Conservative Radiance Interpolants for Ray Tracing

Seth Teller Kavita Bala Julie Dorsey
MIT Imagery and Simulation Group*

Abstract

Classical ray-tracing algorithms compute radiance returning to the eye along one or more sample rays through each pixel of an image. The output of a ray-tracing algorithm, although potentially photorealistic, is a two-dimensional quantity – an image array of radiance values – and is not directly useful from any viewpoint other than the one for which it was computed.

This paper makes several contributions. First, it directly incorporates the notion of radiometric error into classical ray-tracing, by lazy construction of *conservative radiance interpolants* in *ray space*. For any relative error tolerance ϵ, we show how to construct interpolants which return radiance values within ϵ of those that would be computed by classical (e.g., Whitted) ray-tracing. The second contribution of the paper is an explication of the four sources of aliasing inherent in classical ray tracing – termed *gaps, blockers, funnels, and peaks* – and an adaptive subdivision algorithm for identifying ray space regions *guaranteed* to be free of these phenomena. Finally, we describe a novel data structure that exploits object-space coherence in the radiance function to accelerate not only the generation of single images, but of image sequences arising from a smoothly varying sequence of eyepoints. We describe a preliminary implementation incorporating each of these ideas.

1 Introduction

One long-standing goal of visual simulation is to allow users to traverse realistically illuminated, geometrically detailed environments at interactive rates. Given a scene description, comprised of surface geometry and reflectance properties, and some specification of light sources, a class of techniques known as "global illumination algorithms" computes equilibrium distributions of radiant energy throughout the scene. That is, they compute some representation of radiant energy incident upon, or emerging from, every surface element. Radiosity, Ray-Tracing, and hybrid algorithms span a spectrum, in the way that they trade freedom of the synthetic viewpoint against the degree of complexity of the light sources, surface geometry, and surface reflectance properties.

1.1 Solution Complexity vs. Viewing Freedom

Radiosity algorithms, at one extreme, estimate radiosity (in units of power per area) on every input surface [11, 26]. Radiosity algorithms are usually formulated for polyhedral scenes, and can correctly handle only ideal diffuse emitters and receivers. Since the human visual system is very good at detecting and interpreting highlights that result from the directional variation of radiance, this representation is restrictive. However, since the computed solution (essentially a radiometric annotation of the input scene geometry) is *view-independent*, it is meaningful from any synthetic viewpoint.

Ray tracing algorithms, at the other end of the spectrum, estimate radiance (in units of power per source area per receiver steradian) for one or more rays through each pixel of an image [29, 10]. Radiance is a four-dimensional quantity over each surface, whose computation involves diffuse and specular effects, as well as reflection, transmission, and secondary emission effects. The radiance computation is therefore

*{seth,kaybee,dorsey}@graphics.lcs.mit.edu

extremely complex – so complex, in fact, that ray tracing algorithms *fix* the synthetic viewpoint and viewport, requiring them as additional inputs, and compute radiance *only for the specified viewpoint.* Thus, the output of a ray tracing algorithm, although potentially photorealistic, is a two-dimensional quantity – an image array of radiance values – and is not directly useful for any viewpoint other than the one for which it was computed.

A class of hybrid techniques has arisen that compute more general representations of the radiance function – typically, some discretized approximation to the radiance and/or irradiance field at each surface, or in space [27, 28, 25, 24, 4]. However, these representations have typically been extremely expensive to compute and store. Thus, the important visual cues provided by these algorithms (shadows, smooth shading, transparency, reflections, etc.) have not been achievable in interactive visual simulation applications.

1.2 Solution Error vs. Running Time

Rendering is a numerical simulation problem, and as such is subject to various sources of error. Given a fixed scene geometry and reflectance model, another long-standing problem in computer graphics is to achieve a controllable tradeoff between the time expended computing a solution, and the fidelity of that solution. Hierarchical radiosity techniques [13] explicitly incorporate solution error via lazy enumeration of coupling coefficients according to a per-coefficient error estimate. Sophisticated error bounds for radiosity solutions have also been described (e.g. in [18]). Ray tracing and hybrid algorithms have historically employed supersampling or distribution sampling to decrease solution error.

1.3 Our Approach

Even using today's most powerful workstations, it is overambitious to attempt to represent the radiance distribution for all surfaces, from all viewing directions, for a non-trivial scene. However, it is feasible to compute radiance *lazily* for a small *predictive* neighborhood around the synthetic viewpoints exercised in a real visual simulation application – thereby exploiting both the *visibility coherence* of an observer moving through the scene, and the *smoothness* of the radiance field. We describe a system for radiance computations based on a number of novel ideas:

- An **error tolerance** ϵ directly controls the tradeoff between solution error and rendering time;
- An **adaptive, per-surface radiance interpolant** captures the radiance (angular flux density) leaving the surface to within ϵ, for a lazily-constructed envelope of directions;
- Radiance interpolation is performed by **quadrilinear interpolation** in **ray space**;
- The four causes of aliasing in classical ray tracing[1] – gaps, blockers, funnels, and peaks – are identified, and ray tree **topologies** and **isotopies** defined. A **conservative predicate** identifies isotopies; ray trees of like topology are interpolated only when they form an isotopy.
- The fundamental visibility operation in ray tracing, ray casting, is **explicitly decoupled** from the fundamental radiometric operation, integration of irradiance, yielding a sampling scheme that suffers absolutely no unrecognized geometric aliasing, and strictly bounded radiometric aliasing, at every pixel.

