Perturbation Methods for Interactive Specular Reflections

Min Chen, James Arvo

Abstract— **We describe a new approach for interactively approximating specular reflections in arbitrary curved surfaces. The technique is applicable to any smooth implicitly-defined reflecting surface that is equipped with a ray intersection procedure; it is also extremely efficient as it employs local perturbations to interpolate point samples analytically. After ray tracing a sparse set of reflection paths with respect to a given vantage point and static reflecting surfaces, the algorithm rapidly approximates reflections of arbitrary points in 3-space by expressing them as perturbations of nearby points with known reflections. The reflection of each new point is approximated to second-order accuracy by applying a closed-form perturbation formula to one or more nearby reflection paths. This formula is derived from the Taylor expansion of a reflection path and is based on first and second-order path derivatives. After preprocessing, the approach is fast enough to compute reflections of tessellated diffuse objects in arbitrary curved surfaces at interactive rates using standard graphics hardware. The resulting images are nearly indistinguishable from ray traced images that take several orders of magnitude longer to generate.**

Keywords— **animation systems, illumination effects, implicit surfaces, matting and compositing, optics, ray tracing**

I. Introduction

COHERENT reflection from surfaces is an important optical effect that has long been a challenging problem for computer graphics, where it is better known as *specular reflection*. The two principal challenges inherent in the simulation of specular reflections are speed and realism, where realism requires both geometric and radiometric accuracy. By geometric accuracy, we mean both shape and location of the reflection. Radiometric accuracy includes both the directly observed brightness, or *radiance*, of the reflection, as well as its indirect illumination effects, of which the most pronounced are caustics [1], [2].

The most general and accurate means of simulating specular reflections is ray tracing, especially when curved surfaces or multiple interreflections are involved. Even indirect lighting effects involving specular reflection can be accurately simulated via Monte Carlo methods that are based on ray tracing [2]. However, full ray tracing is generally impractical for interactive applications despite extensive work on ray tracing acceleration schemes [3]. A popular alternative to ray tracing is environment mapping [4], which can produce convincing visual effects that mimic specular reflection with far less computation. Recently, Cabral et. al [5] combined radiance environment maps with image-based rendering in reflection space for interactive viewing of arbitrary objects with a class of reflectance functions. Unfortunately, environment mapping provides a good approximation only when reflected objects are static and far from the reflector. When these conditions are violated, the results are of very poor accuracy. In contrast to environment mapping, Mitchell and Hanrahan [1] made use of Fermat's principle to compute exact reflection paths from curved surfaces defined by implicit functions. They found the reflection points by solving a non-linear system numerically, which is a computationally expensive procedure.

Both authors are from the California Institute of Technology. E-mail: {chen,arvo}@cs.caltech.edu.

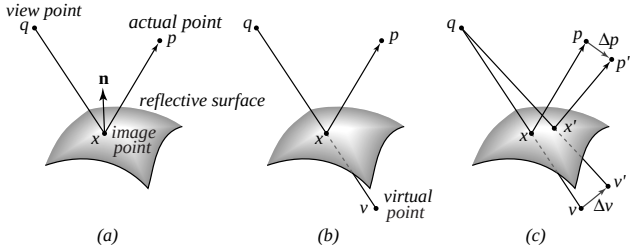


Fig. 1. *Three different reflection problems. (a) Computing the point p as seen reflected at x. (b) Computing the virtual point v corresponding to the actual point p. (c) Computing how the position of the virtual point v changes as the actual point p is perturbed.*

Other approaches have been proposed in which specular reflections of entire objects are computed at once by constructing a *virtual object* rather than forming the reflection pixel-by-pixel. In such an approach, the scene is rendered by explicitly constructing the virtual objects and then handling them much as ordinary objects. It is the latter class of techniques that is most amenable to interactive applications. The idea of using virtual objects to simulate specular reflections has been explored by a number of researchers. Panduranga [6] described how several hidden surface algorithms could be enhanced to include reflections in curved surfaces, essentially by accommodating virtual objects. Rushmeier and Torrance [7] used the virtual object idea in the context of radiosity to handle specular reflections from planar surfaces. Taking advantage of graphics hardware, Ofek and Rappoport [8] used the virtual object method to interactively approximate single-level reflections on curved objects which are polygonal meshes or linear extrusions. In their method, virtual vertices are computed by a space subdivision and an "explosion map" acceleration scheme.

Fig. 1 depicts three different strategies for simulating specular reflections: ray tracing (left), virtual objects (middle), and our approach (right), which combines aspects of both ray tracing and virtual objects. The key ideas behind the three different approaches can be characterized by the type of question they attempt to answer. In the ray tracing approach, one asks: Given a viewpoint and a point on a reflecting surface, what point of the environment is

seen reflected there? However, if we drive the rendering based on the objects of the scene and not the pixels, we ask a different question: Given a viewpoint and a point in 3-space, where does its reflection appear? This is the question Mitchell and Hanrahan [1] confronted in rendering caustics. Unfortunately, the latter question is much more difficult to answer for several reasons. First, the answer may not be unique. That is, a single point may have many reflections, even in a single smooth surface. See Fig. 8. Secondly, while the reflected point is typically easy to express analytically using only the surface normal at the reflection point (and standard ray-intersection procedures), finding the position of the reflection point, or the virtual point usually requires numerical approximation, even for very simple reflecting geometries.

Our approach is based on a third type of question: How does the reflection of an object move when the object moves? The rationale for considering this form of reflection problem is that specular reflections in smooth surfaces vary continuously as a function of object position, except when occlusion or boundary conditions intervene. Moreover, when expressed in terms of differential motions, this question can be answered *analytically* even for very complex geometries. Previous work on exploring differential movements of this nature includes Ward and Heckbert's irradiance gradient for higher-order interpolation [9], and Igehy's ray differential for texture filtering [10]. In this paper we describe an algorithm for rapidly computing highly accurate reflections in curved specular surfaces by exploiting this third category of reflection problem. The method is fast enough to interactively update the specular reflections on static reflective surfaces while moving diffuse objects.

By characterizing how specular reflections change as a result of perturbing an object with a known reflection, we may rapidly approximate a family of very similar reflections from a single nearby reflection. The central idea behind our approach is to approximate the reflection of any point in 3-space by viewing it as a perturbation of a known reflection. Thus, from a sparse set of known reflections, we can quickly construct any reflection at all. In effect, this is an interpolation method for reflections, as it allows a continuous reflection to be constructed from discrete samples while preserving the exact behavior at those samples. From these interpolated reflections, virtual objects are constructed and rendered along with real objects using graphics hardware supporting alpha blending and z-buffering. This hardware rendering is most similar in spirit to the work of Ofek and Rapport. Our most significant point of departure is in the use of perturbation theory [11] to accurately locate the position of the virtual object. To that end, one of the central contribution of this paper is the use of a closed-form expression for perturbing specular reflections. By using second-order approximation, we can attain extremely faithful reflection geometry.

In the next section, we present the general idea of our path perturbation theory and present the closed-form formulas for first- and second-order perturbations of a specular reflection path. Detailed derivation of these formulas are given by Chen [12] and Chen and Arvo [13]. Section III is devoted to the description of an algorithm for rapidly approximating specular reflections on arbitrary curved surfaces, with a focus on the computation of virtual objects using path perturbation formulas shown in Section II. Results and further discussion of this perturbation approach are demonstrated in Sections IV and V. We conclude with some future directions.

II. Specular Path Perturbation

Our perturbation method is motivated by the fact that reflections tend to have a great degree of coherence; in general, as two objects $\mathbf{p}$ and $\mathbf{p}'$ grow nearer to each other, as shown in Fig. 1c, so will their reflections $\mathbf{x}$ and $\mathbf{x}'$. Here coherence means local smoothness of specular reflections. That is, given a known reflection path from a fixed point $\mathbf{q}$ to $\mathbf{p}$ via a smooth surface, it is likely that the reflection point $\mathbf{x}$ will vary continuously as a function of the position of $\mathbf{p}$, that is, $\mathbf{x} = \Psi(\mathbf{p})$, where the smooth function $\Psi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is called the *path function*. The essence of specular path perturbation is to approximate families of closely related reflection paths using a high-dimensional Taylor expansion of Ψ around a given path. In this way, the reflection from $\mathbf{q}$ to a new point $\mathbf{p}'$ in the neighborhood of $\mathbf{p}$ can be obtained by perturbing $\mathbf{x}$ analytically.

