PERTURBATION METHODS FOR IMAGE SYNTHESIS

Thesis by
Min Chen

Advisor
James R. Arvo

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

California Institute of Technology
Pasadena, California

1999

(Defended May, 1999)

Acknowledgements

There are many people I wish to thank. First and foremost, I offer my sincere thanks to my advisor, James Arvo for his steadfast confidence in me and getting me admitted into Caltech. Without his unhesitating support and encouragement, this thesis work would never have been completed. He has been a wonderful influence and a guide in my research life. From working with him, I tasted the joy of serious research work and developed strong interest in solving challenging graphics problems. I am grateful to him for his kindness and sharing his brilliant insights with me.

I am also deeply indebted to my colleague, Anil Hirani. He helped me to re-capture my self-confidence and have the courage to face the mathematical challenge. Also, I should thank him for his posing the Implicit function theorem into my attention and kindly helping me to find my way to completion.

I also wish to thank my other committee members, Peter Shröder, whose consistent confidence and support are always a great cure for my depression. Many thanks to him for a lot of new and related reference he provided, which not only broadened my view, but also a great help to my work. Besides, he is also a pleasure to exchange personal feelings. Al Barr, who generously offered his time and expertise to solve my problem and gave me very constructive suggestions.

Special thanks to my officemate Mark Meyer for stimulating discussions and offering detailed comments. Thank Dave Felt for his help with my videotape. Thanks also to my women friends at Caltech, Zoë Wood and Eve Schooler, who taught me to be robust and brave in an intense research environment.

The person to whom I am most indebted is my fiancé, Hui, whose love and understanding are truly without bound. It is difficult to imagine how I could have completed this work without his support at every step along the way.

PERTURBATION METHODS FOR
IMAGE SYNTHESIS

by

Min Chen

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Abstract

This thesis presents new mathematical and computational tools for applying perturbation methods to image synthesis. The key idea is to characterize families of closely related optical paths by expanding a given path into a high-dimensional Taylor series. Our methods are based on the closed-form expressions for linear and higher-order approximations of specularly reflected rays of light, which are derived from the Implicit Function Theorem and Fermat's Variation Principle. The expressions hold for general multiple-bounce specular paths and provide a mathematical foundation for exploiting path coherence in incremental rendering. To illustrate their use, new algorithms based on the perturbation formulas are developed for several applications, including fast approximation of specular reflections on curved surface and direct caustic contour generation.

First, *path Jacobians* are introduced to formulate the linear perturbation of a ray path connecting a fixed point and a perturbed point through any number of intermediate reflection points. These Jacobian matrices are the derivatives of the reflection points with respect to the perturbed endpoint. A recurrence relation is derived to compute the closed-form expressions of path Jacobians for a general multiple-bounce reflection path, even when the new reflection path cannot be expressed in a closed form.

Next, the concept of path Jacobian is generalized to tensors of third order, which we call *path Hessians.* A simiar recurrence formula for path Hessians is derived from tensor calculus. The expressions for path Jacobians and path Hessians hold for any reflection path involving implicitly-defined reflective surfaces. Based on the close-form expressions for the path Jacobian and the path Hessian, a reflection path can be expanded as a Taylor series up to the second order. This perturbation formula gives rise to a fast and accurate interpolation scheme for the unknown reflection paths nearby.

Finally, new algorithms are presented for some applications that will benefit from our perturbation method.

In one application, a new approach is presented for interactively approximating specular reflections in arbitrary curved surfaces. The technique is quite general as it employs local perturbations to interpolate point samples and is applicable to any smooth implicitly-defined reflecting surface that is equipped with a ray-intersection procedure. After ray tracing a sparse set of reflection paths with respect to a given vantage point and static reflecing 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 using the Taylor expansions of one or more nearby reflection paths. After preprocessing, the approach is fast and accurate enough to compute specular reflections of tessellated objects in arbitrary curved surfaces at interactive rates using standard graphics hardware.

In another application, the path Jacobian formula is used to directly compute caustic contours on a plane formed by a given point light source and an implicitly-defined specular surface. Caustic irradiance for a given point on the plane is computed using wavefront tracing and some classical results from differential geometry. Using the closed-form expression for path Jacobian, an analytic formula for the gradient of caustic irradiance is also derived. The level curves with constant caustic irradiance on the plane, *caustic contours*, are traced by numerically solving an isolux ordinary differential equation.

