A Note on Virtual Light Fields

Mel Slater
Department of Computer Science
University College London
http://www.cs.ucl.ac.uk/staff/m.slater

Abstract

This short paper describes an attempt to construct a light field for virtual scenes with global illumination. The light field is constructed directly without the necessity for intermediate images produced by another rendering program. The light field is represented as a set of parallel subfields, where each subfield is a set of rectangularly organized parallel rays in some orientation. It is shown that rendering an object into the 4-dimensional ray space can be decomposed into a set of 2-dimensional rasterisations. Radiance is propagated through the light field starting from the emitters, using a method similar to progressive refinement radiosity. Once the light field is illuminated in this way, sets of parallel rays can be projected onto a lens and accumulate radiance onto an image plane. Thus constant time rendering walkthrough can be achieved. The light field so generated is approximate, and the resulting images are poor. However, this note suggests a line of research which may prove useful in the longer term.

1. Introduction

A light field (or lumigraph) [5][8] provides a discrete sample of the distribution of radiance in a space constructed from a set of 2D images with known camera parameters. It is ideally employed in situations where the images are captured by a digital camera from real scenes. A light field for virtual scenes can also be constructed. In this case an existing rendering system may be used to produce the required array of 2D images. In the case of a light field for virtual scenes, if the illumination is local, then there is clearly no problem in producing the required 2D images - for example, OpenGL may be used for this purpose. However, there would really not be much point in doing this. If the illumination is to be global then again a rendering system (such as Radiance [7]) might be used to construct the required images. However, as noted in [5] this may result in weeks of processing time. This note discusses the possibility of directly producing light fields for globally illuminated virtual scenes without the intermediate need to render a large number of 2D images. This is called a *virtual light field* (VLF) approach.

A light field approach for globally illuminated scenes is attractive. It provides the major advantage of image-based rendering - rendering time becomes constant, and independent of the original number of objects in the virtual scene. The cost may be a very large memory requirement, although it is very likely that this will be overcome by the use of compression, as in the original light field papers. The idea presented in this note may be thought of as a combination of light field and illumination network [1]. Both employ the idea of a fixed ray-based data structure which is a discretisation of the distribution of radiance in a space. However, the illumination network carries radiant information in a data structure attached to each object, which includes ‘links’ from object to object. Two objects are linked in this way if there is an unoccluded ray that joins them, and the link is a pointer from one object connecting to the other along such a ray. Since the illumination network carries the information on objects its rendering time is still linearly dependent on the number of objects in the scene. Objects are subdivided into patches, and the illumination network determines the radiance of the patches. It is finally the objects which are rendered. The light field and VLF approach does not render objects at all. Rather the objects illuminate the rays, and the rays are rendered. The VLF might be thought of as a view-independent ray tracer, though it handles diffuse reflection simply, and also diffuse-specular interactions in any combination. However, the technique as described here is also very much an approximation to the global illumination solution.

2. A Parallel Ray Data Structure

The goal is a computationally efficient discrete representation of ‘all rays in a space’. [5] and [8] used a *light slab* approach. Two parallel planes are each tiled by a rectangular grid. Each vertex on one plane is connected by a ray to every vertex on the other plane. Such a pair of parallel planes is called a light slab. If we think of a scene bounded by near-far, left-right, and top-bottom planes, it can be seen that three copies of the light-slab can be used as a ray approximation space (or six copies if each ray represents only one direction). Consider the near-far light slab. If a camera is placed at a tile vertex on the near plane and the image plane is taken as a rectangular subset of the far plane (a rectangle encompassing the scene) then by rendering the scene with pixels corresponding to vertices on the far plane, radiance values may be set for each ray connecting vertices on the far plane to the vertex representing the centre of projection on the near plane. (Issues of sampling corrections are not discussed here). By shifting the centre of projection to each near-plane vertex in turn, all the rays connecting the far plane to the near plane may have their radiance values determined. Images from viewpoints other than the original set of vertices on the near-plane can be reconstructed by interpolation. However, such viewpoints must be outside the convex hull of

the scene, since only rays that are unoccluded between their original centre of projection and their origin in the scene will have formed the images.