2 Previous Work

Ray casting was introduced in 1968 as a technique for visibility determination and the rendering of hard shadows [2]. Classical ray tracing was proposed in 1980 to simulate reflection and refraction

[1]We do not consider texture mapping in this work.

phenomena [29]. This approach identifies only those paths along which radiance reaches the eye from the specified light sources or background [10]. Given a scene description comprised of implicit primitives (each with associated material properties), one or more local or infinite point light sources, an ambient light level, and a synthetic eyepoint, view frustum, and viewport, the ray tracing algorithm computes a representation of the radiance returning to the eye along a set of sample "eye rays" through each pixel. Radiance values are computed by recursively spawning a **ray tree** for each eye ray [10]. Exemplary pseudocode for a backward ray-tracer can be found in [14]. A host of refinements and generalizations have since been proposed [10, 7].

2.1 Ray Tracing Acceleration

Ray tracing algorithms perform an expensive operation – finding the first intersection of a ray with some scene object – tens or hundreds of times per image pixel, and are therefore extremely compute intensive. These algorithms can be accelerated by making individual intersection tests faster via spatial subdivision or creative use of a z-buffer; by amortizing the per-ray intersection cost with a more general ray representation; or by spawning fewer rays via adaptive sampling or importance sampling (see [10], pp. 204-5 for a taxonomy of such techniques current to 1989).

Most ray tracing acceleration methods have concentrated on reducing or amortizing the cost of ray casting. This has been done by casting bundles of rays, as did beam-tracing for polyhedral scenes [15] and cone-tracing [1]; employing a fixed-grid [6] or adaptive [8] 3D spatial hierarchy of scene objects; and employing a 5D hierarchy associating ray sets and the object(s) encountered by rays in the set [3].

While many researchers have concentrated on reducing the per-ray intersection cost, relatively few published algorithms have the intent of reducing the number of rays cast. Popular public domain ray tracers such as RayShade [17] use an adaptive subdivision technique based on variance of screen space radiance samples. This is a necessary, but not sufficient, criterion for avoiding aliasing.

Some researchers have addressed the problem of generating image sequences for scenes whose geometry [9] or material properties [22] change with time. To the authors' knowledge, however, no existing work explicitly addresses the generation of ray-traced images from a sequence of smoothly varying eyepoints.

2.2 Truncation Error

When gathering irradiance on a given surface, classical ray tracing accounts for irradiance only along a finite set of directions: from all point light sources; along the reflection direction of the sample ray; and along the transmission direction of the ray. Moreover, recursive ray spawning is suppressed when the ray tree depth (weight) exceeds (falls below) some prespecified threshold. This "truncated" integration can give rise to error by excluding ray paths along which significant radiance may flow (e.g., light bouncing among many surfaces before reaching the eye). Researchers have addressed this source of error using distribution ray tracing [5], forward ray tracing [10] and path tracing [16]. This paper does not address truncation error, as our system is designed to compute an ϵ-approximation to the radiance values computed by classical ray tracing.

3 Algorithm Overview

This section presents a brief overview of our algorithm. With every object we associate a lazily constructed representation of radiance from the object, which we call a "radiance interpolant." To generate an image, an eye ray is cast through each pixel of the associated viewport. If the eye ray hits no object, a background radiance is returned. Otherwise, the eye ray hits an object, and a radiance interpolant for the object is lazily refined through analytic ray tracing operations. This interpolant construction may or may not succeed. If it succeeds, a radiance value for the original eye ray is produced through quadrilinear interpolation of radiance values stored with the interpolant. Otherwise, a radiance value is produced by tracing the original eye ray through the scene as in classical ray tracing.

4 Aliasing

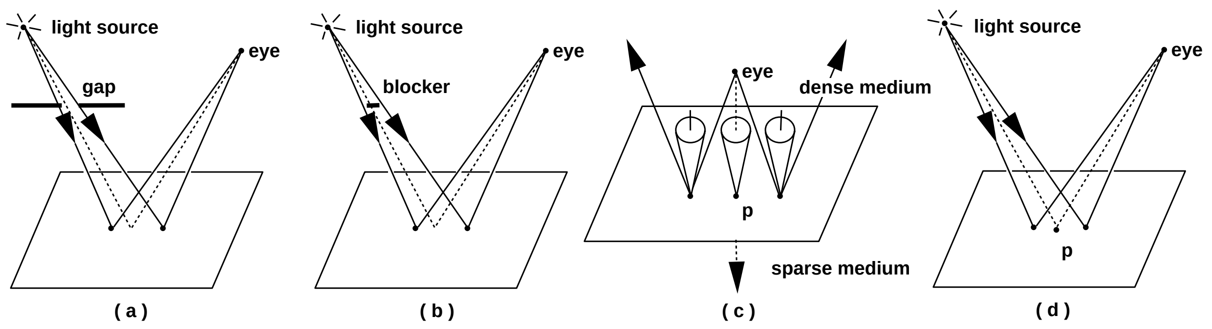

Figure 1: Aliasing errors due to insufficient radiance sampling.