Table of Contents

List of Figures

Nomenclature

Symbol	*Page*	*Meaning*
$\mathbf{p}(p_1, p_2, p_3)$	11	The varied end point of a path
$\mathbf{p}'$	11	The new position for the varied endpoint
$\mathbf{q}(q_1, q_2, q_3)$	18	The fixed end point of a path
$\mathbf{x}(x_1, x_2, x_3)$	11	The reflection point of a one-bounce path
$\mathbf{x}'$	11	The new reflection point for the perturbed path
$\mathbf{x}_i$	27	The ith reflection point of a multiple bounce path
Ψ	18	The single path function, $\mathbb{R}^3 \rightarrow \mathbb{R}^3$
Ψ_i	19	The component of the path function Ψ, $\mathbb{R}^3 \rightarrow R$
$\mathbf{J}$	20	The single path Jacobian, 3×3 matrix
$\mathbf{H}$	20	The single path Hessian, third order tensor
ϵ	18	Perturbation quantity
$\Delta\mathbf{p}(\epsilon_1, \epsilon_2, \epsilon_3)$	18	Perturbation vector from $\mathbf{p}$
η	21	Refractive Index of a media
c	21	Speed of light in a vacuum
v	21	Velocity of light in a medium
$C(s)$	21	Parametric curve of a ray path
det	23	Determinant of a matrix
C^p	24	p-times continuously differentiable function space
$g(\mathbf{x})$	25	The implicit function defining the reflecting surface of a one-bounce path
$\mathbf{G}$	25	The reflecting surface of a one-bounce path
∇	25	Gradient Operator
λ	25	Lagrange multiplier
f	27	The single Fermat function, $\mathbb{R}^3 \rightarrow \mathbb{R}^4$
D	27	The single Fermat Jacobian, 4×3 matrix

Symbol	*Page*	*Meaning*
sub	27	Operator $Hom(\mathbb{R}^3, \mathbb{R}^4) \rightarrow Hom(\mathbb{R}^3, \mathbb{R}^3)$
$\mathbf{G_i}$	27	The ith reflecting surface of a multiple bounce path
$g_i(\mathbf{x})$	27	The implicit function defining the ith reflecting surface of a multiple bounce path
$F_{g_i}(\widehat{\mathbf{x}_{i-1}}, \mathbf{x}_i, \widehat{\mathbf{x}_{i+1}}, \lambda_i) = \mathbf{0}$	31	Fermat equation corresponding to the ith bounce
$\mathbf{J}_i^*$	30	The ith path Jacobian(w.r.t $\mathbf{p}$), 3×3 matrix
$\mathbf{J}_i$	30	The ith reflection Jacobian(w.r.t $\mathbf{x}_{i-1}$), 3×3 matrix
f_i	31	The ith Fermat function, $\mathbb{R}^3 \rightarrow \mathbb{R}^4$
f_{i1}	31	The ith reflection function, $\mathbb{R}^3 \rightarrow \mathbb{R}^3$
f_{i2}	31	The component of f_i mapping to the Lagrange multiplier, $\mathbb{R}^3 \rightarrow R$
$\mathbf{x}_i'$	34	The ith new reflection point for the perturbed path
aug	32	Operator $Hom(\mathbb{R}^3, \mathbb{R}^4) \rightarrow Hom(\mathbb{R}^4, \mathbb{R}^4)$
∇T	41	The gradient of a matrix T, third order array
$\nabla_m T$	41	The mth layer of ∇T, a matrix with the same dimensions as T
$\nabla(T_{ij})$	41	The gradient vector entry of ∇T, a vector
$\mathbf{H}_i^*$	43	The ith path Hessian(w.r.t $\mathbf{p}$), third order tensor
D_i	44	The ith Fermat Jacobian(w.r.t $\mathbf{x}_{i-1}$), 4×3 matrix
$\mathbf{v}$	66	Virtual vertex for $\mathbf{p}$
$\mathbf{v}'$	66	Virtual vertex after perturbation in $\mathbf{p}$
W	87	Wavefront surface
$\mathbf{t}$	82	Unit tangent direction to a surface or a curve
$\mathbf{n}$	83	Surface normal
$\mathbf{p}$	83	Principal normal of a curve

