Real-time Reflection using Ray Tracing with Geometry Field

Shengying Li, Zhe Fan, Xiaotian Yin, Klaus Mueller, Arie E. Kaufman, Xianfeng Gu

Department of Computer Science, Center for Visual Computing, Stony Brook University, U.S.A.

Abstract

A novel method for accurate reflections in real time is introduced using ray tracing in geometry fields, which combine light fields with geometry images. The geometry field of a surface is defined ray space, mapping a ray to its intersection point with the surface, represented as the uv coordinates on the geometry image. It makes intersection tests much more efficient using lookups within a fragment shader, and allows conventional textures to be applied, such as normal maps, to enhance geometric subtleties. Our method can be generalized to also handle refraction and self-reflection. Experimental results demonstrate its capability for accurate reflection for complex scenes in real-time.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Raytracing

1. Introduction

Inter-reflection among mirror objects has always been of great interest in computer graphics. Handling reflections with high accuracy in real time remains a challenge, due to the expensive global computational process, especially for the intersection testing, which is not suitable for graphics hardware.

Ray tracing is the most well known algorithm for achieving accurate renderings. Great efforts have been spent to accelerate ray-tracing by designing special hardware [WSS05] [PBMH02, CHH02, WSE04]. Environment mapping [BN76, Gre86] is an alternative approach that provides fast reflection of the surroundings assuming the scene elements are infinitely far away from the reflector. Localized environment maps [SKALP05] offer better approximations for flat surfaces.

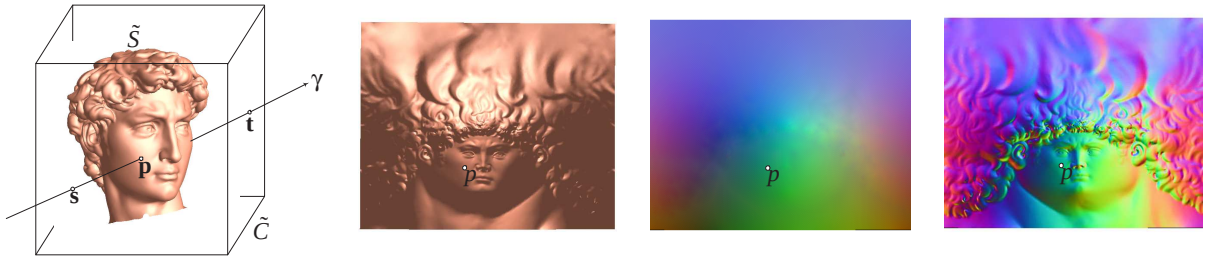
Inspired by the work of light fields and geometry images, we have developed a novel method, *geometry field*, which converts intersection testing to table lookup. Hence, the computational power of graphics hardware can be fully utilized. Without sophisticated data structures and optimizations, our simple algorithm has demonstrated the capability of real time rendering of inter-reflections. Our method responds to the trend of fast increases in memory capacities. In our experiments, we obtain convincing results using current graphics hardware.

The light field [LH96, GGSC96] is one of the central concepts in image based rendering, which maps a ray to the color of the intersection point. Surface reflections have been rendered using light fields [YYM05], layered light fields and layered depth images [LR98], and surface light fields [WAA*00]. Ray classification [AK87] records potentially intersecting primitives in each cell of the 5D ray space discretization, and the virtual light field [SMKY04] extends the idea and uses a light field data structure to propagate radiance for globally illuminated scenes.

The geometry image [GGH02] represents a surface as an image. The color at a pixel represents the position vector of the surface. Similarly, a normal map represents the normals of a surface by an image. Geometry images are generalized to multi-chart geometry images [SWG*03], and smooth geometry images [LHSW03].

The geometry field of a surface S is a map, where the input is an incident ray, and the output is the texture coordinates of the first intersection point. Suppose S is represented as a geometry image $\mathbf{r}(u,v)$, γ is a ray in $\mathbb{R}^3$ intersecting S. The first intersection point p is $\mathbf{r}(u_p,v_p)$. Then the map $G:\gamma\rightarrow(u_p,v_p)$ is the geometry field of surface S. From (u_p,v_p), the position and the normal of p can be obtained from the geometry image and the normal map.

Conventional expensive intersection testings are replaced by simple table lookups in a precomputed geometry field.



