	chair	forest & large chair	forest & small chair	armadillo
						
<b>scene statistics</b>						
# triangles	2.64M	17.52k	408.41k	425.92k	425.92k	345.95k
<b>constrained Delaunay tetrahedralization</b>						
# tetrahedra	8.53M	63.08k	1.43M	1.48M	1.46M	1.21M
# faces	17.07M	126.17k	2.86M	2.96M	2.92M	2.43M
# constrained faces	2.65M	19.40k	408.77k	428.44k	429.10k	350.78k
construction time	209.30 s	0.69 s	24.09 s	26.17 s	27.09 s	17.94 s
weight	9.41k	23.49k	14.06k	24.76k	24.21k	6.60k
render time	3.62 s	1.51 s	4.29 s	2.82 s	2.01 s	1.10 s
avg # tetrahedra / ray	158.18	115.12	183.13	142.74	126.48	86.82
<b>quality Delaunay tetrahedralization</b>						
# tetrahedra	11.71M	123.29k	1.84M	1.96M	1.97M	1.86M
# faces	23.42M	246.90k	3.69M	3.92M	3.95M	3.73M
# constrained faces	2.65M	21.26k	409.04k	430.69k	430.94k	357.32k
construction time	491.13 s	2.27 s	51.54 s	56.19 s	58.99 s	120.27 s
weight	2.20k	8.10k	2.65k	8.46k	8.22k	2.30k
render time	1.01 s	0.42 s	0.75 s	0.63 s	0.48 s	0.61 s
avg # tetrahedra / ray	45.05	40.39	45.20	47.06	44.02	47.49
<b>kd-tree</b>						
# leaf nodes	7.65M	47.27k	1.76M	1.65M	1.52M	452.50k
# n-emp leaf nodes	4.01M	27.51k	992.63k	921.46k	848.34k	203.38k
construction time	42.30 s	0.30 s	10.62 s	10.72 s	10.00 s	2.48 s
avg # triangles / n-emp leaf node	2.56	2.54	2.42	2.49	2.48	2.31
render time	0.37 s	0.23 s	0.29 s	0.24 s	0.22 s	0.28 s
avg # nodes / ray	45.17	38.94	43.78	32.58	30.86	46.22
avg # intersections / ray	2.24	2.41	2.19	2.43	2.40	2.20

Table 1: Statistics. Various statistics of the constrained Delaunay tetrahedralizations, quality Delaunay tetrahedralizations, and kd-trees.

ing constrained tetrahedralizations for ray tracing, and suggests that minimizing the weight of constrained tetrahedralizations will most likely increase ray tracing performance.

## 6.2 Numerical Robustness of Ray Traversal

Construction and traversal of acceleration structures is subject to numerical robustness errors due to the approximate nature of floating point arithmetic.

The construction of the constrained tetrahedralizations relies on adaptive precision floating point arithmetic and robust geometric predicates [She97] in order to ensure robust implementations of the geometric algorithms. This is common practice in computational geometry. The traversal of the constrained tetrahedralizations on the other hand does not include any mechanism to detect or avoid numerical robustness errors or to handle degenerate cases. These mechanisms often compromise speed and are in computer graphics usually traded for speed.

Although we did not experience any problems with numerical robustness, it is possible to detect and even to avoid these errors. Detecting problems during traversal can be done using a traversal algorithm based on the plane intersections method. If the ray parameters computed during traversal are not increasing, a wrong decision has been made. Avoiding problems during traversal can be done by simply temporarily moving a vertex in order to resolve the problem (see subsection 6.6).

## 6.3 Time Complexity of Ray Traversal

The time complexity of traversing a constrained tetrahedralization with a ray is linear in the number of tetrahedra traversed by the ray, and the number of tetrahedra traversed by a ray is relatively small. This assumes that the time for locating the camera position is amortized over all rays, that ray connectivity is exploited and that a quality Delaunay tetrahedralization is used. Interestingly, this time complexity does not depend on the total size of the tetrahedralization.

In contrast, the time complexity of traversing hierarchical acceleration structures such as kd-trees or bounding volume hierarchies with a ray is logarithmic at best in the total size of the kd-tree or bounding volume hierarchy.

This difference in time complexity cannot be overstated. The number of tetrahedra traversed by the ray can be interpreted as the geometrical complexity of the neighborhood of the ray. Constrained tetrahedralizations are probably the only acceleration structure for ray tracing that exhibit a time complexity in function of local geometric complexity rather than global geometric complexity.