For a vector-valued function Ψ with three components Ψ_1, Ψ_2 and Ψ_3, the second-order Taylor expansion can be expressed as:

$$\Psi(\mathbf{p}+\Delta\mathbf{p}) = \Psi(\mathbf{p})+\mathbf{J}\Delta\mathbf{p} + \frac{1}{2}\Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{p}+\mathrm{O}(||\,\Delta\mathbf{p}\,||^3), \quad (1)$$

which is the *perturbation formula* to update a given path through $\mathbf{p}$ to a new path reaching the neighboring point $\mathbf{p}+\Delta\mathbf{p}$. Here, the first-order derivative $\mathbf{J}$ is a 3×3 matrix defined by

$$\mathbf{J} \equiv \frac{\partial\Psi(\mathbf{p})}{\partial\mathbf{p}} = \begin{bmatrix} \Psi_{1,1} & \Psi_{1,2} & \Psi_{1,3} \\ \Psi_{2,1} & \Psi_{2,2} & \Psi_{2,3} \\ \Psi_{3,1} & \Psi_{3,2} & \Psi_{3,3} \end{bmatrix}, \quad (2)$$

where $\Psi_{i,j} \equiv \partial\Psi_i/\partial\mathbf{p}_j$. The second-order derivative $\mathbf{H}$ is a third-order tensor, which consists of three Hessian matrices $\mathbf{H}^1, \mathbf{H}^2, \mathbf{H}^3$ corresponding respectively to coordinates x, y and z, given by

$$\mathbf{H}^i \equiv \frac{\partial^2\Psi_i(\mathbf{p})}{\partial\mathbf{p}^2} = \begin{bmatrix} \Psi_{i,11} & \Psi_{i,12} & \Psi_{i,13} \\ \Psi_{i,21} & \Psi_{i,22} & \Psi_{i,23} \\ \Psi_{i,31} & \Psi_{i,32} & \Psi_{i,33} \end{bmatrix} \quad (3)$$

for $i = 1, 2, 3$, where $\Psi_{i,jk} \equiv \partial^2\Psi_i/\partial\mathbf{p}_j\partial\mathbf{p}_k$. The term $\Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{p}$ in equation (1) denotes a vector $\big(\Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}^1\Delta\mathbf{p},\ \Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}^2\Delta\mathbf{p},\ \Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}^3\Delta\mathbf{p}\big)$. We call $\mathbf{J}$ and $\mathbf{H}$ the *path Jacobian* and the *path Hessian*, which provide the first- and second-order approximations to the path function, respectively.

To extend equation (1) to a general path with N bounces, we order the reflection points from the perturbed point $\mathbf{p}$ to the fixed point $\mathbf{q}$ as $\mathbf{p} = \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\mathrm{N}}, \mathbf{q} = \mathbf{x}_{\mathrm{N}+1}$, and

consider each reflection point $\mathbf{x}_i$ as a function of the previous point $\mathbf{x}_{i-1}$. By computing the first- and second-order derivatives of this function with respect to $\mathbf{x}_{i-1}$, which correspond to a Jacobian matrix $\mathbf{J}_i$ and a Hessian tensor $\mathbf{H}_i$ with forms similar to (2) and (3), we can perturb the entire path recursively, beginning at $\mathbf{p}$ and stepping toward $\mathbf{q}$. That is,

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} + \frac{1}{2}(\Delta\mathbf{x}_{i-1})^{\mathrm{T}} \mathbf{H}_i \Delta\mathbf{x}_{i-1}, \tag{4}$$

where $i = 1, 2, 3, \ldots, N$, $\Delta\mathbf{x}_{i-1} = \mathbf{x}'_{i-1} - \mathbf{x}_{i-1}$, $\Delta\mathbf{x}_0 = \mathbf{p}' - \mathbf{p}$ and the tensor product $(\Delta\mathbf{x}_{i-1})^{\mathrm{T}} \mathbf{H}_i \Delta\mathbf{x}_{i-1}$ is defined as before.

The closed-form formula for path Jacobians is derived from the *Fermat principle* [14], *Lagrange Multiplier Theorem* [15], and the *Implicit Function Theorem* [16]. Based on the expression for path Jacobians, path Hessians can be computed via tensor differentiation. We provide an outline of the derivations and major results below. Interested readers can refer to Chen and Arvo [13] and Chen [12] for further details.

It follows from the Fermat principle of shortest distance that the reflection point $\mathbf{x}$ is a point lying on the reflective surface that locally minimizes the optical path length from $\mathbf{p}$ to $\mathbf{q}$. Given an implicit function $g(\mathbf{x}) = 0$ defining the reflecting surface, we can apply the method of Lagrange Multipliers to the resulting constrained optimization problem [1] to obtain a non-linear system in the case of one-bounce reflections. That is, we must solve

$$\begin{aligned} -\frac{(p_i - x_i)}{\|\,\mathbf{p} - \mathbf{x}\,\|} - \frac{(q_i - x_i)}{\|\,\mathbf{q} - \mathbf{x}\,\|} + \lambda \frac{\partial g(\mathbf{x})}{\partial x_i} &= 0 \\ g(\mathbf{x}) &= 0, \end{aligned} \tag{5}$$

where $i = 1, 2, 3$. We denote it by $F(\mathbf{p}, \mathbf{x}, \lambda) = \mathbf{0}$, where F has the four components (F_1, F_2, F_3, F_4) shown in equation (5). From the Implicit Function Theorem, we know that there exists an explicit function $f : \mathbf{p} \to (\mathbf{x}, \lambda)$ within a neighborhood of the given path, provided that the 4×4 Jacobian matrix

$$\left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial(\mathbf{x}, \lambda)}\right] \equiv \left[\frac{\partial F}{\partial \mathbf{x}} \;\; \frac{\partial F}{\partial \lambda}\right]$$

is nonsingular. Moreover, the path Jacobian $\mathbf{J}$ can be computed analytically without explicit knowledge of the function f [16, pp.211-213]. In particular,

$$\mathbf{J} = \mathbf{sub}\left(-\left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial(\mathbf{x}, \lambda)}\right]^{-1}_{4\times 4} \left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial \mathbf{p}}\right]_{4\times 3} \right), \tag{6}$$

where $\mathbf{sub} : Hom(\mathbb{R}^3, \mathbb{R}^4) \to Hom(\mathbb{R}^3, \mathbb{R}^3)$ is an operator taking a 4×3 matrix to a 3×3 matrix by dropping the last row, and $Hom(V, W)$ represents the space of linear mappings from the vector space V to the vector space W [17, pp.45]. For clarity, we have indicated the matrix dimensions with subscripts in equation (6).

For an N-bounce path, a similar derivation can be performed on each ray segment formed by three consecutive points. By introducing an operator $\mathbf{aug} : Hom(\mathbb{R}^3, \mathbb{R}^4) \to Hom(\mathbb{R}^4, \mathbb{R}^4)$ to expand a 4×3 matrix with a zero column, the path Jacobians $\mathbf{J}_i$ in a multiple bounce path can be shown to satisfy a recurrence relation derived from the chain rule:

$$\mathbf{J}_i \;=\; \mathbf{sub}\left(-\left[A_i + \mathbf{aug}(B_i \cdot \mathbf{J}_{i+1})\right]^{-1} T_i \right), \tag{7}$$

where $\mathbf{J}_{\mathrm{N}+1} = 0$ and $\mathbf{J}_i (i = N, \ldots, 1)$ is computed sequentially from the fixed point $\mathbf{q} = \mathbf{x}_{\mathrm{N}+1}$ to the perturbed point $\mathbf{p} = \mathbf{x}_0$, and A_i, B_i, T_i are given by

$$\begin{aligned} A_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial(\mathbf{x}_i, \lambda_i)}\right]_{4\times 4} \\ B_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial \mathbf{x}_{i+1}}\right]_{4\times 3} \\ T_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial \mathbf{x}_{i-1}}\right]_{4\times 3}. \end{aligned} \tag{8}$$

Letting $\Gamma_i = [A_i + \mathbf{aug}(B_i \cdot \mathbf{J}_{i+1})]$ and $D_i = -\Gamma_i^{-1} T_i$, we may rewrite equation (7) as

$$T \;=\; -\Gamma \cdot D, \tag{9}$$

where we have omitted the subscript i for simplicity. Henceforth, we assume that all the variables with no subscripts are associated with a particular reflection point $\mathbf{x}_i$. For the second-order approximation, we need to compute the gradient of the matrix D, denoted by ∇D [18, pp.62]. The gradient ∇D is a third-order tensor and can be described with several different formalisms. It can be viewed as: the scalar components $(\nabla D)_{mij}$, the vector entries $\nabla(D_{ij})$, or the matrix layers $\nabla_m D$, defined respectively as

$$(\nabla D)_{mij} \equiv \frac{\partial D_{ij}}{\partial(\mathbf{x}_{i-1})_m},\; \nabla(D_{ij}) \equiv \frac{\partial D_{ij}}{\partial \mathbf{x}_{i-1}},\; \nabla_m D \equiv \left\{ \frac{\partial D_{ij}}{\partial(\mathbf{x}_{i-1})_m} \right\},$$

where $i = 1, 2, 3, 4$ and $m, j = 1, 2, 3$. By differentiating both sides of equation (9) with respect to $\mathbf{x}_{i-1}$ using Cartesian tensor notation, we obtain

$$\nabla_m D = -\Gamma^{-1} \left(\nabla_m T + \nabla_m \Gamma \cdot D \right) \tag{10}$$

for $m = 1, 2, 3$. The formulas for ∇T and $\nabla \Gamma$ are given by

$$\nabla(T_{jk}) = \frac{\partial T_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial T_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial T_{jk}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D \tag{11}$$

$$(\nabla \Gamma)_{mjl} = \begin{cases} \left(\nabla_m A + \nabla_m B \cdot \mathbf{J}_{i+1} + B \cdot \nabla_m C\right)_{jl} & (l = 1, 2, 3) \\ (\nabla A)_{mjl} & (l = 4) \end{cases} \tag{12}$$

where

$$\begin{aligned} \nabla\left(A_{jr}\right) &= \frac{\partial A_{jr}}{\partial \mathbf{x}_{i-1}} + \frac{\partial A_{jr}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial A_{jr}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D \\ \nabla\left(B_{jk}\right) &= \frac{\partial B_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial B_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial B_{jk}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D \\ \nabla\left(C_{kl}\right) &= \nabla\left((\mathbf{J}_{i+1})_{kl}\right) \cdot \mathbf{J}_i \end{aligned} \tag{13}$$