(a) Original surface in $\mathbb{R}^3$ (b) Texture parameter domain (u,v) (c) Geometry image $\mathbf{r}(u,v)$ (d) Normal map $\mathbf{n}(u,v)$

Figure 1: *Geometry Field. A ray* γ *in* $\mathbb{R}^3$ *intersects a surface* $\tilde{S}$ *at a point* **p** *as shown in (a).* $\tilde{C}$ *is the bounding box of the surface,* **s** *and* **t** *are the intersection points of* γ *with* $\tilde{C}$. **p** *has a unique texture coordinates* (u,v) *as shown in (b). The position of* **p** *and the normal at* **p** *can be obtained from the geometry image (c) and (d), using the texture coordinates* (u,v). *A geometry field of a surface* $\tilde{S}$ *is map, the input is a ray* γ *in* $\mathbb{R}^3$, *the output is the texture coordinates* (u,v) *of the intersection point* **p** *between* γ *and* $\tilde{S}$.

Precomputing has been broadly applied for illumination in [BF89] [SKSU05], animation in [BS96] and displacement map in [WWT*04].

The geometry field representation has several advantages. First, it effectively reduces the memory cost. In general, a geometry field is 4 dimensional. If the position, normal and color material information are stored for each entry, the storage requirement will be extremely high. Instead, only the texture coordinates are stored, such that the size of a geometry field is within the memory capacity of current hardware. Second, due to the regular structure of a geometry field, it can be represented as a generalized texture. The whole rendering algorithm can be implemented in common graphics hardware. The time cost is independent of the geometric complexity of the scene, and only dependent on the size of the geometry field. Figure 1 illustrates the concept of the geometry field.

Most real-time reflection methods make assumptions of the geometries of the reflectors or their geometric relations to the scene. In contrast, the geometry field handles arbitrary surfaces and scenes without making any assumptions. In experiments, we constructed geometry fields for complicated surfaces, such as Michelangelo's David head model with $100K$ faces. The inter reflections are computed using the GPU at $60FPS$. The method can be generalized for self-reflection and refraction as well.

2. Geometry Field

The geometry field for a surface is a map which records all ray/surface intersection information. It reduces the cost of intersection testing by table lookup in run time. An implementation on graphics hardware can achieve high parallelism. The efficiency can be further improved by the regularity of geometry images, which leads to large spatial coherence.

In our framework, all surfaces are represented as geometry images. General triangular meshes are converted to geometry images by conformal parameterizations, which help to reduce the distortions. Figure 1 demonstrates the geometry image of the David head surface. A ray is represented by its two intersection points (in,out) with the bounding box of the surface. Hence, the geometry field is a 4 dimensional function.

To construct the geometry field, we first construct a bounding box of the geometry image, then compute the intersection points of rays with the box. We only consider rays with two intersection points, the entrance s and the exit t and parameterize them using the point pair (s,t). Next, we compute the intersection point of each ray with the geometry image and record the texture coordinates of the one closest to s. In order to speed up the intersection calculation, we use an octree structure to subdivide the cube. All the computations are carried out in the local coordinates of the geometry image. The result is a $4D$ lookup table and the internal representation on the GPU is described in the next implementation section.

3. Rendering with Inter-reflection

Geometry fields and geometry images are generated offline in software. The run time rendering algorithms have been implemented on a NVidia's GeForce 7800 graphics card.

3.1. Data Structures

Each geometric object O_i is represented as a geometry field GF_i and a geometry image GI_i, which incorporates the position, normal, material and texture information.

Furthermore, each geometric object is normalized in the unit cube of the bounding box. Points on the 6 faces of the unit cube are sampled uniformly and enumerated. So, the $4D$ geometry field is stored as a $2D$ texture, with (i,j) representing the indices for the in and out points on the unit cube. Each geometric object has a geometry field, which is stored as a single $2D$ texture. Each object may have multiple instantiations sharing the geometry field with different local frames.

3.2. Algorithm Pipeline

Non-reflective surfaces are simply rendered using conventional OpenGL fixed pipeline functionalities. Reflective surfaces are rasterized with our geometry field-based vertex shader and fragment shader. The vertex shader is conventional but the fragment shader is capable of computing the inter-reflection. We adapt simple convolution kernels (currently linear) for interpolation when looking up in a geometry field. The algorithm pipeline composes the following steps:

1. Calculate the eye ray in world coordinates.
2. Calculate the reflection ray γ in world coordinates.
3. Calculate the local shading $\mathbf{c}$ of the reflector surface.
4. *Trace ray* γ to get reflected color $\mathbf{c}_r$, blend it with the local shading $\mathbf{c}$.

