Interpolation over Light Fields with Applications in Computer Graphics*

F. Betul Atalay[†] David M. Mount[‡]

November 1, 2002

There is a growing interest in algorithms and data structures that combine elements of discrete algorithm design with continuous mathematics. This is particularly true in computer graphics. Consider the process of generating a photo-realistic image. The most popular method for doing this is *ray-tracing* [Gla89]. Ray-tracing models the light emitted from light sources as traveling along rays in 3-space. The color of a pixel in the image is a reconstruction of the intensity of light traveling along various rays that are emitted from a light source, transmitted and reflected among the objects in the scene, and eventually entering the viewer's eye. There are many different methods for mapping this approach into an algorithm. At an abstract level, all ray-tracers involve forming an image by combining various continuous quantities, or *attributes*, that have been generated from a discrete set of sampled rays. These continuous attributes include color, radiance (brightness), surface normals, and reflection and refraction vectors. These attributes vary continuously either as a function of the location on the surface of a smooth object or as a function of the location of the viewer and the locations of the various light sources in 3-space.

Producing images by ray-tracing is a computationally intensive process. Scenes can involve hundreds of light sources and from thousands to millions of objects, often represented as smooth surfaces, such as subdivision surfaces, Bezier surfaces and NURBS. Reflective and transparent objects cause rays to be reflected and refracted, further increasing the numbers of rays that need to be traced. In traditional ray-tracing solutions, each ray is traced through the scene as needed to compute the intensity of a pixel in the image. Much of the work involves determining the first object that is intersected by each ray and the location that the ray hits.

In this paper, we propose one approach to help to accelerate this process by reducing the number of intersection calculations. Our algorithm facilitates fast, approximate rendering of a scene from any viewpoint, and is most useful when the scene is rendered from multiple viewpoints, as arises in computing animations. Rather than tracing each input ray to compute the required attributes, we can collect and store a relatively sparse set of *sampled rays* and associate some continuous geometric attributes with each sample in a fast data structure. We can then use inexpensive *interpolation* methods to approximate the value of these sampled quantities for other input rays. Using an adaptive strategy, it is possible to avoid oversampling in smooth areas while providing sufficiently dense sampling in regions of high variation. We dynamically maintain a *cache* of the most recently generated samples, in order to reduce the space requirements of the data structure.

*This material is based upon work supported by the National Science Foundation under Grant No. 0098151.

[†]Department of Computer Science, University of Maryland, College Park, Maryland. Email: betul@cs.umd.edu.

[‡]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. Email: mount@cs.umd.edu.

The information associated with a given ray is indexed according to the directed line that supports the ray, which in turn is modeled as a point in a 4-dimensional *line space*. Given a ray to be traced, we access the data structure to locate a number of neighboring sampled rays in line space. We then interpolate the continuous information from the neighboring rays. The resulting data structure is called an *RI-tree* or *ray interpolant tree*.

Each object can be modeled abstractly as a function f mapping rays to a set of geometric attributes that are used in color computation. The attributes depend on the object's surface reflectance properties. For objects whose surfaces are neither reflective nor transparent, denoted *simple surfaces*, the function returns the point of intersection and the surface normal at this point. For objects whose surfaces are either reflective or transparent the function additionally returns the *exit ray*, that is, the reflected or refracted ray, respectively, that leaves the object's surface. The exit ray is represented by its origin, the *exit point*, and directional *exit vector*. In general, objects may be both reflective and refractive, but our implementation currently does not support this. These quantities are depicted in Fig. 1 and the function is described schematically below.

$$f \; : \; \text{Ray} \rightarrow \{\, \underline{\text{NormalVector}}, \underline{\text{IntersectionPoint}} \,\} \qquad \text{For simple surfaces,}$$
$$f \; : \; \text{Ray} \rightarrow \{\, \text{NormalVector}, \text{IntersectionPoint}, \underline{\text{ExitPoint}}, \underline{\text{ExitVector}} \,\} \qquad \text{Otherwise.}$$

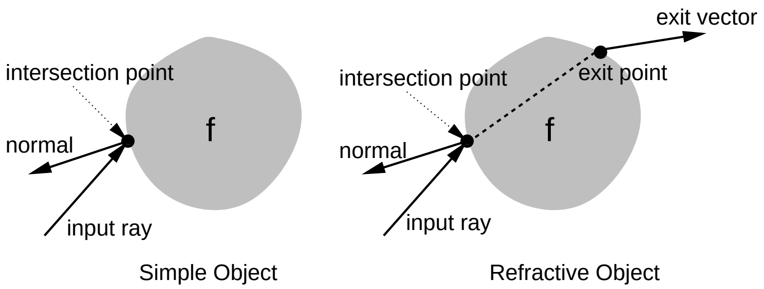


Figure 1: Geometric attributes.