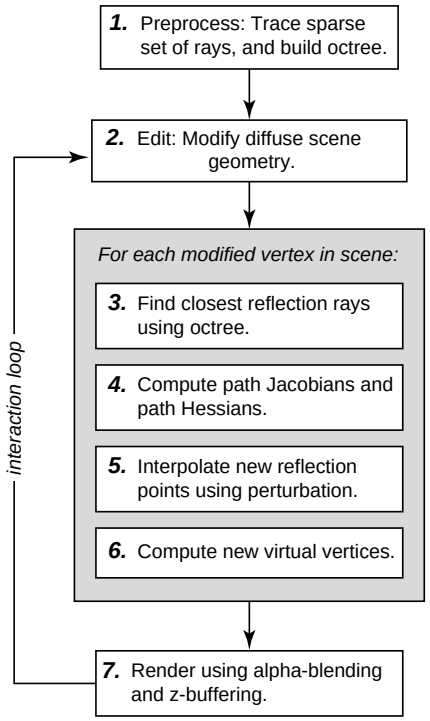


Fig. 2. *Overview of the algorithm for interactively computing specular reflections using local perturbations.*

for $j, r = 1, 2, 3, 4$ and $m, k, l = 1, 2, 3$. It follows from equation (3) and $\mathbf{J} = \mathbf{sub}(D)$ that the Hessian tensor can be formed from ∇D by[1]

$$\mathbf{H}_{jkm} = \frac{\partial \Psi_j}{\partial \mathbf{p}_k \mathbf{p}_m} = \frac{\partial \mathbf{J}_{jk}}{\partial \mathbf{p}_m} = (\nabla \mathbf{J})_{mjk} = (\nabla D)_{mjk}. \quad (14)$$

for $j, k, m = 1, 2, 3$. The dependence of the ith path Hessian on $\mathbf{J}_i$, $\mathbf{J}_{i+1}$ and $\nabla \mathbf{J}_{i+1}$ implied by equation (13) suggests that path Jacobians and path Hessians for an N-bounce path can be computed together, from the fixed point propagated to the perturbed point. Pseudo-codes describing all the steps of the computation is given in Section III-C.2.

III. A Perturbation Approach for Specular Reflections

In this section we present an algorithm for rapidly computing highly accurate reflections of dynamic objects in static curved specular reflecting surfaces. We define an interpolation scheme in the sense that we employ the analytic perturbation formula (4) shown in the previous section to construct a continuous reflection from discrete samples, which preserves the exact behavior at those samples. By combining ray tracing and hardware-supported alpha-blending and z-buffering, the entire process is fast enough for real-time interaction.

A. Overview

An overview of the process is shown in Fig. 2. It can be divided into three parts: *preprocessing*, *computing virtual objects*, and *hardware rendering*. The latter two parts form a loop during interactive rendering. Before the interaction loop begins, two steps are performed as preprocessing: a sparse set of rays is traced through the environment using standard ray tracing, and an octree hierarchy is constructed to partition the resulting reflection rays into small candidate sets. The heart of the algorithm, computing virtual objects, is shown within the shaded box of Fig. 2. Initially, for each tessellated vertex of a reflected object, these steps compute its virtual vertices which are then used to generate specular reflections of the scene. Updating each virtual vertex is accomplished in four steps, as shown in the figure: (1)*finding the closest reflection rays using the octree*, (2)*computing path Jacobians and path Hessians*, (3)*interpolating new reflection points using perturbation*, and (4)*computing new virtual vertices*. Finally, both the real objects and the virtual objects are rendered using standard graphics hardware that supports both alpha-blending and z-buffering. After tessellation, the specular reflectors are rendered as transparent surfaces, with transparency determined by their reflectivity. The following sections describe these steps in detail.

B. Preprocessing

B.1 Sparse Sampling

Our perturbation formula requires the existence of true reflection paths in order to interpolate new reflections. Due to the smooth variation of specular reflections, these samples may be relatively sparse over the image plane. In addition, if both the viewpoint and reflecting surfaces are fixed, these sampled paths can be reused for each update of the interactive computation.

The sampling over the image plane is performed with a ray tracer and done uniformly. We cast a sparse set of sample rays from the given vantage point and trace specular reflections in the scene using standard ray tracing techniques, skipping to every nth pixel in a scan line, and skipping to every nth scan line. In our tests, n varied from 8 to 32. We store all reflection paths with a depth less than or equal to the maximum level of specular reflections to be handled. All reflection points and associated reflecting surfaces along the path are retained. The reflection paths are distinguished into different types according to the number of bounces and the order in which surfaces are hit. For example, given a scene containing two reflectors, A and B, and a maximum reflection depth of two, the sampling results in four types of reflection rays: those that hit A, those that hit B, those that hit A then B, and those that hit B then A. These rays will serve as the samples from which all other reflections are interpolated. By perturbing different types of sample rays, we can approximate both single-bounce and multiple-bounce specular reflections. In

[1]For a multiple-bounce path, $\mathbf{p}$ in equation (14) is $\mathbf{x}_{i-1}$, and Ψ, $\mathbf{J}$, D and $\mathbf{H}$ are all associated to the ith reflection point.

the example of two reflectors, the four types of rays will be used to interpolate up to the second-level reflections.

For a smooth and implicitly-defined reflecting surface, we tessellate it into triangles, which are used both for z-buffered rendering and for accelerating ray-surface intersections; that is, ray-surface intersection is approximated by ray-triangle intersection. To improve the accuracy of the reflected rays, we use the surface normal at the intersection point instead of the triangle normal to determine the reflection ray. Surface normals for the curved surfaces can be obtained from the gradient of the function used to define them implicitly. To effectively handle the large number of triangles, a bounding slab hierarchy [19] is used to reduce the computation cost of tracing rays.

B.2 Constructing the Ray Octree

Because our approach interpolates by perturbing several nearest-neighbors, it is important to find a set of nearest reflection rays for any given point very quickly. Since there may be a large number of reflection rays, we wish to avoid a linear search. The approach we take is to construct an octree whose root node encloses the space bounding all dynamic objects of the environment. The subdivision of the octree is determined by the distribution of the precomputed reflection rays; a set of rays is associated with each node of the octree that is guaranteed to contain the k nearest rays for any point inside the cell. Once the octree is created, the closest rays to any point can be found by searching only those rays stored with the cell containing the point.

The key element for this space subdivision approach is to identify a small set of ray candidates for each cell of the octree. The procedure RayCandidateSet in Fig. 3 finds a subset of the rays that is guaranteed to contain the k closest rays to any point within a sphere S, which encloses a cell of the octree. The octree is then built recursively, building each candidate set from the candidate set of its parent node. The recursive subdivision terminates when further subdivision fails to reduce the candidate set or the candidate set has exactly k members.

Once the octree is constructed, we find the k closest rays to a given point $\mathbf{p}$ by first locating the octree cell containing $\mathbf{p}$, then searching its candidate list. Since points are very likely to be clustered closely together, locating the cell can be optimized by finding the first common ancestor with the previous point, and then descending to a leaf, which is a well-known optimization used in ray tracing [20].

Given a sphere S, a set of rays $\mathcal{R}$, and an integer k, the algorithm shown in Fig. 3 returns a set of rays $\mathcal{R}^*$ that will contain the k closest lines to any point in the sphere. When S is small compared to the original scene, $\mathcal{R}^*$ will generally be a small subset of $\mathcal{R}$.

```
RayCandidateSet( Sphere S, Rays R, integer k )

  Point A  ← center of sphere S
  Scalar r  ← radius of sphere S
  Rays R* ← empty set

  Compute distance interval for each ray, and keep
  all rays that intersect S.

  for each ray ϱ ∈ R do
    d ← distance from ray ϱ to the point A
    ϱ.min ← d − r
    ϱ.max ← d + r
    if ϱ.min ≤ 0 then add ϱ to R*
  endfor

  Ensure that the set R* contains at least k rays.

  if |R*| < k then
    Rays S ← elements of R − R* sorted by
                 increasing min distance
    add the first k − |R*| elements of S to R*
  endif

  Include all rays that are among the k closest to
  some point in S.

  Scalar α ← maximum of ϱ.max among all ϱ ∈ R*
  for each ϱ ∈ R − R* do
    if ϱ.min < α then add ϱ to R*
  endfor

  For any point p ∈ S the subset R* ⊂ R now
  contains the k rays of R that are closest to
  p (and possibly others).

  return R*
  end
```

Fig. 3. Pseudo-code to find the k closest rays to any point within a sphere enclosing an octree cell.