For example, the number of tetrahedra traversed by a ray in a constrained tetrahedralization of a model of a single house will not change if several other houses are also added to the tetrahedralization. This is not the case for hierarchical acceleration structures such as kd-trees or bounding volume hierarchies, for which traversal starts at the root of the hierarchy. Although there have been some attempts to exploit ray connectivity with hierarchical acceleration structures, for ex-

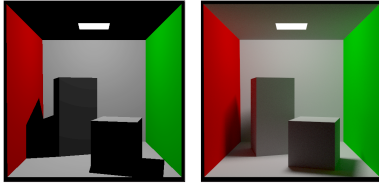


Figure 9: Exploiting ray connectivity. The *Cornell Box* rendered using (a) primary rays and shadow rays and (b) path tracing. Only the origin of the camera was located. The ray origin of all shadow rays and secondary rays was determined by exploiting ray connectivity, reducing rendering time by about 30%.

ample by augmenting kd-trees with neighbor links [MB92] or with sparsely distributed bounding boxes linked from bottom to top [HB07], exploiting ray connectivity with hierarchical acceleration structures is difficult.

Unfortunately, we have not yet been able to gather experimental evidence for this difference in time complexity. One reason is that a single traversal step in a kd-tree or bounding volume hierarchy is cheaper than a single traversal step in a constrained tetrahedralization, which means that the effect of this difference in time complexity might only show up for large scenes.

#### 6.4 Domain of the Constrained Tetrahedralization

Pellegrini [Pel97] assumes a subdivision of the entire space for solving the ray shooting problem in simplicial complexes such as constrained tetrahedralizations. In practice the domain of the constrained tetrahedralization is the convex hull of the piecewise linear complex rather than the entire space. This poses a problem for rays originating outside of the tetrahedralization. In this case the ray origin location fails.

The easiest solution is to enlarge the domain of the tetrahedralization to contain all possible ray origins. This is easily done by adding additional constraints to the piecewise linear complex when the constrained tetrahedralization is constructed, for example the bounding box of all possible camera positions.

The most general solution for rays that originate outside of the domain of the tetrahedralization is to determine the tetrahedron where the ray enters the tetrahedralization, for example by clipping the ray to the convex hull of the piecewise linear complex. This does not necessarily have to be an expensive operation, since the convex hull of the piecewise linear complex can easily be embedded in a larger simpler geometric structure, such as a bounding box.

#### 6.5 A Unified Data Structure for Global Illumination

So far, we presented constrained tetrahedralizations as an acceleration structure for ray tracing. However, constrained tetrahedralizations have the potential to unify several data structures currently used in global illumination algorithms.

In a constrained triangulation, arbitrary data can easily be associated with vertices, edges, faces and tetrahedra, and additional vertices, edges, faces and tetrahedra can easily be inserted to accommodate the data. Moreover, this data is di-

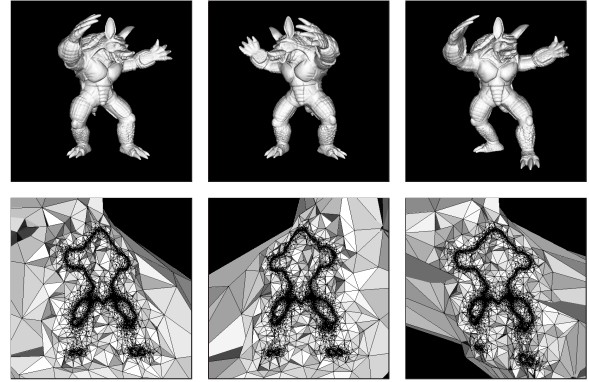


Figure 10: Ray tracing deforming geometry. The top row shows three frames of animations of a deforming *Armadillo* model. The bottom row shows the corresponding deformed constrained tetrahedralization. All frames were ray traced without reconstructing the tetrahedralization and without updating the topology of the tetrahedralization. (See accompanying video for full animations.)

rectly accessible during ray traversal. This means that other spatial data structures frequently used in global illumination, such as the photon map or the irradiance cache, can be merged into the constrained tetrahedralization, and that expensive lookups into these spatial data structures during ray traversal are completely eliminated. Tetrahedralizations are an excellent tool for interpolation of irregularly spaced data, and constrained tetrahedralizations can easily accommodate discontinuities in that data by introducing additional constraints. Therefore, constrained tetrahedralizations are well suited for interpolating global illumination data, such as irradiance estimates from photon maps.

#### 6.6 Deforming and Dynamic Geometry

Although several approaches for ray tracing animated scenes have been proposed, traditional acceleration structures have severe problems with handling deforming and dynamic geometry [WMG\*07]. Constrained tetrahedralizations on the other hand can support deforming and dynamic geometry relatively easy.