There are four ways in which aliasing errors can arise in classical ray tracing. They are depicted in Figure 1, with sample rays shown as solid lines, and dashed lines depicting significant radiance paths/nonpaths missed due to insufficient sampling. First, **gaps** might not admit sample ray paths, yet admit intervening rays (Figure 1-a). Second, **blockers** might admit sample rays, yet block intervening rays (Figure 1-b). (Horizons or terminators occur when an object "self-blocks," obscuring paths from its own surface to a light source.) Third, when traversing an interface from a dense to a sparse medium (e.g., from water to air), sample rays might be totally internally reflected, whereas intervening rays are transmitted (Figure 1-c). We call this source of aliasing a **funnel**, because the set of transmitted rays through a dense/sparse boundary have directions which fall within the Gauss map of all normals on the boundary, Minkowski-producted (roughly, expanded) by a cone whose opening angle is determined by the ratio of indices of refraction of the involved media. Finally, an intervening ray might encounter a **peak** in radiance centered on a specular highlight (for example, at point **p** in Figure 1-d). Radiance peaks occur at extremal paths, that is, those of locally minimal or maximal optical length (see [20] for a discussion of such paths in the context of ray tracing algorithms).

5 Ray Trees

Rays spawned by a ray tracing algorithm not only have geometric attributes, but can also be thought of as "carrying" radiance, a physical quantity. Moreover, we have seen that as a sample ray changes position or direction, the radiance along that ray (as computed by classical ray tracing) changes smoothly, except at discontinuities caused by gaps, blockers, or funnels. Thus, it should be possible to interpolate between radiance samples to produce values which are weighted averages of those associated with the samples. When is it sensible to interpolate radiance samples?

The spawning of a ray can be thought of as inducing a *ray tree* [10] (Figure 2). Each tree leaf corresponds to an aggregation of direct illumination from a light source or the background, is labeled with a unique identifier for that source, and stores the radiance collected from the source. Each internal node corresponds to a recursively spawned ray, traced to gather radiance reflected by, or transmitted through, some scene object. Such nodes are labeled with a unique identifer for the intervening object, and store the radiance gathered from the object (in the figure, ray paths are marked with arrows indicating the direction along which radiance is gathered). Each ray tree edge corresponds to a ray cast (i.e., a ray-scene intersection operation); we find it useful to consider *ordered* lists of edges as representing a sequence of such calls.

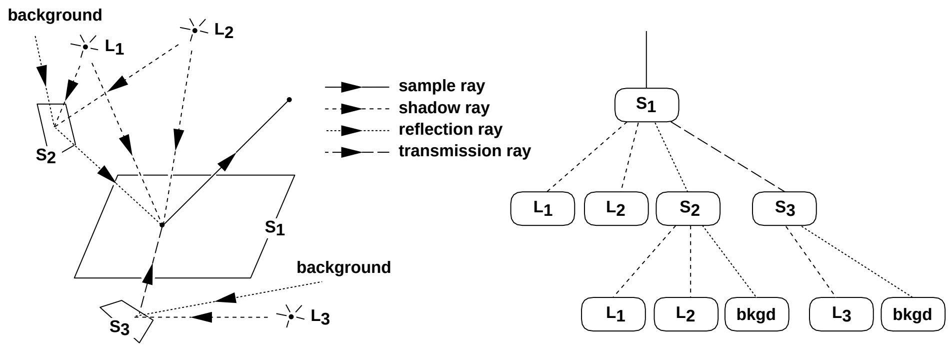

Figure 2: The ray tree induced by a spawned eye ray.

5.1 Ray Tree Isomorphisms

Two ray trees $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic* if there exists a one-to-one correspondence Π between their nodes, and if, whenever some node $\mathcal{N}_{\mathcal{A}}$ in $\mathcal{A}$ has as its k^{th} subtree $\mathcal{M}_{\mathcal{A}}$, its corresponding node $\Pi(\mathcal{N}_{\mathcal{A}})$ in $\mathcal{B}$ has as its k^{th} subtree the node $\Pi(\mathcal{M}_{\mathcal{A}})$. We say that isomorphic ray trees have the same *topology.* Together, every surface point and viewing direction of that point (for a fixed surface, a 4D range) induce a ray tree of some topology. A crucial observation is that **radiance changes discontinuously only when the topology of the ray tree changes**. Imagine a 2D pencil of rays through a neighborhood of points on some directly illuminated surface (Figure 3-a). An observer experiences a radiance discontinuity when viewing the hard shadow edge cast by the occluder. Radiance discontinuities occur under purely directional variation as well (Figure 3-b). Consider a 2D pencil of rays through a fixed point **p** on some indirectly illuminated surface. An observer viewing **p** from varying directions will experience a radiance discontinuity when the reflection of object S_2 starts (or stops) containing **p**.