Symbol	*Page*	*Meaning*
κ_n	83	Normal curvature
κ_g	83	Geodesic curvature
ξ,η	84	Two principal directions
$\mathbf{u}$,$\mathbf{v}$	84	Two perpendicular tangent directions
κ_ξ, κ_η	84	Two principal curvatures
κ_u, κ_v	83	Two normal curvatures in the other two perpendicular directions
κ_{uv}	84	Geodesic torsion relating the two directions $\mathbf{u}$ and $\mathbf{v}$
$\mathbf{P}(\mathbf{w})$	82	The projection matrix around the vector $\mathbf{w}$
$\mathbf{Q}(\mathbf{p})$	100	The skew matrix defined for a vector $\mathbf{p}$
$\mathbf{S}$	81	Shape Operator
$\mathbf{I}$	82	Identity Matrix
s	83	Arc length parameter of a curve
τ	85	Torsion
$\mathbf{b}$	85	Binormal vector
$\mathbf{r}$	91	Caustic contour function on a plane M, $R \rightarrow M$
ϕ	92	Caustic Irradiance
$\nabla\phi$	92	Caustic Irradiance Gradient

Chapter 1

Introduction

One long-standing goal of computer graphics is to allow users to explore and manipulate realistically illuminated, geometrically complex environments. Ideally, such environments would also exhibit global illumination effects, such as lighting, specular reflections, caustics, refraction, shadows, and so on. As a consequence, users demand more flexibility and better interaction with accurately rendered scenes. They would like to be able to interact with a scene, accounting for both viewpoint change and object space change (including changes in position, lighting, attributes, etc.), and perceive that at least some degree of global illumination effects get updated interactively.

Unfortunately, in order to achieve reasonable performance, current global illumination systems typically trade freedom of viewer and scene motion for both scene complexity and physical accuracy; thus user interaction is severely limited or even prohibited. Currently, only Monte Carlo approaches [48] can handle a wide range of surface geometries, reflection models and lighting effects that occur in reality. However, it does not currently appear feasible to apply Monte Carlo methods to interactive context due to its notorious long converging time. To compromise, three methods are normally employed by current global illumination systems: ray tracing, radiosity and hybrid methods. Ray tracing [86] succeeds in producing brilliant specular and transparent effects, but its costly pixel-by-pixel path tracing and view-dependent property make it impractical for interactive use. Alternatively, radiosity algorithm [36] accounts for subtle color bleeding and penumbra effects, and its view-independent advantage has been exploited for interactive viewing in architecture walk-through. Some techniques [29, 31, 72] even extend radiosity to dynamic environments. However, radiosity is constrained by diffuse and polygonal environments. In order to compensate for each other, some hybrid systems [23, 74, 73, 83, 85] used multiple pass rendering to combine ray tracing and radiosity for more realistic global illumination effects.

However, one important reason why ray tracing and radiosity are impractical for interactive applications is that most rendering systems recompute the entire image of the scene without considering any coherence between the image sequences. In the light of this realization, the current trend in image synthesis research toward interactive rendering focuses on incremental rendering. That is, rather than rendering the entire scene, many researchers have considered updating only the elements affected by a change by exploiting temporal and object-space coherence. Along this line, many incremental rendering algorithms based on ray tracing and radiosity have been developed, providing a promising step toward our interactive goal.

The goal of this thesis is to explore new mathematical and computational tools for interactive rendering. The central contributions of this thesis fall in two areas: new theoretical results, and new rendering algorithms. We introduce perturbation theory into image synthesis by exploiting path coherence in interactive image sequences, and derive closed-form expressions for linear and second-order approximations of a general specular reflection path through arbitrary implicitly-defined surfaces. Then, these analytical results have been used to develop new algorithms for rendering specular reflections and caustic contours.

1.1 Previous Work in Interactive Rendering

Typically, there are two strategies towards interactive rendering. One is to improve the performance of global illumination algorithms to decrease the computation time for a single image, the other is to exploit temporal and spatial coherence among adjacent frames to accelerate the generation of image sequences. Systems that aim at interactive rendering can be categorized by the global illumination algorithm they use. We first review some related work based on ray tracing, radiosity and image-based rendering(IBR) techniques, respectively.