C. Computing Virtual Objects

After the preprocessing, the sparsely-sampled reflection rays are retained in a hierarchical structure. Each ray is tagged to record the position of reflectors along the reflection path. Then, when a non-specular object in the scene is manipulated, the algorithm computes new virtual objects for each reflected object that was modified.

The virtual object of a reflected object is constructed by connecting virtual vertices computed for each vertex of the tessellated object. In order to "close" virtual objects, we choose to compute virtual vertices for all the object vertices rather than determining which vertices are invisible from the current viewpoint. Each reflected object may correspond to several virtual objects, by virtue of multiple reflecting surfaces, multiple reflections within a single surface, or reflections at multiple levels. In the following sections, we describe how to compute a virtual vertex for an object vertex in detail.

C.1 Finding Closest Reflection Rays

Once the known reflection rays are stored in an octree, it is easy to search the tree to get the candidate ray sets $\mathcal{R}^*$ for any modified scene vertex $\mathbf{p}'$. For convex reflectors, a

point and its reflection has a one-to-one correspondence, we simply pick the closest ray in $\mathcal{R}^*$ and perturb it. However, a point may have more than one reflection point in a concave surface, Fig. 8 illustrates such an example. To generate multiple reflections in a non-convex surface, we require a mechanism that can detect rays associated with multiple reflections and divide the rays into groups. With a correct grouping, multiple reflections can be split by interpolating the rays in the respective groups.

Without prior information about reflective properties of a curved reflector, it is very difficult to split reflections correctly and robustly. In our implementation, we use several heuristics to decide whether the chosen k closest reflection rays ($k > 1$) should be split or not. Based on an assumption that two different reflections in a concave surface are generally perturbed from reflection rays which are well-separated or point in different directions, we choose the *reflection distance* (distance between corresponding reflection points of two rays) and the *ray direction* as two criteria for grouping. That is, the reflection rays in the same group are assumed to always have close reflection points and similar reflection directions. We compute the angle between two reflection directions to measure the direction similarity, and separate them into groups that remain within user-defined thresholds.

For a given point $\mathbf{p}'$, we first find the k closest reflection rays ($k = 5$ in our case) in ascending order from the ray candidate set and then do the grouping test. Initially, we assign the nearest of the chosen k rays as the representative of the first group. Then, for each of other $k - 1$ rays, we check the angles and the reflection distances between it and representatives of existing groups. If either the angle or the distance is over a pre-defined threshold for each representative, we assume this ray will contribute to another new reflection point for $\mathbf{p}'$ and assign it as the representative of a new group. Otherwise, it belongs to an existing group. Since our perturbation for a single reflection is based on the nearest neighbor, this new ray can be thrown out to keep only one closest ray for each group. Finally, depending on the geometry of the reflector, we will obtain either one ray (group) or several rays (groups) for perturbation; the latter case results in multiple reflections in a single surface.

C.2 Computing Path Jacobians and Path Hessians

For a given point $\mathbf{p}'$, once the closest reflection ray is found, we find the point $\mathbf{p}$ on the ray which is closest to $\mathbf{p}'$. Now the problem of computing the new reflection $\mathbf{x}'$ of $\mathbf{p}'$ is reduced to the third reflection problem shown in Fig. 1c, that is, computing how the position of the known reflection point $\mathbf{x}$ changes as the actual point $\mathbf{p}$ is perturbed by $\Delta\mathbf{p} = \mathbf{p}' - \mathbf{p}$. To apply the perturbation formula (4) to approximate the new reflection point, we compute the path Jacobians and/or path Hessians for the closest reflection path found.

In Section II, we showed the closed-form formulae for path Jacobians and path Hessians for a general multiple-bounce reflection path. Given a reflection path from $\mathbf{p}$ (varying) to $\mathbf{q}$ (fixed) via reflection points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, and the corresponding reflecting surfaces $g_1, g_2, \ldots, g_N$, the steps for computing the path Jacobian for each $\mathbf{x}_i$ can be summarized by the pseudo-codes in Fig. 4, where the subroutine SimpleJac is called to compute the path Jacobian for a single bounce path segment.

PathJacobians(**Point** $\mathbf{p}$, $\mathbf{q}$, $\mathbf{x}[\,]$, **Surface** $g[\,]$, **int** N)

1 **Matrix** $\mathbf{J}[\,]$
2 $\mathbf{x}_0 \leftarrow \mathbf{p}$

Compute the last path Jacobian
3 $\mathbf{J}_N \leftarrow$ SimpleJac($\mathbf{x}_{N-1}$, $\mathbf{x}_N$, $\mathbf{q}$, $\mathbf{0}$, g_N)

Compute the other path Jacobians from back to front
4 **for** $i = N-1, \ldots, 1$
5 $\quad\mathbf{J}_i \leftarrow$ SimpleJac($\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $\mathbf{J}_{i+1}$, g_i)
6 **endfor**
7 **return** $\mathbf{J}$

SimpleJac($\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $\mathbf{J}_{i+1}$, g_i)

1 **Matrix** $A, B, C, \Gamma, T, D_i, \mathbf{J}_i$

Compute lagrange multiplier λ_i for the ray segment $\mathbf{x}_{i-1} - \mathbf{x}_i - \mathbf{x}_{i+1}$
2 **Vector** $h \leftarrow$ gradient of g_i at $\mathbf{x}_i$
3 $h_j \leftarrow$ Nonzero component of h
4 **Scalar** $\lambda_i \leftarrow \frac{1}{h_j}\left(\frac{(\mathbf{x}_{i-1})_j - (\mathbf{x}_i)_j}{||\,\mathbf{x}_{i-1} - \mathbf{x}_i\,||} + \frac{(\mathbf{x}_{i+1})_j - (\mathbf{x}_i)_j}{||\,\mathbf{x}_{i+1} - \mathbf{x}_i\,||}\right)$

Compute the Jacobian matrices of F with respect to different variables using equation (8), evaluated at $(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)$.
5 $A \leftarrow [\partial F/\partial(\mathbf{x}_i, \lambda_i)]$
6 $T \leftarrow [\partial F/\partial \mathbf{x}_{i-1}]$
7 **if** $\mathbf{J}_{i+1} \neq \mathbf{0}$ **then**
8 $\quad B \leftarrow [\partial F/\partial \mathbf{x}_{i+1}]$
9 $\quad C \leftarrow B * \mathbf{J}_{i+1}$
10 $\quad C \leftarrow C$ *expanded with a zero column*
11 $\quad \Gamma \leftarrow A + C$
12 **else**
13 $\quad \Gamma \leftarrow A$
14 **endif**

Apply recursive formula (7)
15 $D_i \leftarrow \Gamma^{-1} T$
16 $\mathbf{J}_i \leftarrow$ the first three rows of D_i
17 **return** $\mathbf{J}_i$

Fig. 4. Pseudo-codes to evaluate path Jacobians for a specular reflection path from $\mathbf{p}$ to $\mathbf{q}$ via N reflection points stored in the array $\mathbf{x}[\,]$.

The Jacobian matrices A, B and T in lines 5, 6 and 8 of SimpleJac are computed by evaluating the first-order partial derivatives of the four explicit equations F_i shown in equation (5). Since F_i's are based on the gradient of the implicit function, the gradient vector and the Hessian matrix of the corresponding implicit function g are required to compute the path Jacobian. For any smooth function g,

its mixed partial derivatives are order-independent, which reduces the cost of computing the required partials.

As shown in Section II, the ith path Hessian is not only dependent on the ith path Jacobian, but also dependent on the $(i+1)th$ path Jacobian and path Hessian. Therefore, if the second-order approximation is required, we start from the last bounce, and for each reflection point compute the path Jacobian followed by the path Hessian. The following pseudo-code summarizes the steps to compute the second-order approximation (including both path Jacobians and path Hessians) of an N-bounce path.