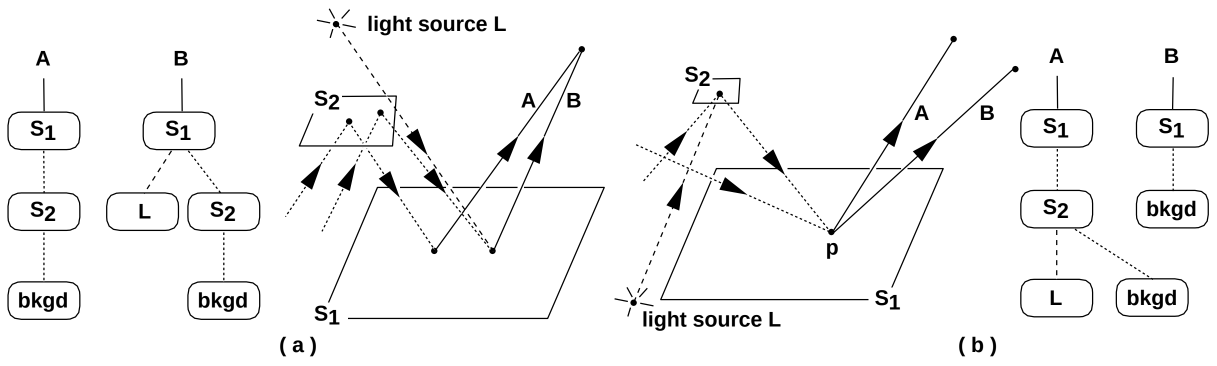

Figure 3: Radiance discontinuities occur where the ray tree topology changes.

5.2 Ray Isotopies and Radiance Interpolation

Intuitively, if one is to avoid interpolating across a radiance discontinuity, one must average only samples which have been derived from isomorphic ray trees. This is a necessary, but not sufficient, condition for discontinuity-free radiance interpolation since intermediate rays may induce ray trees that are not isomorphic to those of the sample rays (cf. Figures 1-a, b and c). A sufficient condition is that *all rays in the ray space convex hull* of the sample set induce isomorphic ray trees, banishing any radiance discontinuities there. We call such a set an *isotopy* of rays. The following subsections detail the identification of isotopic regions for a scene comprised of convex objects. Recall that ray isotopies can be interrupted only by gaps, blockers, and funnels.

Detecting Gaps: Since convex objects cannot have holes, any two sample ray trees which span a gap must encounter distinct objects, and can not be isomorphic. Thus, the presence of gaps inside a ray subregion can be detected by comparing the topologies of the region's extremal ray trees.

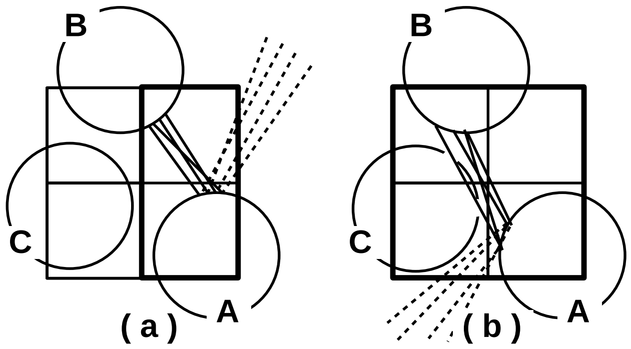

Figure 4: Establishing the absence (a) or potential presence (b) of blockers.

Detecting Blockers: We detect blockers using a dynamic version of *shaft culling* [12]. Shaft culling was originally developed for use in a radiosity algorithm based on ray-casting.

Our implementation uses kd-trees to partition the scene hierarchically for efficient ray casting [10]. Every ray between two points A and B propagates through some well-defined set of cells in this spatial subdivision. Every two cells in the subdivision have some "least common ancestor"; that is, the deepest (smallest) cell which contains them both. Thus, given the extreme rays in some convex ray space region R, all of which connect scene objects A and B, we search the least common ancestor (shown in bold in Figure 4-b) of all cells touched by the rays. If the least common ancestor has no descendants other than A or B, by convexity it must be true that no ray in the interior of R meets any object other than A or B. If its descendants contain objects other than A and B, we apply shaft culling against the bounding boxes of these objects to test for potential blockers.

Note that this blocker identification technique is *conservative*, i.e., it sometimes reports objects that are not truly intersected by the rays. However, it always identifies objects that are truly blockers (such as the sphere C in Figure 4-b).

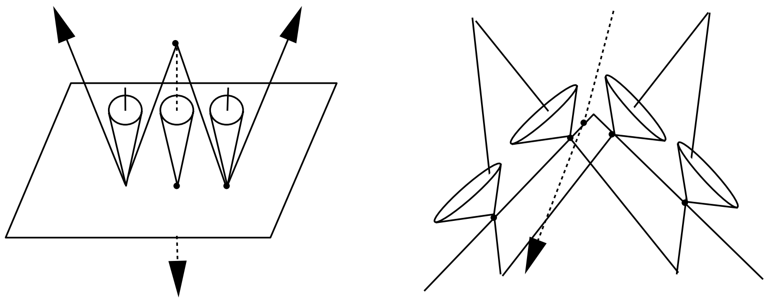

Figure 5: Totally-internally reflecting (TIR) samples may interpolate to non-TIR samples (bold and dashed, at left), and vice-versa (right), even when all geometric primitives are convex.