1.1.1 Ray Tracing

Ray tracing is an attractive method to generate high quality images because it can produce some realistic optical effects such as reflection, refraction and shadows. Two operations are involved in a ray tracing process: ray casting to find a ray path and shading computation along the path. Compared to the shading computation, path finding operation is much more expensive. Therefore, many ray tracing acceleration techniques have concentrated on reducing the cost of ray casting by exploiting various kinds of coherence between rays. This has been done by casting bundles of rays, such as beams [41] and cones [3]; employing a fixed-grid [32] or adaptive 3D spatial hierarchy of scene objects [34] and directional techniques [6]. A good summary of these techniques can be found in Arvo and Kirk [7].

Besides the above techniques used to accelerate the generation of a single ray-traced image, some researchers have addressed the problem of generating ray-traced image sequences rapidly for interactive viewing or manipulative scenes whose geometry or material properties change with time. Most incremental rendering methods are based on the observation that great similarities exist between the reference image before the interaction and that after. This fact is obvious in the case of geometry change in which only a small local area is affected by the dynamic object. Although locality is not true for the viewpoint change, close correspondence can be seen among the pixels of the two images. Thus, instead of re-rendering the whole image from scratch, we can explore incremental recomputation by exploiting temporal, frame-to-frame coherence, and avoid redundant computation. Currently, three techniques are often employed to infer the new ray-traced image from the reference image: *reuse*, *reprojection* and *interpolation*.

Reuse methods take advantage of image space coherence to reuse as many pixels (and their associated paths) from the previous image as possible, and only updates the pixels (paths) affected by a change. It is very effective for geometrical and material changes, in which the ray paths associated with most pixels are unchanged. By doing some book-keepings for the next frame, reuse methods trade space for time.

As for the change of color parameters only, a color tree is enough for updating the way of illumination calculation. Cook [26] conserved a shade tree for the symbolic evaluation of the illumination model for each surface. Since the color manipulation does not modify the shape of the tree, the local illumination may be updated by simply evaluating the shade tree. Similarly, Séquin and Smyrl [71] preserved in a tree the color expressions of all intersections obtained by ray tracing. The image is updated by traversing these color trees with modified parameters. Both methods propose several compression techniques to reduce the memory usage and improve the display time.

To handle visibility changes, Murakami and Hirota [61] extended these previous techniques for a scene with geometrical manipulation. The space is subdivided into regular voxels, and a ray is indexed by the list of voxels it traverses. Any change to the scene is associated with it some affected voxels, which in turn determine the potential affected rays. Murakami and Hirota also used a hashing scheme to quickly identify the rays affected by a given voxel. However, the visibility determination is performed with respect to the affected voxels rather than the transformed objects. Therefore, all intersections between a ray and the objects in the affected voxels must be computed and saved, and recomputed whenever a voxel is affected. Jevans [46] instead stored in each voxel a tag to the original generating pixel (or block of pixels) that spawns the ray passing through the voxel. Changed voxels caused the appropriate pixels to be traced, a change in a ray of any level causes the entire pixel to be traced. Both methods must balance voxel subdivision and memory consumption.

Briere [18] presented a more general system that efficiently detects and recomputes the exact portion of the image that has changed after an arbitrary manipulation of a scene. He employed two tree structures to allow for such an incremental recomputation. The *color tree* preserves the entire expression leading to the final color of a pixel. Any changes affecting the value of parameters in these trees, such as shading and texture parameters, are quickly displayed by re-evaluating only the subtrees dependent upon the changing parameters. The *ray tree* preserves only the visibility specific information of the rays generated from a pixel. They are combined into *tunnels*, formed by a union of shafts, to quickly detect the modified ray segments with respect to the user selection, geometrical changes are handled by re-shooting rays from the previous valid intersection points. Some optimizations such as grouping and building hierarchy are also used to speed up the detection. However, the high memory requirement is a drawback.