PathJacHess(**Point** $\mathbf{p}$, $\mathbf{q}$, $\mathbf{x}[\,]$, **Surface** $g[\,]$, **int** N)

1 **Matrix** $\mathbf{J}[\,]$
2 **Tensor** $\mathbf{H}[\,]$
3 $\mathbf{x}_0 \leftarrow \mathbf{p}$

Compute the last path Jacobian and Hessian.
4 $\mathbf{J}_{\mathrm{N}} \leftarrow$ SimpleJac($\mathbf{x}_{\mathrm{N\text{-}1}}$, $\mathbf{x}_{\mathrm{N}}$, $\mathbf{q}$, $\mathbf{0}$, g_{N})
5 $\mathbf{H}_{\mathrm{N}} \leftarrow$ SimpleHess($\mathbf{x}_{\mathrm{N\text{-}1}}$, $\mathbf{x}_{\mathrm{N}}$, $\mathbf{q}$, g_{N}, $\mathbf{J}_{\mathrm{N}}$, $\mathbf{0}$, $\mathbf{0}$)
Compute the other path Jacobians and path Hessians from back to front
6 **for** $i = N-1, \ldots, 1$
7 $\quad\mathbf{J}_i \leftarrow$ SimpleJac($\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $\mathbf{J}_{i+1}$, g_i)
8 $\quad\mathbf{H}_i \leftarrow$ SimpleHess($\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, g_i, $\mathbf{J}_i$, $\mathbf{J}_{i+1}$, $\mathbf{H}_{i+1}$)
9 **endfor**
10 **return** $\mathbf{J}$, $\mathbf{H}$

To compute the path Hessian for each intermediate bounce, PathJacHess calls a subroutine SimpleHess recursively, which is provided in Fig. 5. To evaluate the partial derivatives in lines 7, 8 and 9 of SimpleHess, the second-order partial derivatives of the four explicit equations $F_1, \cdots, F_4$ in equation (5) are required, which in turn requires the third-order derivatives of the implicit function g.

Due to the computational cost associated with path Hessians, we use the second-order perturbation only when the local curvature of the curved surface is estimated to be large. For nearly flat areas, the linear approximation with path Jacobians is sufficient. The local curvature of an implicit surface can be approximated in several ways; for example, by computing the second-order derivatives of the implicit function, or by using finite differences.

SimpleHess($\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, g_i, $\mathbf{J}_i$, $\mathbf{J}_{i+1}$, $\mathbf{H}_{i+1}$)

1 **Matrix** $A, B, T, D_i \leftarrow$ obtained from SimpleJac
2 **Tensor** ∇T, ∇A, ∇B, ∇C, ∇E, $\nabla\Gamma$, ∇D_i, $\nabla\mathbf{J}_{i+1}$
3 **Tensor** $\mathbf{H}_i$
4 $\nabla\mathbf{J}_{i+1} \leftarrow \mathbf{H}_{i+1}$ reorganized according to equation (14)
Compute the gradient ∇T, ∇A, ∇B, ∇C according to equations (11) and (13).
5 **for** $j, r = 1, 2, 3, 4$
6 $\quad$**for** $k, l = 1, 2, 3$
7 $\quad\quad\nabla(T_{jk}) \leftarrow \frac{\partial T_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial T_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial T_{jk}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D_i$
8 $\quad\quad\nabla(A_{jr}) \leftarrow \frac{\partial A_{jr}}{\partial \mathbf{x}_{i-1}} + \frac{\partial A_{jr}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial A_{jr}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D_i$
9 $\quad\quad\nabla(B_{jk}) \leftarrow \frac{\partial B_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial B_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial B_{jk}}{\partial(\mathbf{x}_i, \lambda_i)} \cdot D_i$
10 $\quad\quad\nabla(C_{kl}) \leftarrow \nabla((\mathbf{J}_{i+1})_{kl}) \cdot \mathbf{J}_i$
11 $\quad$**endfor**
12 **endfor**
Compute the gradient $\nabla\Gamma$ according to equation (12).
13 **for** $m = 1, 2, 3$
14 $\quad\nabla_m E \leftarrow \nabla_m A + \nabla_m B \cdot \mathbf{J}_{i+1} + B \cdot \nabla_m C$
15 **endfor**
16 **for** $m = 1, 2, 3$
17 $\quad$**for** $j = 1, 2, 3, 4$
18 $\quad\quad$**for** $l = 1, 2, 3$
19 $\quad\quad\quad(\nabla\Gamma)_{mjl} \leftarrow (\nabla_m E)_{jl}$
20 $\quad\quad$**endfor**
21 $\quad\quad(\nabla\Gamma)_{mj4} \leftarrow (\nabla A)_{mj4}$
22 $\quad$**endfor**
23 **endfor**
Compute the gradient ∇D_i according to equation (10).
24 **for** $m = 1, 2, 3$
25 $\quad\nabla_m D_i \leftarrow -\Gamma^{-1}(\nabla_m T + \nabla_m\Gamma \cdot D_i)$
25 **endfor**
Compute path Hessian from ∇D_i using equation (14).
26 **for** $j, k, m = 1, 2, 3$
27 $\quad(\mathbf{H}_i)_{jkm} \leftarrow (\nabla D_i)_{mjk}$
28 **endfor**
29 **return** $\mathbf{H}_i$

Fig. 5. Pseudo-code to evaluate the path Hessian tensor for an intermediate bounce of an N-bounce specular reflection path.

C.3 Interpolating Reflection Points

Let $\mathbf{p}'$ be a point near $\mathbf{p}$. Suppose $(\mathbf{p}, \mathbf{x}_1, \ldots, \mathbf{x}_{\mathrm{N}}, \mathbf{q})$ is a known reflection path found to be closest to $\mathbf{p}'$. When viewed from the fixed vantage point $\mathbf{q}$, the Nth level reflection of $\mathbf{p}$ appears at the position $\mathbf{x}_{\mathrm{N}}$ in the corresponding curved surface. When the perturbation from $\mathbf{p}$ to $\mathbf{p}'$ is small, the Nth reflection of $\mathbf{p}'$ on the same surface can be obtained by perturbation. With path Jacobians and/or path Hessians evaluated for this closest known path, we can choose to approximate the Nth level reflection of $\mathbf{p}'$ to different accuracy. That is, the new reflection point $\mathbf{x}'_{\mathrm{N}}$ can be approximated to first-order accuracy by

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} \tag{15}$$

or to second-order accuracy by

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} + \frac{1}{2}(\Delta\mathbf{x}_{i-1})^{\mathrm{T}} \mathbf{H}_i \Delta\mathbf{x}_{i-1} \tag{16}$$

where $i = 1, 2, 3, \ldots, N$, $\Delta\mathbf{x}_{i-1} = \mathbf{x}'_{i-1} - \mathbf{x}_{i-1}$ and $\Delta\mathbf{x}_0 = \mathbf{p}' - \mathbf{p}$. Shown in Fig. 6 are several known single-level reflection paths associated with the given view point and reflector. All other single-level reflections are approximated

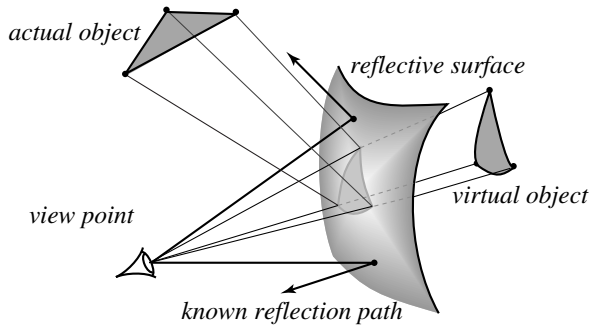


Fig. 6. *The reflection of an actual object is computed by perturbing one or more known reflection paths. All virtual objects are then z-buffered with the rest of the environment. The reflecting surface is alpha-blended, with transparency corresponding to reflectivity of the actual surface.*

by perturbing these, using the nearest reflection ray in each case. The resulting virtual vertices define a virtual object corresponding to the reflection, as described in the following section.

C.4 Computing the Virtual Vertex

Once the reflection point $\mathbf{x}'$ of the point $\mathbf{p}'$ is approximated, it is used to compute the corresponding virtual point $\mathbf{v}'$. This virtual point is placed at a distance from the eye that is equal to the optical path length from the eye to the actual point. This places the point $\mathbf{v}'$ behind the reflecting surface. By preserving the optical path length in the virtual objects, we preserve their ordering with respect to z-buffering. That is, z-buffering will have the correct effect on the resulting virtual objects and actual objects.

It is important that all surfaces of reflected objects be finely tessellated, as the mapping from actual to virtual object is non-linear, and will warp all surfaces. It is also necessary to assign pre-computed vertex colors to the virtual object that match the actual object, rather than allowing the 3D graphics hardware to shade them. This is because the orientation and distance of the virtual objects with respect to the light sources will in general produce shading that is inconsistent with the corresponding actual object.

Based on the octree hierarchy created in the preprocessing pass, the algorithm that combines the above four steps to compute a virtual vertex for a single point in 3D is summarized in Fig. 7. Here, the virtual vertex computed is for a single-level specular reflection, the point $\mathbf{q}$ passed in is the view position where the sample rays originate and the second-order perturbation is used.