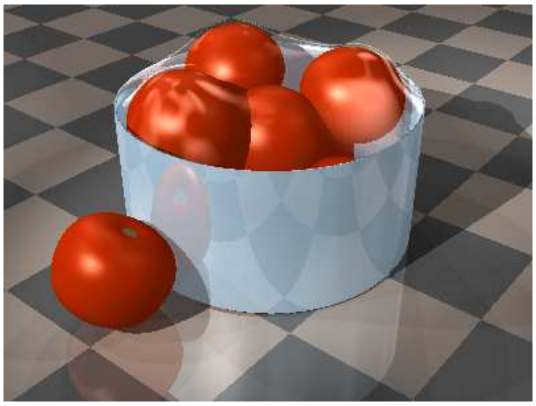
For many real world objects, which have large smooth surfaces, f is expected to vary smoothly. In the context of ray-tracing, this is referred to as *ray coherence*. Nearby rays follow similar paths, hit nearby points having similar normal vectors and hence are subject to similar reflections and/or refractions. In the neighborhood of discontinuities, nearby input rays may follow quite different paths. In cases where we cannot find sufficient evidence to interpolate, we perform ray-tracing instead.

In a traditional ray-tracer, each object is associated with a procedure that computes intersections between rays and this object. For objects whose boundaries are sufficiently smooth, we replace this intersection procedure with our data structure, which is used to approximate the function f through interpolation.

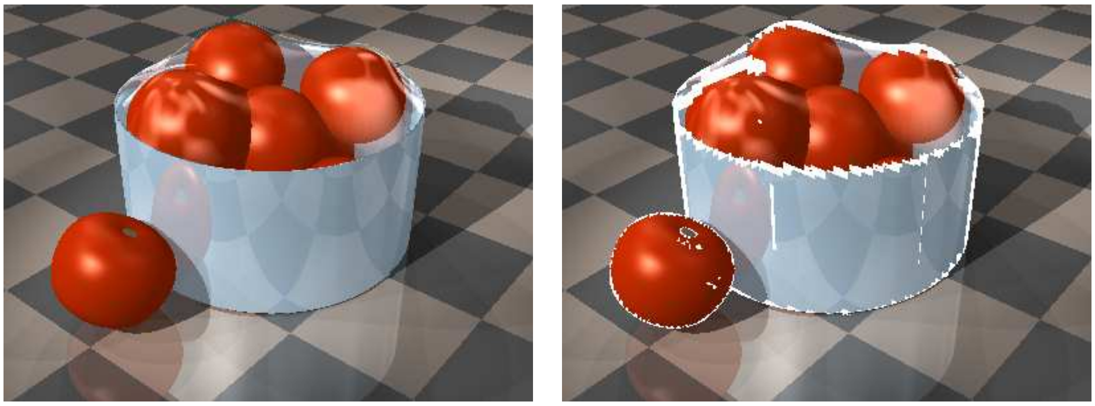
In particular, given objects defined by smooth curved surfaces, the RI-tree can produce high-quality renderings faster than standard methods. We also investigate a number of tradeoffs between the space and time used by the data structure and the accuracy of the interpolation results.

References

[Gla89] A. S. Glassner(editor). *An Introduction to Ray Tracing*. Academic Press, San Diego, 1989.



(a)



(b)

Figure 2: (a) Ray-traced image(The scene consists of a number of tomatoes, modeled as spheres, placed within a reflective bowl, modeled using Bezier surfaces. This is covered by a reflective and transparent but non-refractive plastic wrap modeled by 100 Bezier patches. There is a Bezier surface tomato next to the bowl, and they are both placed on a reflective table within a large procedurally textured sphere. The wrap reflects the procedural texture of the sphere. (b) Interpolated image and corresponding color-coded image, white areas show the ray-traced regions. Speed-up (Floating point operations): 2.40 Memory used: 47 MB