Detecting Funnels: Finally, we address the issue of detecting funnels; that is, regions of ray space in which extremal samples are (are not) totally internally reflecting across a dense/sparse interface, whereas interpolated samples are not (are) totally internally reflecting (cf. Figure 1). Unfortunately, it is easy to exhibit instances of both positive and negative funnels (Figure 5); we know of no simple, general method for detecting their presence. However, a *conservative* subdivision strategy preserves the correctness of our approach: we simply label any raytree containing a dense/sparse interface as non-isotopic. This may cause excessive sampling; however, it will never allow interpolation across a funnel.

We note that existing beam-tracing [15] cone-tracing [1], and pencil-tracing [23] algorithms encounter this problem in a different form; refraction causes linear descriptions of ray bundles to become nonlinear. Each of these algorithms employed geometric approximation techniques to trace refracted ray bundles, but showed no bounds for the geometric and radiometric errors so incurred.

5.3 Bounding Interpolant Error

Ray tree leaves correspond to a recursion termination (zero gathered radiance), a ray's escape from the scene (to background, collecting constant gathered radiance), or the evaluation of radiance from a point light source (Phong shading, non-constant radiance). The latter cases, appropriately weighted, are the only sources of radiance at the root of the ray tree. Thus, to bound the radiance that could arise from any query ray in some isotopy, we must bound the radiance computed at the ray tree leaves, then multiply these bounds by the aggregate weight of the subtree (note that this weight is the same for all ray trees in the isotopy).

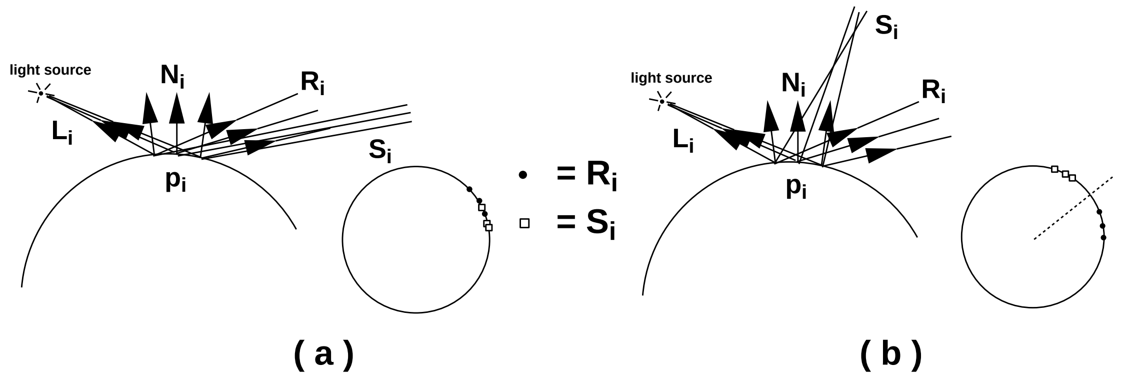

Figure 6: A radiance peak can exist only if the reflection directions $\mathbf{R}_i$ and the sample directions $\mathbf{S}_i$ are not separable on the Gauss map.

The radiance function is unimodal for a fixed viewing position or a fixed viewing direction. However, when the sample direction varies, radiance can have many maxima. Bounding the radiance exactly within some ray isotopy could be done by identifying a ray within the region for which $\mathbf{N} \cdot \mathbf{H}$ (or $\mathbf{N} \cdot \mathbf{L}$) is maximized, then evaluating radiance along that ray. However, again we formulate a simpler, conservative strategy. We simply subdivide until any maxima of $\mathbf{N} \cdot \mathbf{H}$ are banished from the interior of the isotopy. We do so by analyzing the Gauss map of the surface normals (Figure 6).

For an isotopic sample set $\mathbf{S}_i$, define the sample rays' points of intersection with some primitive as the $\mathbf{p}_i$, and the induced light vectors $\mathbf{L}_i$ and normals $\mathbf{N}_i$. A radiance peak can occur only if there exists some interpolated sample direction which is the reflection of its induced light source direction. This can occur if and only if the Gauss map of the sample directions intersects that of the reflection directions $\mathbf{R}_i$ (Figure 6-a). Otherwise, the maximum radiance arises at one of the $\mathbf{S}_i$. A simple conservative test for non-intersection of two Gauss maps is that of strict linear separability; that is, two sets $\mathbf{A}_i$ and $\mathbf{B}_j$ of vectors are strictly linearly separable iff there exists a vector $\mathbf{D}$ such that $\mathbf{D} \cdot \mathbf{A}_i < 0 \ \forall i$ and $\mathbf{D} \cdot \mathbf{B}_j > 0 \ \forall j$. This 3D linear programming problem can be solved in time linear in the total number of vectors [19].

6 Radiance Interpolants

It remains to elucidate an interpolant mechanism that, given an isotopic region of rayspace, produces for every query ray in the region radiance values within ϵ of those that would be computed by classical ray tracing. This section describes a mechanism by which all rays through some volume of space can be represented, and by which a sample or "query" ray can be checked for inclusion in a represented set. We wish to generate ray traced images using far fewer radiance samples than would classical ray tracing. To do so, we lazily construct *radiance interpolants in ray space*, then judiciously substitute queries to such interpolants for most (but not all) radiance sampling operations.