This two-plane parameterization is efficient for the construction of images and the reconstruction of new images, for example, the texture-mapping hardware may be easily employed in the construction of new images. However, an alternative approach to illuminating a light field which does not require intermediate 2D images is considered. The purpose of the light field is to act as a discrete representation of all possible rays. It is shown in [2] that a uniform distribution of rays in a space may be constructed as follows, based, say, on a bounding sphere with centre at the origin. Choose a point *p* at random on the surface of the sphere, which also defines a vector *p* from the origin. Consider the plane through the origin and which is orthogonal to this vector. Choose a random set of points with a uniform distribution on the disc formed by the intersection of the plane and the sphere, determining a set of rays through these points parallel to the vector *p*. A uniform distribution of rays is generated by repeatedly randomly choosing such points *p* on the sphere, and following this procedure.

For computational convenience and efficiency consider a cube $[-1, 1]^3$ rather than a sphere. Call this the *scene cube*. The base of the scene cube, parallel to the *xy* plane, is tiled by a rectangular $n \times n$ grid. Each grid-vertex is the origin of a ray parallel to the *z*-axis. Hence we have a cube with interior populated by a regular 2D array of vertical lines. Consider a hemisphere with centre at the origin of the cube, and base on the *xy* plane. A point chosen at random on this hemisphere will determine a direction vector *p* as above. The cube is rotated so that its rays are parallel to *p*. Assuming that each ray can represent two directions, an approximate uniformly distributed set of rays can be generated by repeated random choices for *p*, or by choosing such points which are uniformly distributed on the sphere. Again, for computational efficiency we use a hemi-cube instead of a hemisphere, and tile the top face of the hemi-cube into a rectangular grid with $m \times m$ vertices (with a corresponding tiling on the side faces). Now every ray in this representation can be expressed by (i, j, u, v) where (u, v) represents a direction vector determined by a vertex on the hemi-cube and (i, j) represents the particular ray from the 2D array of rays parallel to this direction.

Given any arbitrary ray (*r*) in the space, an approximating ray can easily be determined: find the direction vector from the hemi-cube which is closest to the direction vector of the ray. This gives the (u, v) component, selecting the set of rays in the representation which are (almost) parallel to the given ray. Now suppose *M* is the rotation matrix that takes a vertical ray into this direction, then apply M^{-1} to the ray *r*, to align it with the vertical field, and then find its nearest vertex on the base of the scene cube. This will determine its (i, j) component.

3. Rendering Objects into the Light Field

The parallel ray representation is a data structure for the construction of the virtual light field. A similar data structure has been used for approximate visibility occlusion in [3]. The important point is that a uniformly distributed set of rays may be thought of as being partitioned into a set of *parallel subfields*. This makes the rendering of objects into the 4-dimensional space of rays, representing the light field, computationally efficient. However, what is meant by 'rendering an object into the light field'? Here a similar approach to that found in the illumination network [1] is used.

A ray segment between points p_0 and p_1 may be represented parametrically as $p_0 + t \cdot dp$ where $dp = p_1 - p_0$. A ray that intersects an object will form a sequence of t-intersections: $0 < t_1 < t_2 < \ldots < t_n < 1$. Rendering an object into the light field means finding such a sequence of t-intersections for each ray that intersects the object. This can be implemented very efficiently. Consider for a moment the vertical subfield (i.e., all rays are parallel to the z-axis of the scene cube in its original orientation). In this case finding the sequence of t-intersections is equivalent to a rasterisation of the object. The pixels correspond to the tiling on the base of the cube, and the t-intersections are z-depths. However, we require all z-depths rather than the closest one. When the object is a polygon this is exactly equivalent to rasterisation including z-depths, which is easily achieved using the rasterisation hardware. Now consider a non-vertical parallel subfield. There will be a matrix that transforms the direction of this subfield back to the vertical. If this matrix is applied to the object, then the object can be rasterised in terms of the vertical subfield. Rendering an object into the light field is therefore almost equivalent to carrying out as many 2D rasterisations of the object as there are rotation directions supplied by the hemi-cube. In the case of a polygon, it is exactly the same as this plus the cost of rotations of the polygon.