ComputeVirtualVertex(**Point** $\mathbf{q}$, $\mathbf{p}'$, **Surface** g)

Cube $C \leftarrow$ cell containing $\mathbf{p}'$
Rays $\mathcal{R} \leftarrow$ candidate set of C

Find the nearest ray to p' using a linear search.
Ray $\varrho \leftarrow$ the ray of C closest to $\mathbf{p}'$

Perturb the path according to equation (16).
Point $\mathbf{x} \leftarrow$ origin of the ray ϱ
Point $\mathbf{p} \leftarrow$ the point on ϱ closest to p'
Vector $\Delta\mathbf{p} \leftarrow \mathbf{p}' - \mathbf{p}$
Matrix $\mathbf{J}$,**Tensor** $\mathbf{H} \leftarrow$ PathJacHess($\mathbf{p}$, $\mathbf{q}$, $\mathbf{x}$, g, 1)
Point $\mathbf{x}' \leftarrow \mathbf{x} + \mathbf{J}\Delta\mathbf{p} + \frac{1}{2}\Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{p}$

Create the virtual vertex .
return $\mathbf{x}' + \dfrac{||\,\mathbf{x}' - \mathbf{p}'\,||}{||\,\mathbf{x}' - \mathbf{q}\,||}(\mathbf{x}' - \mathbf{q})$

Fig. 7. Pseudo-code to compute the virtual vertex for a single-level specular reflection of a perturbed point $\mathbf{p}'$, viewed from $\mathbf{q}$.

D. Hardware Rendering

After all the virtual objects are constructed, the entire scene, consisting of real and virtual objects, is rendered using standard graphics hardware that supports both z-buffering and alpha-blending. The use of z-buffering and alpha-blending in this context is made possible by several important facts. First, since virtual objects are positioned behind the reflective surface with respect to the view point, reflectivity of a surface can be simulated using alpha-blended transparency to "merge" specular reflections onto real objects. Second, since the optical length is preserved in computing the virtual vertices, the correct depth information and ordering of real and virtual objects is maintained. Therefore, hidden surface removal and relative visibility are correctly handled by z-buffering, even for reflections.

The idea of using a hardware graphics pipeline to simulate more advanced effects has a long history. For example, Williams simulated shadows of curved objects using multiple z-buffer passes [21], while Haeberli and Akeley proposed the use of multiple images and an *accumulation buffer* to perform hardware anti-aliasing [22]. In a similar spirit, Diefenbach and Badler [23] employed accumulation buffers and projective texture mapping to simulate specular reflection and transmission. However, our approach fits naturally into a *single* pass though a standard graphics pipeline, with the only caveat being that the shading of virtual objects must be handled in software.

IV. Results

We have implemented our perturbation algorithm using Open Inventor on a SGI Indigo2. The reflecting surface we chose is a vase model defined by the implicit function $g(x, y, z) = 0$, where

$$g(x,y,z) = \sqrt{x^2+y^2} - \left[a\left(\frac{z}{h}\right)^3 + b\left(\frac{z}{h}\right)^2 + c\left(\frac{z}{h}\right) + d\right].$$

Here a, b, c, d are polynomial coefficients for the contour of the vase, and h is the height of the vase. Specifically, $a = 5.80$, $b = -9.78$, $c = 3.90$, $d = 0.47$, $h = 2$. This surface is a good test of our method because it has mixed convexity. At some locations, an object may have two reflection images on the vase, one near the top and the other near the bottom. The vase is equipped with a bounding slab hierarchy to speed up the ray-surface intersection.

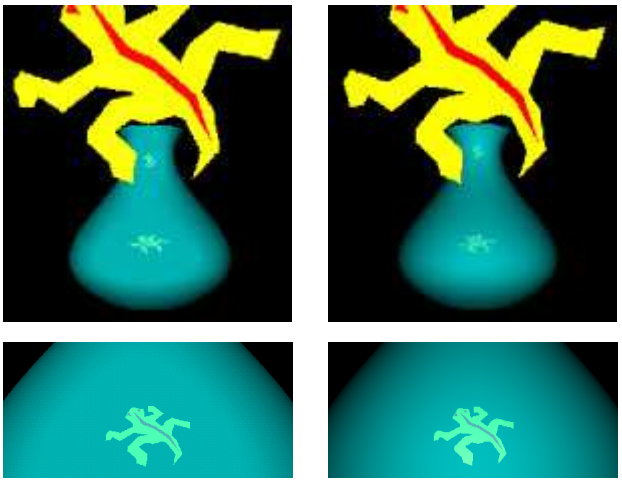


Fig. 8. *Side-by-side comparison of one-bounce reflection images generated by the perturbation method (left) and ray tracing (right). The results are nearly identical, yet the images in the left column can be computed very rapidly (approximately 0.1 seconds per update) as the lizard-shaped polygon is moved interactively.*

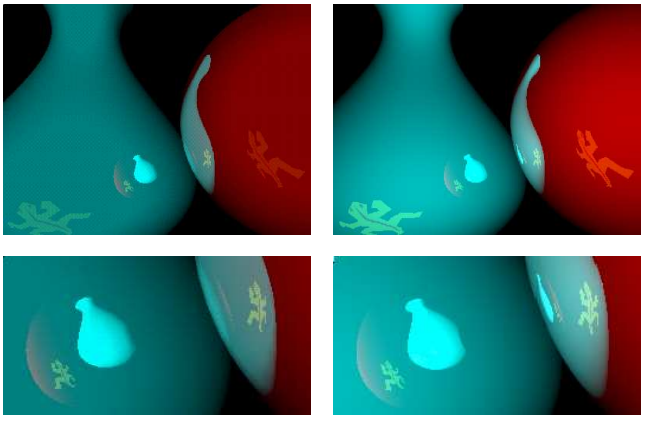


Fig. 9. *Side-by-side comparison of multiple-bounce reflection images generated by the perturbation method (left) and ray tracing (right). The images on the left show all reflections up to two bounces, while the ray traced images on the right show all reflections up to a depth of five.*

For a scene with a reflecting vase and a diffuse polygonal lizard, Fig. 8 shows a side-by-side comparison of the reflection images of the lizard generated by our perturbation method and ray tracing. The top image is the full view of the scene, and the bottom shows a closeup of the reflection near the bottom. The images in the left column are generated by perturbation and the ray traced images on the right are rendered by PovRay, a widely available ray tracer. The image resolution is 640×480 and the lizard is triangulated by connecting 61 vertices. After casting 40×30 sample rays in the preprocessing, we computed the bottom reflection with linear interpolation using the single-bounce path Jacobian, and approximated the top reflection to the second-order accuracy using the path Hessian to handle the greater curvature near the neck of the vase. The resulting image is nearly indistinguishable from the ray traced version.

TABLE I

Performance comparison between perturbation and ray tracing.

	pre-processing		updates	
	Fig. 8	Fig. 9	Fig. 8	Fig. 9
Perturbation	0.43	9.6	0.1	0.7
Ray Tracing	41	67	41	67

As an example of generating multiple-bounce specular reflections by the perturbation method, another reflector has been added to the scene. Here, the second-level specular reflections are computed by perturbing two-bounce paths to first-order accuracy using the recursive formula derived for multiple-bounce path Jacobians. The side-by-side comparison with a ray traced image is shown in Fig. 9. Similarly, the bottom images show a closeup view of the two-bounce reflections. 80×60 sample rays were casted for this 640×480 image. The reflections (virtual objects) of the vase and the sphere are constructed by tessellating them into 1250 triangles. The ray traced images are rendered with a maximum depth of 5, which accounts for the third-level reflections in the right column. The reflections up to the second-level are nearly identical for both methods. The slight difference in shading in the two images is due to slightly different shading algorithms used in hardware and software.

We have compared our perturbation algorithm and PovRay on a SGI Indigo2 working at 175 Mhz $R10000$, with Impact graphics board. Excluding the preprocessing time from our perturbation method and the time for parsing and creating bounding slab tree from ray tracer, their performances on these two scenes are summarized in TABLE I. The second column shows the time for rendering the initial whole scene, and the third column shows the time for updating the scene while moving the lizard. Fig. 8 and Fig. 9 demonstrate that perturbation provides a great speedup over conventional ray tracing with little sacrifice in accuracy. Most importantly, by exploiting path coherence from frame to frame, which is also a motivation of path perturbation, we can rapidly update specular reflections while moving diffuse objects interactively. In the case of the scenes shown in Fig. 8 and Fig. 9, each update takes less than one second.

In Fig. 10, we render a scene with a reflecting vase, a window, a floor, and a moving icosahedron. The reflections of the window, the floor and the icosahedron are all interpolated to second-order. Hidden surface removal is performed by z-buffering the entire scene, including the reflections simulated by virtual objects. Fig. 10 shows several frames in an animation sequence, which are generated interactively at 30 frames/sec.