**Deforming Geometry** The data structure used to store a constrained tetrahedralization during ray traversal is static. Static data structures do not allow to modify the topology of the constrained tetrahedralization, but do allow to change the position of the vertices. Vertices can be moved freely as long as the topology of the tetrahedralization does not change. This is the case if the vertex stays within the union of the tetrahedra incident on the vertex. Moving vertices is useful for avoiding numerical robustness errors (see subsection 6.2) and for handling deforming geometry.

Figure 10 shows two frames of a deforming *Armadillo* model. All frames are rendered with a single static constrained quality tetrahedralization. Although the global deformation is quite large, the local deformation remains small, and the topology of the tetrahedralization does not change. The time to image for the 100-frame animation is 276 s us-

ing kd-trees and 189 s using a constrained tetrahedralization. The constrained tetrahedralizations is 30% faster because the build time is amortized over all frames.

Interestingly, it is possible to exactly describe for which deformations this works. If the deformation field is  $C^1$  continuous and divergence-free, then no local or global self-intersections can occur [vFTS06]. These deformation fields can model a wide variety of deformations. This means that in contrast with existing acceleration structures, constrained tetrahedralizations support deformations without any additional effort, simply by applying the deformation to all vertices. The range of deformations for which this works is probably much larger than that of several recently proposed deformable acceleration structures.

**Dynamic Geometry** Traditional acceleration structures have to be rebuilt from scratch every frame in order to support dynamic geometry. This is because efficient insertion and removal operations that do not compromise ray tracing performance are currently not available for hierarchical acceleration structures. Dynamic constrained tetrahedralizations on the other hand do allow to modify the topology and provide efficient insertion and removal operations. These data structures are similar to half-edge data structures used for meshes. This means that in contrast with existing acceleration structures, constrained tetrahedralizations can easily support dynamic geometry without completely rebuilding the tetrahedralization.

## 6.7 Other Advantages

Constrained tetrahedralizations have several other advantages not mentioned so far.

**Level-of-Detail** Combining traditional acceleration structures with level-of-detail is difficult. Dynamic constrained tetrahedralizations on the other hand use the same half-edge like data structures as dynamic meshes. It should be possible to directly perform progressive mesh operations such as vertex splits or edge collapses on meshes embedded in constrained tetrahedralizations.

**Ray Tracing on the GPU** Traditional hierarchical acceleration structures such as kd-trees and bounding volume hierarchies need to maintain a stack during ray traversal. This can be difficult on GPU-like architectures [PGSS07]. Constrained tetrahedralizations on the other hand only need the current tetrahedron for ray traversal.

## 6.8 Related Work

In electromagnetics, Yun et al. [YZI02] presented a ray tracing procedure for radio wave propagation based on a constrained Delaunay triangulation of a planar straight line graph. However, their work is limited to two dimensions.

In computer graphics, Márton [Már95] investigated the use of Voronoi diagrams for accelerating ray tracing. However, the faces of the acceleration structure are not aligned with the scene geometry. Marmitt and Slusallek [MS06] presented a method for ray tracing unstructured volume data by

traversing a Delaunay tetrahedralization, using a tetrahedron traversal algorithm similar to ours based on Plücker coordinates. Their work is based on earlier similar work by Garity [Gar90]. However, their methods are limited to volume data and are not designed for geometric models.

In finite element methods, Favre and Löhner [FL94] presented a method to ray trace a finite element mesh to display the results of a field solver analysis. Their idea is roughly similar to ours, but their work lacks an experimental analysis and offers few insights into the behavior of the approach.

## 7 Conclusion and Future Work

Constrained tetrahedralizations have been largely ignored in computer graphics, although they have a number of interesting and unique properties. Constrained tetrahedralizations are not hierarchical, have an interesting time complexity, can easily support deforming and dynamic geometry, and have the potential to unify several data structures used in global illumination. Constrained tetrahedralizations offer several new perspectives on acceleration structures for ray tracing and deserve attention.

There are plenty of opportunities for future work. The most important one is evaluating the performance of constrained tetrahedralizations for a large variety of geometric models. This has been difficult because geometric models in computer graphics are often very large and very badly conditioned. This is a problem for current implementations for geometry repair, and for current implementations for computing constrained Delaunay tetrahedralizations and quality Delaunay tetrahedralizations, which were developed in the context of finite element methods. Although these problems are mostly practical, solving them is a major effort, that cannot be justified without a proof of concept that shows that constrained tetrahedralizations are promising.

## Acknowledgments

Ares Lagae is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO). We are grateful to Peter Vangorp and Jurgen Laurijssen for last minute help, to Jan Welkenhuyzen from Materialise for help with geometry repair, and to Tim Volodine for several useful suggestions. We acknowledge *The Utah 3D Animation Repository*, *The Stanford 3D Scanning Repository*, and the *AIM@SHAPE Shape Repository* for the scenes used in this paper.