In this representation a ray $[i, j, u, v]$ is actually a look-up address of a sequence of t-intersections. Hence the light field is a 4-dimensional array of sequences of t-intersections. However, with each t-intersection additional information must be recorded. The term *T-Intersection* is used to describe the data recorded at each intersection of a ray with an object. It consists of: *(1)* The identifier of the object; *(2)* The t-value at this intersection; (*3)* A direction bit (*opposite* or *same*); (*4)* Unshot radiance; (*5)* Radiance; (*6)* A diffuse reflection radiance accumulator.

The direction bit is used to record the direction of light out of the object. Each ray has a direction (in the vertical subfield from the *xy* plane in the direction of the positive *z*-axis, and all other subfields are determined by rotation from this). Consider two successive T-Intersections T_1 to T_2. along a ray representing a link between objects 1 and 2, with $T_1 \cdot t < T_2 \cdot t$. Then $T_1 \cdot direction = same$ and $T_2 \cdot direction = opposite$ since from the intersection with object 1 light flows to object 2 in the same direction as the ray, whereas for object 2 light flows to object 1 in the opposite direction to the ray.

All objects are rendered into the 4-dimensional light field as the first step. Hence by the end of this process any ray $[i, j, u, v]$ will correspond to a sequence of T-Intersections, with all radiance fields set to zero, but the first three items (the

object identifier, the t-value, and the direction bit) filled in. Starting from objects that are light emitters, radiance can be propagated through the structure in a manner similar to progressive refinement in radiosity [4].

4. Propagating Radiance

Consider a particular object *S* which is an emitter. Using the same method as described above, find all rays that intersect *S*. Suppose *r* is such a ray from p_0 to p_1, and T_i is a T-Intersection along *r* such that $T_i \cdot object = S$. Suppose T_{i+1} is the next T-Intersection along *r* and $T_{i+1} \cdot object = R$. In the general case the BRDF of *R* will determine a set of rays (approximately) through $p = p_0 + p_1(T_{i+1} \cdot t)$, along which the incident radiance from *S* will be reflected. The closest T-Intersections to *p* along these rays will have their radiance and unshot radiance fields set appropriately, and object *R* is marked as containing unshot radiance. All objects which are emitters are treated in this way, after which cycle 0 of the radiance propagation is completed.

Cycle 1 treats all objects which were marked during cycle 0 as containing unshot radiance. Consider such an object *R*. As before all rays intersecting *R* are computed. This is now treated exactly the same as an emitter. For any T-Intersection for *R* there may be a neighboring object R' which is the previous or next T-Intersection along the same ray, defining a segment along which light is reflected from *R*. R' has its radiance and unshot radiance fields set appropriately, and the unshot radiance for *R* is reset to zero.

In general cycle *i* treats all objects which were marked in cycle $i-1$ as containing unshot radiance. The iteration terminates when the unshot radiance falls below a preset threshold, or when the amount of radiance absorbed by the whole system converges. As in the case of progressive refinement radiosity, objects should be treated in descending order of magnitude of unshot radiance. In practice general BRDFs have not been used, but rather objects which are combined ideal specular and diffuse reflectors. Consider an ideal specular reflector. An incident ray will result in one reflected ray. The current implementation finds the closest ray to this reflected ray in the light field, and uses that to propagate the radiance. An alternative is to set the unshot radiance into a number of nearest neighbor rays to the true reflected ray. Glossy reflection can be approximated by considering a cone of rays with apex at the origin of the reflected ray.