The algorithm for tracing a ray is as follows:
Color TraceRay(Ray γ, ObjectId i, int depth)

1. For each object $O_j, i \neq j$ in the scene,
   a. Transform the incoming ray starting point and the ray direction from the world coordinate to the local frame of O_j.
   b. Check if the ray hits the bounding box of O_j (the unit cube).
   c. If the ray hits the box at points s and t, convert s and t to the indices of the geometry field GF_j.
   d. Lookup the geometry field GF_j to obtain the texture coordinates (u_j, v_j) of the intersection point.
   e. If (u_j, v_j) is valid, lookup the position from the geometry image GF_j, compute the depth d_j. Otherwise, set $d_j = \infty$.
2. Choose the nearest $(u_k, v_k), k = min_j\{d_j\}$. If (u_k, v_k) is valid, look for the position, color, and normal of the intersection point from the geometry image GI_k, compute the shading and blend it to the local color $\mathbf{c}$, and compute the next level reflected ray γ_r.
3. If the recursion depth exceeds the limit, return $\mathbf{c}$. Otherwise, call TraceRay(γ_r, k, depth+1) to get the reflected color $\mathbf{c}'_r$.
4. Blend $\mathbf{c}$ and $\mathbf{c}_r$. Return the blended color.

4. Experimental Results

This section demonstrates the results for our algorithm. The experiments are applied on the windows platform with a $3.6GHz$ CPU, $3G$ RAM and NVidia's GeForce 7800 GPU.

4.1. Preprocessing

We have tested our algorithms on triangular meshes, whose complexities are up to tens of thousands of faces. The complexity information and the performance of our algorithms are detailed in Table 1. All geometry fields are of size of $(24 \times 24 \times 6)^2$.

4.2. Real Time Rendering

In our real-time rendering system, the camera position, the relative positions of reflected objects and mirror objects can be monitored interactively by users. Figure 2 shows the rendering snapshots for inter-reflection among different geometries. The frame rates are all around $60FPS$, which are independent of the geometric complexities.

In the first row, the first two figures show multiple David heads reflecting on a mirror torus. Because the mirror distortion is directly calculated without approximation, all multiple reflection images are changing continuously on the torus in our experiments. The second figure also illustrates that geometric features of David head's hair in the mirror are rendered with high fidelity. The next two figures of the first row present that multiple instances of David head reflect on a mirror ball.

Inter-surface reflections between two mirrored balls are illustrated in the first two figures of the second row in Figure 2. From the snapshots, the mirror image of the red sphere is shown on the blue sphere. In the center area of the mirror image, the mirror image of the blue ball on the red ball is also recognizable. This demonstrates the second level inter-reflection. The last two figures show inter-reflection between sphere-David head and male-female faces.

5. Conclusions and Future Work

This paper has proposed a novel algorithm for real-time inter-reflection by ray tracing based on a new object representation, geometry field. A geometry field is a combination of a light field with a geometry image, and it represents the intersection point of a surface with an arbitrary ray as a $4D$ lookup table.

Conventional intersection testing in ray tracing is replaced by indexing a geometry field, such that real time inter-surface reflection is achieved on current graphics hardware. The method is efficient, accurate and simple. Experiments have shown the high quality of the reflection results and the fast rendering speed in several examples.

Applications of geometry fields are not limited to reflection. It can be easily generalized for computing refraction and self reflection. In the future, we will do research on full-featured ray-tracing based on geometry fields in real time.

Table 1: *Performance (in sec) of our preprocess algorithm.*

Surface	Triangular Faces	Parameterization	Geom. Image	Geom. Field
Sphere	1,922	15	0.8	420
Torus	7,938	34	2.2	420
Female Face	50,000	130	9.5	432
Male Face	80,072	200	16	504
David Head	101,144	250	20	672