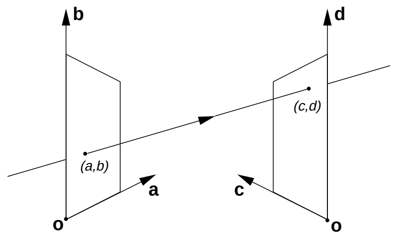

Figure 7: A ray coordinatized by its 2D intercepts with two reference planes.

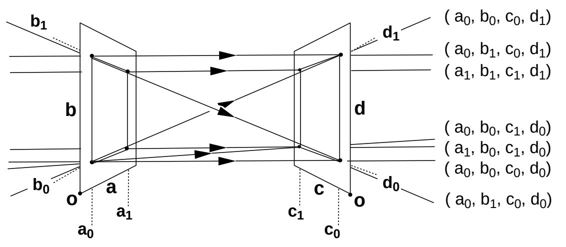

Figure 8: A ray subregion corresponds to a 4D hypercube – the root of a linetree.

Given a pair of reference planes, those rays parallel to neither plane can be *coordinatized* by a pair of 2D intercepts (e.g., *(a,b)* and *(c,d)* in Figure 7). Thus, the rays through a pair of bounded regions (e.g., squares) on two reference planes form a four-dimensional set.

Given a ray described in 3D, the ray's 4D ray-space coordinates can be defined with reference to a box around the object which ray intersects. Every such box has six "face-pairs" of opposing (parallel) faces. To represent a ray, we select the face-pair that is maximally perpendicular to the ray; the two faces involved must be expanded to capture all rays that could intersect the object.

All rays in a subregion of ray space can be generated by allowing (a, b, c, d) to take values in the domain $[a_0..a_1] \times [b_0..b_1] \times [c_0..c_1] \times [d_0..d_1]$. Conversely, one may determine whether a given ray (a, b, c, d) lies inside a region in ray space by performing four interval tests, one for each of the ray coordinates a, b, c, d. Using this *geometric* representation of rays, we associate radiometric information (i.e., radiance values) in a data structure called a "linetree" (Figure 8 depicts the root cell of a linetree). When a radiance query is made on a linetree cell whose sixteen extremal rays do not form an isotopy, the linetree is adaptively subdivided.

7 Implementation

We modified a classical Whitted ray-tracer to use lazy radiance interpolation. This section describes the algorithm in detail. Given an eye ray and an object it intersects, we must determine whether interpolation will fail or succeed for the ray. To do so, we first identify the face-pair and (a, b, c, d) intercepts for that pair. We then find the leaf linetree containing (a, b, c, d) (cf. Figure 8). If the interpolation predicate for that leaf is `TRUE`, interpolation is performed, producing a radiance value for the eye ray. Otherwise, if the (leaf) linetree's depth does not exceed some maximum depth, it is subdivided. Construction of an interpolant is then attempted for the (single) new child which contains the original eye ray. In this fashion, interpolants are "lazily" constructed.

To build an interpolant for an object, 16 interpolant rays are cast. If all 16 hit the object, a shading operation is invoked for each interpolant ray. If the shading calls produce isomorphic raytrees, and all raytrees satisfy the blocker, specularity, and epsilon tests, the predicate associated with the linetree cell is set to `TRUE`. In this case, subsequent radiance queries to this linetree cell will be successfully interpolated. Otherwise, the predicate is set to `FALSE`; subsequent queries to this linetree cell will result in subdivision as described above.

We employ a sphere illuminated by a single light source to illustrate the algorithm's operation. Color Plate 1-(a) depicts how our algorithm isolates all possible sources of aliasing. The yellow and green areas correspond to regions of subdivision due to non-isomorphic raytrees – due to silhouetting and self-shadowing, respectively. The cyan region corresponds to subdivision around the specular highlight on the sphere (i.e., failure of the specularity test). The purple region corresponds to subdivision due to the relative error between the samples exceeding epsilon (i.e., failure of the epsilon test). Color Plate 1-(b) shows the aliasing artifacts that can arise if specular highlights are not detected. The image was generated by disabling the specularity test. This causes erroneous interpolation across the specular highlight. The final result of our algorithm in the case of this simple scene is depicted in Color Plate 1-(c).

A multiple-object scene is required to illustrate failure of the blocker test. Color Plate 2-(a) depicts a face model with 7 primitives, rendered with $\epsilon = 0.5$. Color Plate 2-(b) is color-coded as before. The red regions in the scene correspond to regions of subdivision due to potential blockers.

8 Results

The ray tracer uses two primitive operations: `Intersect`, which computes the intersection of a ray with the objects in the scene, and `Shade`, which computes the radiance at a point by local and global recursive computations of the radiance.

Tot_I and Tot_S are the total number of intersection and shade operations. Ip_I and Ip_S are the intersection and shade operations invoked by our scheme to build interpolants. $Base_I$ and $Base_S$ are the intersection and shade operations invoked when interpolation fails. (For the classical ray tracer, Ip_I and Ip_S are zero.)

8.1 Coherence

This section discusses the algorithm's performance on two models (depicted in Color Plate 2-(a) and Color Plate 3-(a)). The face model comprises 7 primitives. The room model is more complex, with 18 primitives. All images were rendered using our radiance interpolant algorithm. We instrumented both the initial and modified ray tracer to record $Base_I$, $Base_S$, Ip_I, and Ip_S. For these runs, the ray tracer recurses to a maximum depth of 4, and cuts off recursion if the aggregate weight is less than 0.01.