A diffuse object can be handled by only two passes in each cycle, where a pass means ‘find all rays through the object’. For *explanatory purposes only*, suppose that the object has been sub-divided into small surface elements. Each element will maintain an accumulated irradiance carried by all rays incident to the surface in the current iteration cycle. A ‘gathering’ phase finds all rays through the object, and accumulates the unshot irradiance into the surface elements. Then a ‘shooting’ phase finds all rays through the object, and propagates out the accumulated irradiance found at the surface elements (which are reinitialized to zero at the end of this phase). However, rather than sub-divide objects into surface elements, the energy is accumulated into an additional field of that T-Intersection corresponding to the normal of the object at the intersection point.

Each T-intersection therefore has an additional field, (*6)* in the list above, which potentially stores two values: an accumulated irradiance, and an iteration cycle number. When an incident ray hits a diffuse reflector, the closest ray in the VLF to the normal at that point is found, and the T-intersection along that ray corresponding to its intersection with the object found. The unshot irradiance is accumulated into that T-intersection. During the shooting phase when a ray intersects the object, the T-Intersection corresponding to the normal at that point is found again, and the required fraction of accumulated irradiance is propagated out along the appropriate interval of the original intersecting ray.

It is necessary to avoid finding all rays through the object for a third time to reinitialize all the accumulated irradiances to zero when an object’s shooting phase is complete. Instead, for each ray intersecting the object during the gathering operation, the current iteration cycle is compared with the one written into the T-Intersection field, and the irradiance is reset to the new value if this does not match the actual iteration cycle (and the stored iteration cycle in the T-Intersection is updated).

5. Rendering

In order to produce images a virtual ‘eye’ can be placed anywhere in the scene at any viewing orientation. The eye consists of a lens (or any arrangement of lenses) and an image plane. Consider the simplest case of a single convex thin lens. The lens is represented as a rectangular planar polygon, and all rays through the lens can be trivially and rapidly generated. The T-Intersection corresponding to any ray striking the lens can be looked up. If the T-Intersection carries radiance in the appropriate direction (to inside the lens) then the lens formula may be used to compute the refracted ray, which can then be intersected with the image plane, and the radiance along the ray is accumulated on the image plane pixels. For a single convex lens, all rays in a parallel subfield will be refracted through a single point on the focal plane. The image plane can be mapped to a 2-dimensional window on a workstation display.

The lens properties (focal length etc.) can be altered as desired between frames at no cost. The time for this operation of capturing an image onto the image plane through a lens is essentially constant for any eye position anywhere in the scene. It is equivalent to rasterisation of one polygon (the lens) as many times as there are rotations of the parallel subfield plus the cost of the rotations.

6. Discussion

A preliminary implementation of the method described above has been completed. Initially a two-plane parameterization was considered. This ideally has a greater resolution on the image (far) plane of the light slab, than on the near plane which

will contain the vertices for the camera positions. This is because of the need to obtain a high sampling rate for scene objects. The method can afford a lower sampling of viewpoints because of the effectiveness of the interpolation. Moreover, the viewpoints are all around the outside of the scene. However, for the virtual light field, there are no especially preferred viewpoints so that the same high resolution on both planes would be required.

The $n \times n$ partition of the base of the cube in the VLF approach determines the sampling accuracy of objects. This has to be as high as possible, of course it also determines the number of rays that will hit the lens during the image formation. The $m \times m$ partition of the hemi-cube determines directional sampling accuracy, which has implications for accuracy of visibility and also of course for the illumination propagation. Nevertheless some encouraging images can be generated with low levels of m. Figure 1(a) and (b) shows a scene generated with $n = 300$ and $m = 16$. The scene consists of an open box, with a spherical light at the far end and a sphere which is a specular and diffuse reflector. The right hand wall has some specular reflection, all the others are diffuse reflectors. The iteration converged after 5 cycles, taking just over 30 minutes on an Onyx 2, and it used about 2 Gb memory! Each image took about 2 seconds to generate, on a 300×300 display window. Reflections and shadows can be observed. Figure 1(c) uses $n = 1000$ and $m = 8$, and all walls are specular. It took 1.25 hours to converge, and used about 5.5Gb memory. Note that multiple reflections of reflections can be seen. No use has been made of the underlying rasterisation hardware, and the implementation was not memory efficient (the memory requirement could be cut by half with some recoding). Moreover, no attempt has been made to introduce a sampling scheme, these results are raw, a direct implementation of the method as described.