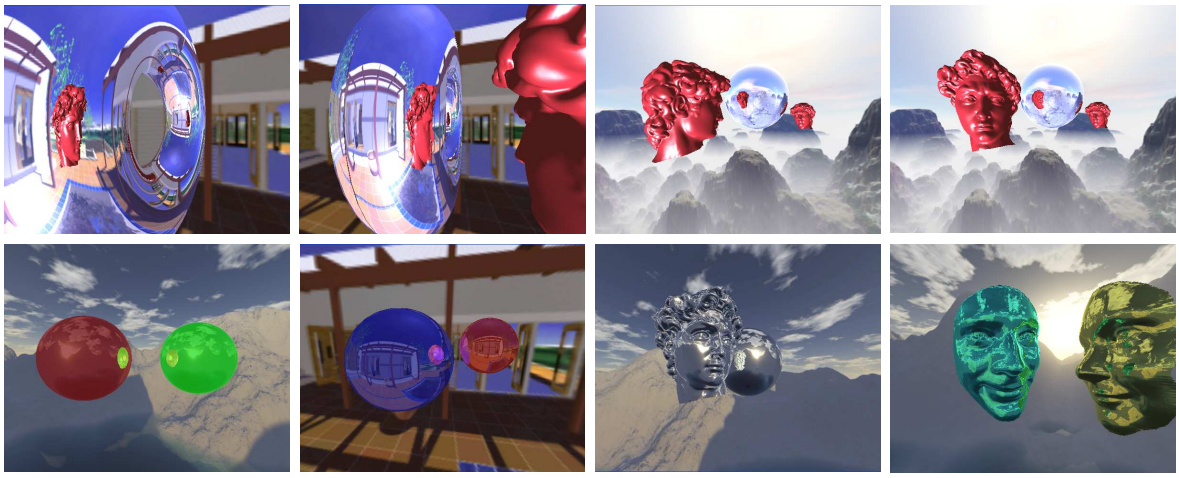


Figure 2: ***Real time rendering results***. *Users can interactively manipulate the camera position and the relative spacial relations among the geometric objects in the scene. The inter-reflections are rendered in real time at* 60*FPS.*

Acknowledgement

This work has been partially supported by NSF grant ACI-0093157, CCR-0306438, NIH grant 5R21EB004099-02 and NSF CAREER CCF0448399.

References

[AK87] ARVO J., KIRK D.: Fast ray tracing by ray classification. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 55–64.

[BF89] BUCKALEW C., FUSSELL D.: Illumination networks: Fast realistic rendering with general reflectance functions. *SIGGRAPH '89 Proceedings 23*, 3 (1989), 89–98.

[BN76] BLINN J. F., NEWELL M. E.: Texture and reflection in computer generated images. *Commun. ACM 19*, 10 (1976), 542–547.

[BS96] BESUIEVSKY G., SBERT M.: The multi-frame lighting method - a Monte-Carlo based solution for radiosity in dynamic environments. In *Rendering Techniques '96* (1996), pp. pp 185–194.

[CHH02] CARR N. A., HALL J. D., HART J. C.: The ray engine. In *ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), pp. 37–46.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *SIGGRAPH* (2002), pp. 355–361.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *SIGGRAPH* (1996), pp. 43–54.

[Gre86] GREENE N.: Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11 (1986), 21–29.

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH* (1996), pp. 31–42.

[LHSW03] LOSASSO F., HOPPE H., SCHAEFER S., WARREN J. D.: Smooth geometry images. In *Symposium on Geometry Processing* (2003), pp. 138–145.

[LR98] LISCHINSKI D., RAPPOPORT A.: Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques* (1998), pp. 301–314.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. In *SIGGRAPH* (2002), pp. 703–712.

[SKALP05] SZIRMAY-KALOS L., ASZÓDI B., LAZÁNYI I., PREMECZ M.: Approximate ray-tracing on the gpu with distance impostors. In *Eurographics* (2005).

[SKSU05] SZIRMAY-KALOS L., SBERT M., UMMENHOFFER T.: Real-time multiple scattering in participating media with illumination networks. In *Proceedings of Eurographics Symposium on Rendering 2005* (June 2005), Eurographics.

[SMKY04] SLATER M., MORTENSEN J., KHANNA P., YU I.: A virtual light field approach to global illumination. *cgi 00* (2004), 102–109.

[SWG*03] SANDER P. V., WOOD Z. J., GORTLER S. J., SNYDER J., HOPPE H.: Multi-chart geometry images. In *Symposium on Geometry Processing* (2003), pp. 146–155.

[WAA*00] WOOD D. N., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D., STUETZLE W.: Surface light fields for 3d photography. In *SIGGRAPH* (2000), pp. 287–296.

[WSE04] WEISKOPF D., SCHAFHITZEL T., ERTL T.: Gpu based nonlinear ray tracing. 625–634.

[WSS05] WOOP S., SCHMITTLER J., SLUSALLEK P.: Rpu: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph. 24*, 3 (2005), 434–444.

[WWT*04] WANG L., WANG X., TONG X., LIN S., HU S., GUO B.: View-dependent displacement mapping. *ACM Transactions on Graphics 22*, 3 (2004), 334–339.

[YYM05] YU J., YANG J., MCMILLAN L.: Real-time reflection mapping with parallax. In *ACM SIGGRAPH 2005 Symposiumon Interactive 3D Graphics and Games* (2005).