Frame	**Intersect Counts (x 1000)**				**Inter.**	**Shade Counts (x 1000)**				**Shade**	**Ip**
	Ip Scheme			**Classical**		**Ip Scheme**			**Classical**		
#	**Base**	**Ip**	**Tot**	**Tot**	**Ratio**	**Base**	**Ip**	**Tot**	**Tot**	**Ratio**	**Success**
1	257	235	492	466	**1.06**	54	83	137	159	**0.86**	70%
2	266	64	330	486	**0.63**	56	23	79	166	**0.47**	71%
3	276	43	319	506	**0.59**	57	15	72	172	**0.42**	71%
4	273	48	321	502	**0.60**	57	17	74	171	**0.44**	71%
5	270	63	333	495	**0.63**	56	23	79	169	**0.47**	71%

Table 1: Face model: Counts in thousands, and ratios with $\epsilon = 0.25$.

Table 1 presents the results for the face image depicted in Color Plate 2, with $\epsilon = 0.5$. The second to fourth columns present $Base_I$, Ip_I, and Tot_I (in thousands) of our scheme, and we present Tot_I of classical ray tracing in the fifth column for comparison. The sixth column presents the ratio of intersect calls made by the interpolant algorithm to those made by the unmodified ray tracer. Similarly, the seventh to ninth columns present $Base_S$, Ip_S, and Tot_S (in thousands) of our scheme, and we present

Tot_S of classical ray tracing in the tenth columns for comparison. The second to last column presents the ratio of shade calls made by the two ray tracers. The last column presents the percentage of pixels for which interpolation was successful.

The first row presents results for the initial rendering of the scene. The next 2 rows correspond to small movements in the eye forward (along the viewing direction). The last 2 rows correspond to small movements of the eye sideways. Similarly, Table 2 presents the results for the room model (Color Plate 3), from which images were generated with $\epsilon = 0.1$.

The results demonstrate the algorithm's use of spatial and temporal coherence. The first time the scene is traced, our algorithm populates the linetrees with interpolant samples by shooting interpolant rays. The first row of Table 1 shows that the first time the scene is rendered our algorithm increases the intersection operations by 6%, whereas shade operations *decrease* by 14% due to coherence of the radiance field. Moving forward or sideways can cause unexplored parts of the scene to be exposed, and interpolants have to be built for these exposed regions. Subsequent small, coherent movements of the eye reduce the number of intersections and shades by up to 41% and 58%, respectively.

Frame	**Intersect Counts (x 1000)**				**Inter.**	**Shade Counts (x 1000)**				**Shade**	**Ip**
	Ip Scheme			**Classical**		**Ip Scheme**			**Classical**		
#	**Base**	**Ip**	**Tot**	**Tot**	**Ratio**	**Base**	**Ip**	**Tot**	**Tot**	**Ratio**	**Success**
1	166	304	470	384	**1.31**	24	78	102	91	**1.13**	76%
2	162	146	308	384	**0.72**	23	40	63	91	**0.69**	77%
3	161	106	267	383	**0.58**	22	29	51	90	**0.56**	78%
4	160	107	267	377	**0.59**	23	27	50	88	**0.56**	77%
5	159	114	273	370	**0.62**	23	29	52	87	**0.60**	76%

Table 2: Room model: Counts in thousands, and ratios with $\epsilon = 0.25$

The results in Table 2 correspond to the room model and they agree with those of the face model. The first time the scene is rendered the number of intersect and shade operations increase by 31% and 13% respectively. However, subsequent movements of the eye decrease the number of intersects by up to 42%, and the number of shades by up to 44%.

8.2 Error Bounds

We also varied ϵ to observe its impact on image quality. Color Plate 3-(a) depicts an image generated with $\epsilon = 0.5$. Some artifacts arise in the mirror and along the bottom of the mirror where interpolation takes place across the edge of the wall. However, reducing ϵ to 0.25 produces an image with no visible artifacts (Color Plate 3-(b)).

ϵ	**Intersect Ratio**	**Shade Ratio**	**Ip Success**
1.00	0.85	0.90	90%
0.50	1.12	0.96	78%
0.25	1.31	1.13	76%
0.00	3.46	2.89	0%

Table 3: Ratios as ϵ is varied.

Table 3 summarizes the quantitative impact a range of ϵ values have on the results. As ϵ values decrease, the number of intersect and shade operations increase, and the success rates drop. However, we find that the change is not very dramatic, and good performance can be achieved in a scene with no visible artifacts. It is also useful to note that decreasing ϵ from 1.0 to 0.25 achieves good image quality but does not substantially increase the work.

9 Conclusion

We described a novel scheme for lazy computation of information about radiance, organized by a per-surface, hierarchical subdivision of linespace. We built a prototype implementation of this strategy, and showed that it accelerates a classical ray-tracing computation by reducing the number of primitive geometric and radiometric operations performed. Our scheme exploits screen-space, object-space, and temporal coherence, and is the first to trade radiometric error bounds for the computation incurred in ray tracing. The approach accelerates not only the generation of single images but also the generation of images for a coherent sequence of viewpoints.