This note has reported early work on the production of a virtual light field. The research is continuing in the construction of a compression scheme, the introduction of an appropriate set of basis functions for sampling, and further consideration of the fundamental discretisation.

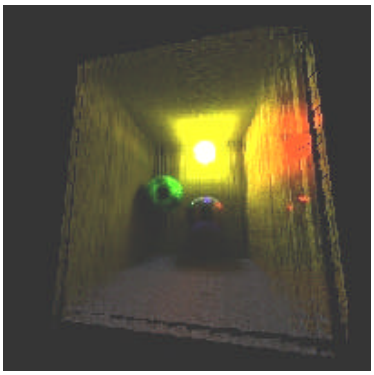
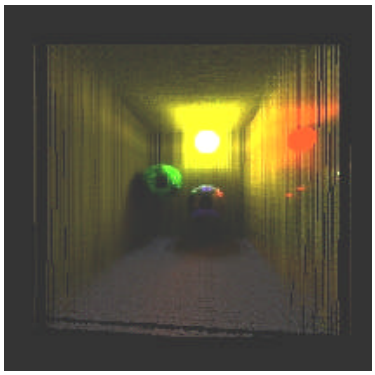
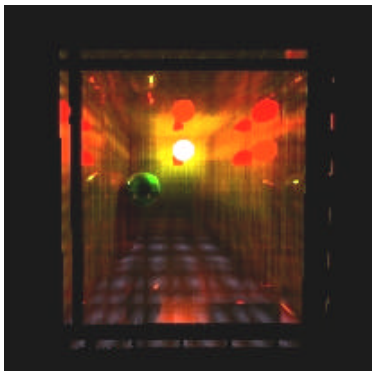
(a)  (b)  (c) 

Figure 1 Spheres inside an open box. In (a) and (b) one wall is specular. In (c) all walls are specular, and multiple reflections of reflections can be seen.

Acknowledgments

This work is supported by the UK EPSRC Senior Research Fellowship programme. Thanks to Dr. Yiorgos Chrysanthou for comments, and to Prof. F. Brooks Jnr for his early encouragement of this line of research.

References

[1] Buckalew, C. and Fussell, D. (1989) Illumination Networks: Fast Realistic Rendering with General Reflectance Functions, Computer Graphics (SIGGRAPH), 23(3), 89-98.

[2] Camahort, E., Lerios, A. Fussell, D. (1998) Uniformly Sampled Light Fields, Proceedings of the 9th Eurographics Workshop on Rendering, Vienna, Austria, June/July.

[3] Chrysanthou, Y., Daniel Cohen-Or and Dani Lischinski (1998) Fast Approximate Quantitative Visibility for Complex Scenes, Computer Graphics International '98, Hannover, Germany, June 1998, 220-227.

[4] Cohen, M.F., Shenchang, Ec., Wallace, J.R. and Greenberg, D.P. (1988) A Progressive Refinement Approach to Fast Radiosity Image Generation, Computer Graphics 22(4), 75-84.

[5] Gortler, S., Grzeszczuk, R., Szeliski, R., Cohen, M. (1996) The Lumigraph, Computer Graphics (SIGGRAPH), Annual Conference Series, 43-52.

[6] Lalonde, P. and Fournier, A. (1999) Interactive Rendering of Wavelet Projected Light Fields, Proceedings of Graphics Interface '99, 107-114.

[7] Larson, G. W. and Shakespeare, R. (1997) Rendering with Radiance: The Art and Science of Lighting Visualization, Morgan Kaufmann Publishers, San Francisco, CA.

[8] Levoy M and Hanrahan, P. (1996) Light Field Rendering, Computer Graphics (SIGGRAPH), Annual Conference Series, 31-42.