## References

- [AF99] ARONOV B., FORTUNE S. J.: Approximating minimum-weight triangulations in three dimensions. *Discrete and Computational Geometry* 21 (1999), 527–549.
- [Che89] CHEW L. P.: Constrained Delaunay triangulations. *Algorithmica* 4, 1 (1989), 97–108.
- [dBCvKO08] DE BERG M., CHEONG O., VAN KREVELD M., OVERMARS M.: *Computational Geometry: Algorithms and Applications*, third ed. Springer-Verlag, 2008.
- [ET92] EDELSBRUNNER H., TAN T. S.: An upper bound for conforming Delaunay triangulations. In *Proceedings*

- of the 8th annual symposium on Computational geometry (1992), pp. 53–62.
- [FL94] FAVRE J., LÖHNER R.: Ray tracing with a space-filling finite element mesh. *International Journal for Numerical Methods in Engineering* 37, 20 (1994), 227–253.
- [Gar90] GARRITY M. P.: Raytracing irregular volume data. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization* (1990), pp. 35–40.
- [Gla89] GLASSNER A. S. (Ed.): *An Introduction to Ray Tracing*. Academic Press Ltd., 1989.
- [Hav00] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University in Prague, 2000.
- [HB07] HAVRAN V., BITTNER J.: Ray tracing with sparse boxes. In *Proceedings of the Spring Conference on Computer Graphics 2007* (2007), pp. 49–54.
- [Már95] MÁRTON G.: Acceleration of ray tracing via Voronoi-diagrams. In *Graphics Gems V*. 1995, pp. 268–284.
- [MB92] MACDONALD D. J., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *The Visual Computer* 6, 3 (1992), 153–166.
- [MS06] MARMITT G., SLUSALLEK P.: Fast ray traversal of tetrahedral and hexahedral meshes for direct volume rendering. In *Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization (EuroVIS) 2006* (2006), pp. 235–242.
- [MTT\*96] MILLER G., TALMOR D., TENG S., WALKINGTON N., WANG H.: Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Proceedings of the 5th International Meshing Roundtable* (1996), pp. 47–61.
- [Pel97] PELLEGRINI M.: Ray shooting and lines in space. *Handbook of discrete and computational geometry* (1997), 599–614.
- [PGSS07] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P.: Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum* 26, 3 (2007), 415–424.
- [PT03] PLATIS N., THEOHARIS T.: Fast ray-tetrahedron intersection using plÅijcker coordinates. *journal of graphics tools* 8, 4 (2003), 37–48.
- [RS92] RUPPERT J., SEIDEL R.: On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete and Computational Geometry* 7, 3 (1992), 227–253.
- [RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. *ACM Transactions on Graphics* (2005), 1176–1185.
- [Sch28] SCHÖNHARDT E.: Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Journal Mathematische Annalen* 98, 1 (1928), 309–312.
- [SG05] SI H., GÄRTNER K.: Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the 14th International Meshing Roundtable* (2005), pp. 147–163.
- [She97] SHEWCHUK J. R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry* 18, 3 (1997), 305–363.
- [She98a] SHEWCHUK J. R.: A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 14th Annual Symposium on Computational Geometry* (1998), pp. 76–85.
- [She98b] SHEWCHUK J. R.: Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 16th Annual symposium on Computational geometry* (1998), pp. 86–95.
- [She07] SHEWCHUK J. R.: General-dimensional constrained Delaunay and constrained regular triangulations, I: Combinatorial properties. *Discrete and Computational Geometry* (2007). to appear.
- [Si06a] SI H.: On refinement of constrained Delaunay tetrahedralizations. In *Proceedings of the 15th International Meshing Roundtable* (2006), pp. 61–69.
- [Si06b] SI H.: TetGen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. <http://tetgen.berlios.de/>, 2006.
- [TH99] TELLER S., HOHMEYER M.: Determining the lines through four lines. *journal of graphics tools* 4, 3 (1999), 11–22.
- [vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Transactions on Graphics* 25, 3 (2006), 1118–1125.
- [Wal04] WALD I.: *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.
- [WH06] WALD I., HAVRAN V.: On building fast kd-trees for ray tracing, and on doing that in  $O(N \log N)$ . In *Proceedings of the IEEE 2006 Symposium on Interactive Ray Tracing* (2006), pp. 61–69.
- [WMG\*07] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. In *Eurographics 2007 State of the Art Reports* (2007).
- [WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive rendering with coherent ray tracing. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001)* 20, 3 (2001), 153–164.
- [YZI02] YUN Z., ZHANG Z., ISKANDER M. F.: A ray-tracing method based on the triangular grid approach and application to propagation prediction in urban environments. *IEEE Transactions on Antennas and Propagation* 50, 5 (2002), 750–758.