We intend to extend this approach in a number of ways. First, we will apply it to environments that exhibit a full range of BRDFs, and greater geometric complexity. Second, the isotopy predicate can be used to eliminate wasteful supersampling over homogenous screen-space regions. Finally, in an interactive setting, we are investigating strategies for persistent storage of radiance values in portions of the scene that have already been traversed, and for discarding such values when they have outlived their usefulness.

10 Acknowledgments

We are grateful to Light Control, Inc. for funding this research. Michael Werman's helpful comments and advice are gratefully acknowledged. We would also like to thank Andrew Myers for his useful suggestions.

References

[1] Amanatides, J. Ray Tracing with Cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (July 1984), H. Christiansen, Ed., vol. 18, pp. 129–135.

[2] Appel, A. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of SJCC* (1968), Thompson Books, Washington, D.C., pp. 37–45.

[3] Arvo, J., and Kirk, D. B. Fast Ray Tracing by Ray Classification. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 55–64.

[4] Chen, S. E., Rushmeier, H., Miller, G., and Turner, D. A Progressive Multi-Pass Method for Global Illumination. *Computer Graphics (SIGGRAPH '91 Proceedings) 25*, 4 (July 1991), 165–174.

[5] Cook, R. L., Porter, T., and Carpenter, L. Distributed Ray Tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (July 1984), vol. 18, pp. 137–45.

[6] Fujimoto, A., and Iwata, K. Accelerated Ray Tracing. *Computer Graphics: Visual Technology and Art (Proc. Computer Graphics Tokyo '85)* (1985), 41–65.

[7] Glassner, A. *Principles of Digital Image Synthesis.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1995.

[8] Glassner, A. S. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications 4*, 10 (1984), 15–22.

[9] Glassner, A. S. Spacetime Ray Tracing for Animation. *IEEE Computer Graphics and Applications 8*, 2 (Mar. 1988), 60–70.

[10] Glassner, A. S., Ed. *An Introduction to Ray Tracing.* Academic Press, 1989.

[11] GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. Modeling the Interaction of Light Between Diffuse Surfaces. *Computer Graphics (Proc. Siggraph '84) 18*, 3 (1984), 213–222.

[12] HAINES, E., AND WALLACE, J. Shaft Culling for Efficient Ray-Traced Radiosity. In *Proc.* 2^{nd} *Eurographics Workshop on Rendering* (May 1991).

[13] HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics (Proc. Siggraph '91) 25*, 4 (1991), 197–206.

[14] HECKBERT, P. Writing a Ray Tracer. In *Introduction to Ray Tracing*, A. Glassner, Ed. Academic Press, 1989, pp. 263 – 294.

[15] HECKBERT, P., AND HANRAHAN, P. Beam Tracing Polygonal Objects. *Computer Graphics (Proc. Siggraph '84) 18*, 3 (1984), 119–127.

[16] KAJIYA, J. T. The Rendering Equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)* (Aug. 1986), D. C. Evans and R. J. Athay, Eds., vol. 20, pp. 143–150.

[17] KOLB, C. RayShade Homepage `http://www-graphics.stanford.edu/~cek/rayshade/`.

[18] LISCHINSKI, D., SMITS, B., AND GREENBERG, D. P. Bounds and Error Estimates for Radiosity. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)* (July 1994), A. Glassner, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 67–74. ISBN 0-89791-667-0.

[19] MEGIDDO, N. Linear-time algorithms for Linear Programming in R^3 and Related Problems. *SIAM Journal Computing 12* (1983), 759–776.

[20] MITCHELL, D. P., AND HANRAHAN, P. Illumination From Curved Reflectors. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 283–291.

[21] NIMEROFF, J., DORSEY, J., AND RUSHMEIER, H. A Framework for Global Illumination in Animated Environments. In *6th Annual Eurographics Workshop on Rendering* (June 12–14 1995), pp. 223–236.

[22] SÉQUIN, C. H., AND SMYRL, E. K. Parameterized Ray Tracing. In *Computer Graphics (SIGGRAPH '89 Proceedings)* (July 1989), J. Lane, Ed., vol. 23, pp. 307–314.

[23] SHINYA, M., TAKAHASHI, T., AND NAITO, S. Principles and Applications of Pencil Tracing. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 45–54.

[24] SILLION, F., ARVO, J., WESTIN, S., AND GREENBERG, D. A Global Illumination Solution for General Reflectance Distributions. *Computer Graphics (SIGGRAPH '91 Proceedings) 25*, 4 (July 1991), 187–196.

[25] SILLION, F., AND PUECH, C. A General Two-Pass Method Integrating Specular and Diffuse Reflection. *Computer Graphics (SIGGRAPH '89 Proceedings) 23*, 3 (July 1989), 335–344.

[26] SILLION, F., AND PUECH, C. *Radiosity and Global Illumination.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1994.

[27] WALLACE, J., COHEN, M., AND GREENBERG, D. A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods' *Computer Graphics (SIGGRAPH '87 Proceedings) 21*, 4 (July 1987), 311–320.

[28] WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (Aug. 1988), J. Dill, Ed., vol. 22, pp. 85–92.

[29] WHITTED, T. An Improved Illumination Model for Shaded Display. *CACM 23*, 6 (1980), 343–349.