V. Discussion

We have described a practical approach for rapidly computing specular reflections in arbitrary implicitly-defined curved surfaces. The technique uses second-order perturbations of known specular reflection paths, which are ray

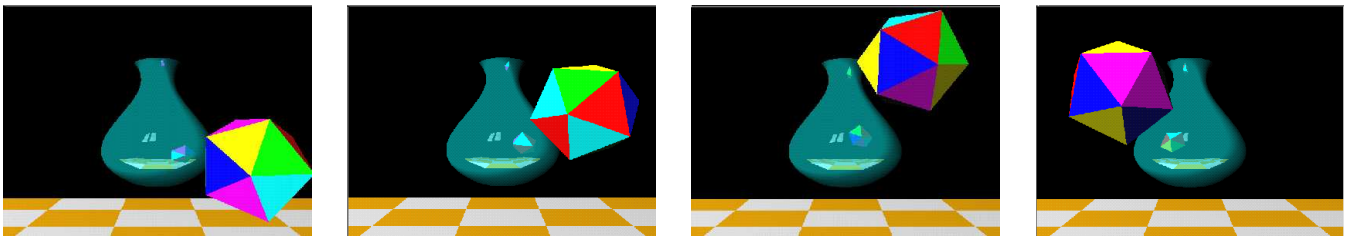
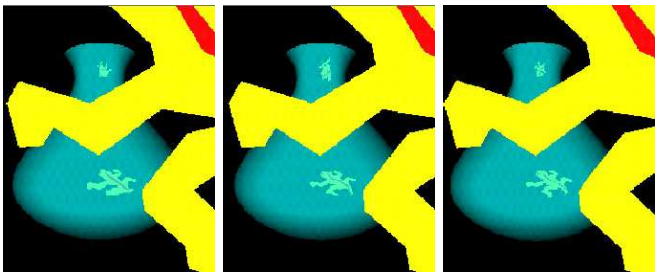


Fig. 10. *Images generated interactively by repositioning, rotating, and scaling the icosahedron. The reflection of the icosahedron is updated in real time as it is moved. Hidden surface removal is performed in one pass of z-buffering for the entire scene, including the reflections.*

traced in a pre-processing step, to compute virtual objects for each reflection. The virtual objects and the real objects are then rendered using standard graphics hardware supporting both alpha-blending and z-buffering. The entire process is fast enough for real-time interaction, even accounting for multiple-bounce specular reflections. The simulated reflections are of very high accuracy, as indicated by our test results, which are nearly identical to ray traced images.

Next we discuss the limitations of our perturbation approach, several tradeoffs between its reflection quality and performance, and contrast it with other related work.



$8 \times 8/413ms$ $10 \times 10/416ms$ $16 \times 16/418ms$

Fig. 11. *Varying sample density affects reflection quality and rendering time.*

Limitations: The implicit function defining the reflective surface is encapsulated in the nonlinear system F and thus the closed-form expression (6) for path Jacobians. Therefore, it is required that the reflective surface has a differentiable implicit definition, which also guarantees a continuous specular reflections for our interpolation. A practical extension would be to generalize the formulas for path Jacobians and path Hessians to more popular surface representations, such as parametric surfaces.

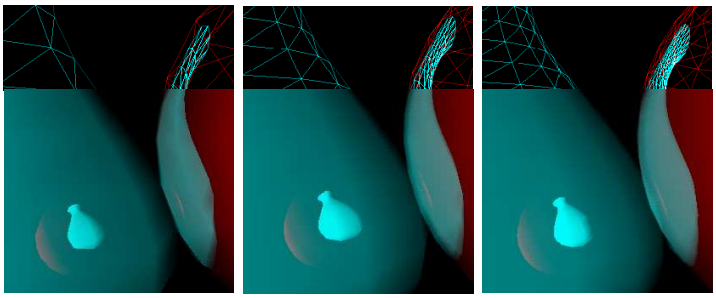
While computing virtual objects in Section III-C, we assumed that the reflected object and the viewpoint always lie on the same side of the reflector. As for *mixed polygons* which are partially located on the other side of the reflector with respect to the viewpoint, our current implementation cannot find appropriate sample rays for some hidden vertices and will thus fail. Although Ofek and Rappoport [8] proposed a second z-buffer (relating to a reflector) for such a problem, this approach is not well supported by current graphics architecture. Another way to handle this problem is to compute the virtual object only for the part of the object that lies in front of the reflector.

Our perturbation approach has another limitation with regard to concave reflectors, which can produce complicated reflections. Ofek and Rappoport [8] classified the space in front of a concave reflector into three reflection regions. An object falling in regions A, B, or C generates a single deformed virtual object, a single upside-down virtual object, or multiple virtual objects, respectively. Thus, when a reflected object crosses regions A and C (or B and C), different object vertices may have different numbers of virtual vertices. It is hard for our virtual object algorithm to find the correspondence between these virtual vertices to correctly "close" the virtual objects, and the resulting reflection image becomes "chaotic". A possible way to handle this case is to detect such a crossing and further decompose the reflected object until it falls entirely within a region of type A or B.

Sample Density: Since the second-order perturbation has an error of order $O(||\,\Delta\mathbf{p}\,||^3)$, the accuracy of our perturbation method depends upon the underlying sample rays. Denser sampling leads to smaller $||\,\Delta\mathbf{p}\,||$, and thus a higher accuracy. Fig. 11 shows how varying sample density affects reflection quality and rendering performance. The image resolution is 512×512. From left to right, the reflections become finer as the sample rays become denser. On the other hand, more sample rays increases the cost of locating nearest neighbors, which increases rendering time. Due to continuous specular reflections, the sampling over the image plane can be relatively sparse. For the scene in Fig. 11, an array of 16×16 sample rays produces nearly identical results compared to those from 32×32 sampling.

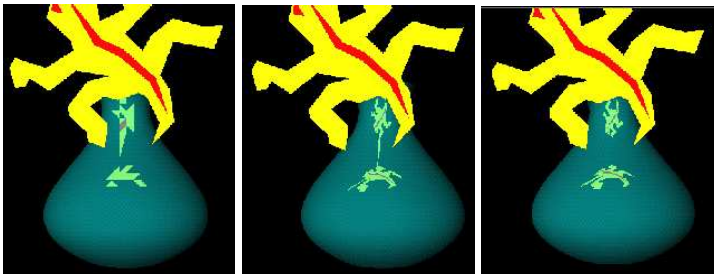
Tessellation: Tessellation presents another important tradeoff to consider when using the perturbation method. Since we create virtual objects by connecting virtual vertices computed for each tessellated vertex of the reflected objects, the complexity of this algorithm is linear in the tessellation size of the reflected objects. However, the smoothness of the simulated specular reflections is improved accordingly. Fig. 12 compares specular reflections as well as rendering times corresponding to different levels of tessellation. Each reflection image is generated by tessellating the vase and the sphere with the number of triangles shown beneath each image.

Approximation Order: Fig. 13 shows three images gen-



200/1.3s 450/2.6s 800/4.5s

Fig. 12. *Varying tessellation levels cause different qualities and rendering times.*



0.36s 0.39s 0.53s

Fig. 13. *Three images rendered using the sparse set of reflection rays. Left: nearest neighbor interpolation. Middle: Linear interpolation using the path Jacobian. Right: Quadratic interpolation using the path Hessian.*

erated by using three interpolation strategies of different orders on a sparse set of sampled reflection rays. Given 40×30 sample rays, the left image uses the nearest neighbor reflection without any perturbation, the middle image applies linear interpolation with the path Jacobian computed for the nearest neighbor, and the right image is generated from quadratic interpolation of the nearest neighbor using the path Hessian. The reflection quality clearly improves from left to right.

Without local perturbation, the sparse sample rays are not dense enough to reasonably approximate the reflection. The accuracy of linear interpolation and quadratic interpolation is dependent on the local curvature of the reflecting surface. Linear approximation works well near the relatively flat bottom of the vase, while quadratic interpolation becomes necessary to attain the same level of accuracy in highly curved regions.

Another alternative is to pre-cache path Jacobians and path Hessians in the preprocessing pass. This can be done by deriving path Jacobians and/or path Hessians in terms of a reflection direction rather than a particular position along this direction. The motivation behind this is that path Jacobians and/or path Hessians for the family of reflection paths represented by the same reflection direction are closely related. Characterizing this relation analytically is future work.

Comparison: Other work related to specular reflections on curved surfaces include environment mapping and the work of Ofek and Rappoport [8]. As mentioned in Section I, environment mapping fails for dynamic objects relatively close to the reflector. Although our method and the approach of Ofek and Rappoport are both based on the idea of virtual objects, they are different in several important respects.

- Curved reflectors are represented differently for these two approaches. Ofek and Rappoport handle curved reflectors with a polygonal mesh, while we deal with any curved surface associated with a differentiable implicit definition. The tessellation of the reflecting surfaces in our approach is convenient for both ray tracing and z-buffering, but plays no role in the computation of reflection points[2]. Hence, other ray intersection and rendering procedures (such as those that work directly with implicit surfaces) could be employed without changing the essence of our approach.
- The explosion map used by Ofek and Rappoport to accelerate the computation of virtual objects is similar to an environment map and thus suffers from the same disadvantage of spherical environment mapping; that is, its accuracy is dependent on the reflector shape. In addition, by approximating the correct reflection ray by a ray from the center of the reflector, none of the triangles in the map correspond to the actual reflection cell, which distorts the specular reflections for more complicated surfaces. Since the surface curvature information is already embedded in the formulae for path Jacobians and path Hessians, our second-order perturbation can generate accurate reflections for general curved surfaces. The approximation error for the perturbation method comes from truncation, visibility and boundary exceptions. Of course, error analysis is required for precise accuracy comparison between these two methods.
- The method proposed by Ofek and Rappoport applies only to single-level specular reflections in curved surfaces. In contrast, the perturbation method accommodates multiple-bounce specular reflections as easily as single-bounce reflections by using the closed-form recursive formulae for path Jacobians and path Hessians.
- Ofek and Rappoport decomposed reflectors of mixed curvexity into pure convex parts and pure concave parts. By perturbing a specular path analytically, accounting for the geometric properties of the reflector (by differentiating its implicit function to higher orders), our method can handle mixed surfaces with no special cases.

In summary, both methods have their advantages and limitations. In general, the approach of Ofek and Rappoport is more efficient for some special types of reflectors, such as planar surfaces, linear extrusions, and spherical-like surfaces, while our method is more accurate for complicated curved surfaces. A very practical extension would be to combine these two methods, as well as environment mapping to handle very complex scenes. Reflections of distant static objects could be approximated using a conventional environment map, while reflections of nearby dynamic objects could be computed more accurately by path pertur-

[2]When the reflector becomes a reflected object, its reflections are computed from the tessellated points.

bation or the explosion map, depending on the reflective geometry and accuracy requirements.

VI. Future Work

Many extensions of the theory of specular path perturbation are possible. For example, one could extend the idea very naturally to a refraction path or a path mixing reflection and refraction. With the index of refraction taken into account, derivations for a refraction path follow a similar strategy, and the same algorithm can be applied to real-time simulation of lens effects. To precisely evaluate the accuracy of path perturbation, error analysis is required to identify and quantify the error in the perturbation method, which results from truncation of the Taylor series, boundary exceptions, and changes in visibility. Another interesting topic is the extension of specular path perturbation to more general parametric surfaces without an implicit definition.

There are also many optimizations to make the approach described in Section III faster and more robust, such as adaptively sampling rays in the object space, based on the contour of the reflector; balancing quality and performance by choosing proper tessellation and interpolation strategies; exploring more efficient ways to store and search the sample ray space; and caching path Jacobians and/or path Hessians.

There are potentially many other applications for perturbation methods of this nature to image synthesis. For example, path derivatives may prove useful for image warping where specular effects are prominent [24], for stereoscopic rendering in the context of emersive displays [25], for interpolating samples to speedup Monte Carlo ray tracing [26], and for improved mutation strategies in metropolis light transport [2].

Acknowledgments

The authors wish to thank Anil Hirani and Al Barr for many valuable discussions, Don Mitchell and Pat Hanrahan for their patience in answering our questions, and Mark Meyer for helpful comments. This work was supported in part by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization, the Army Research Office Young Investigator Program (DAAH04-96-100077), and the Alfred P. Sloan Foundation.

References

[1] Don Mitchell and Pat Hanrahan, "Illumination from curved reflectors," *Computer Graphics*, vol. 26, no. 2, pp. 283–291, July 1992.

[2] Eric Veach and Leonidas J. Guibas, "Metropolis light transport," in *Computer Graphics* Proceedings, Aug. 1997, Annual Conference Series, ACM SIGGRAPH, pp. 65–76.

[3] James Arvo and David Kirk, "A survey of ray tracing acceleration techniques," in *An Introduction to Ray Tracing*, Andrew S. Glassner, Ed., chapter 6. Academic Press, New York, 1989.

[4] James F. Blinn and M. E. Newell, "Texture and reflection in computer generated images," *Communications of the ACM*, vol. 19, no. 10, pp. 542–547, Oct. 1976.

[5] Brian Cabral, Marc Olano, and Philip Nemec, "Reflection space image based rendering," in *Computer Graphics* Proceedings, Aug. 1999, Annual Conference Series, ACM SIGGRAPH, pp. 165–169.

[6] E. S. Panduranga, *Reflections in Curved Surfaces*, Ph.D. thesis, Princeton University, Oct. 1987, Technical Report CS-TR-122-87.

[7] Holly E. Rushmeier and Kenneth E. Torrance, "Extending the radiosity method to include specularly reflecting and translucent materials," *ACM Transactions on Graphics*, vol. 9, no. 1, pp. 1–27, Jan. 1990.

[8] Eyal Ofek and Ari Rappoport, "Interactive reflections on curved objects," in *Computer Graphics* Proceedings, July 1998, Annual Conference Series, ACM SIGGRAPH, pp. 333–342.

[9] Gregory J. Ward and Paul S. Heckbert, "Irradiance gradients," in *Proceedings of the Third Eurographics Workshop on Rendering,* Bristol, United Kingdom, May 1992, pp. 85–98.

[10] Homan Igehy, "Tracing ray differentials," in *Computer Graphics* Proceedings, Aug. 1999, Annual Conference Series, ACM SIGGRAPH, pp. 179–186.

[11] C. C. Lin and L. A. Segel, *Mathematics Applied to Deterministic Problems in the Natural Sciences*, Society for Industrial and Applied Mathematics, Philadelphia, 1988.

[12] Min Chen, "Perturbation methods for image synthesis," M.S. thesis, California Institute of Technology, May 1999, Technical Report CS-TR-99-05, ftp://ftp.cs.caltech.edu/tr/cs-tr-99-05.ps.Z.

[13] Min Chen and James Arvo, "Theory and application of specular path perturbation," May 1999, Submitted for publication.

[14] Max Born and Emil Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, Pergamon Press, New York, third edition, 1965.

[15] David G. Luenberger, *Optimization by Vector Space Methods*, John Wiley & Sons, New York, 1969.

[16] Jerrold E. Marsden and Michael J. Hoffman, *Elementary Classical Analysis*, W. H. Freeman, New York, 1993.

[17] Lynn H. Loomis and Shlomo Sternberg, *Advanced Calculus*, Addison-Wesley, Reading, Massachusetts, 1968.

[18] L. A. Segel, *Mathematics Applied to Continuum Mechanics*, Dover Publications, Inc., New York, 1987.

[19] Timothy L. Kay and James Kajiya, "Ray tracing complex scenes," *Computer Graphics*, vol. 20, no. 4, pp. 269–278, Aug. 1986.

[20] Hanan Samet, "Implementing ray tracing with octrees and neighbor finding," *Computers and Graphics*, vol. 13, no. 4, pp. 445–460, 1989.

[21] Lance Williams, "Casting curved shadows on curved surfaces," *Computer Graphics*, vol. 12, no. 3, pp. 270–274, Aug. 1978.

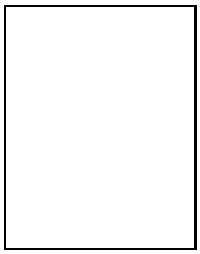
[22] Paul Haeberli and Kurt Akeley, "The accumulation buffer: Hardware support for high-quality rendering," in *Computer Graphics* Proceedings, Aug. 1990, Annual Conference Series, ACM SIGGRAPH, pp. 309–318.

[23] Paul J. Diefenbach and N. Badler, "Pipeline rendering: Interactive refractions, reflections and shadows," *Displays: Special Issue on Interactive Computer Graphics*, vol. 15, no. 3, pp. 173–180, 1994.

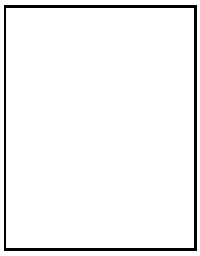
[24] Dani Lischinski and Ari Rappoport, "Image-based rendering for non-diffuse synthetic scenes," in *Proceedings of the Ninth Eurographics Workshop on Rendering*, Vienna, Austria, June 1998.

[25] Wolfgang Krüger, Christina-A. Bohn, Bernd Fröhlich, Heinrich Schüth, Wolfgang Strauss, and Gerold Wesche, "The responsive workbench: A virtual work environment," *IEEE Computer*, vol. 28, no. 7, pp. 42–48, July 1995.

[26] James T. Kajiya, "The rendering equation," *Computer Graphics*, vol. 20, no. 4, pp. 143–150, Aug. 1986.



Min Chen received her B.S. and M.S. in Computer Science from Peking University, China, in 1994 and 1997, respectively. She received another M.S. in Computer Science in 1999 from California Institute of Technology. She is currently a PhD candidate with the Department of Computer Science at the California Institute of Technology. Her research interests include physically-based rendering, image-based rendering, human computer interaction, animation and computer graphics.



James Arvo is an Associate Professor of Computer Science at the California Institute of Technology. He received a B.S. in Mathematics from Michigan Technological University, an M.S. in Mathematics from Michigan State University, and a Ph.D. in Computer Science from Yale University in 1995. His research interests include physically-based image synthesis, human-computer interaction, and artificial intelligence. Dr. Arvo received a young investigator award from the U.S. Army Research Office in 1996, and an Alfred P. Sloan Research Fellowship in 1997 for his work in image synthesis.