In general, reuse methods usually retain a tree-like structure to keep track of the path information (including intersection points, material parameters, etc.) computed for each pixel from the previous image. During a scene interaction, they quickly detect the affected regions through some novel data structures. If a stored tree has been found unchanged, which means that not only the shape but also the intermediate node information are fixed, they just reuse the old path. Whenever a tree is found not exactly the same as before, new rays will be recasted from the valid intersection points to retrace the scene, lacking an efficient way to rapidly update a ray tree based on different tree variations. As a consequence, most reuse methods are usually good for interactive color changes, but cannot handle geometric changes fast enough for interactive use. Moreover, since image space coherence is destroyed by even a slight view point movement, most reuse methods are efficient for a static viewpoint.

Reprojection methods compensate for the viewpoint movement by moving pixels from the reference image to the current image. By exploiting perspective coherence, the pixel shifting is performed by a 3D warping equation. Badt [47], Adelson and Hodges [1] kept the intersections of rays with objects, and applied 3D warp to reproject them to the new view position. Each pixel covered by some reprojections receives the radiance that has been computed for the reprojected point in the reference image. Consider the special property of stereoscopic pairs, this 3D warp can be reduced to an image space reprojection(2D warp) in a stereoscopic ray tracing [2]. Such a reprojection algorithm speeds up the first level rays from the eye to the screen and only works for diffuse objects. It can't accelerate the computation of specular and transparent effects.

Ray tracing computes the radiance returned to the eye along the sample ray from each pixel, interpolation methods try to approximate the radiance along a new ray based on the known radiance along the sample rays. It has been observed that as a sample ray changes position or direction, the radiance along that ray (as computed from previous images) changes smoothly, except at some discontinuous cases. By exploiting such object-space coherence in radiance function, Teller et al. [78] presented an algorithm to speed up the shading computation in ray tracing. The algorithm lazily constructs conservative 4D radiance interpolants in ray space, which are quadrilinearly interpolated to satisfy new radiance queries. By identifying four sources of discontinuity, interpolation across discontinuities is avoided. The constructed 4D interpolants are reused by reprojection to the new viewpoint. Kavita et al. [11] extended this algorithm by identifying radiance non-linearity and provided an error bound for radiance interpolation. Presently, their algorithm can account for viewpoint movement, although further work on extension to object space change is needed.

Besides the above three techniques, Chapman et al. [19, 20] designed a method to use the trajectory of the viewpoint through the scene to compute “continuous intersection” information of rays with the scene through time, so a pixel need only be retraced when the current intersection ends. The restriction of his method is a polygonal scene model. Glassner [35] proposed spacetime ray tracing for animation sequence, which renders dynamically moving objects in 3D space by rendering static objects in 4D spacetime. Adaptive space subdivision and bounding volume techniques are combined to create non-overlapping hierarchy to accelerate the 4D ray intersection, but his method is only designed for pre-defined animation paths.

1.1.2 Radiosity

Radiosity [36] is the global illumination model of choice for polygonal, diffuse environments. Its view-independence advantage makes it ideal for the interactive walk-through of static simulated environments. However, the computation time for a radiosity solution is still prohibitive for interactive scene manipulation. The fact that the movement of an object often causes limited changes to a global illumination solution suggests incremental radiosity for dynamic scenes. Up to now, significant advances have been made towards accelerating radiosity calculation for dynamic scenes and several algorithms [33, 22, 60, 31, 72, 29] have been proposed to deal specifically with changes of geometry, their evolution follows closely with the progress of radiosity method.

One of the most computational expensive operations in classical "full-matrix" radiosity method is the computation of form factors. Baum et al. [13] developed an algorithm to take advantage of coherence properties of the hemi-cube to calculate the form factors, which is the first extension of the radiosity method to dynamic environments. Designed for scene animation, this solution is limited to the predefined trajectories for all the dynamic objects, and is therefore not suitable for general interaction.

The introduction of progressive refinement techniques to radiosity [24] resulted in quick coarse solutions and made it possible to extend it to truly dynamic environments. In 1990, two similar approaches which take advantage of "shooting" process of radiosity propagation were published independently by George [33] and Chen et al. [22]. The two are both based on progressive refinement method and redistributing the energy of an environment after interactive modification of its scene geometry or surface attributes. By treating all object movements as deletion and re-insertion into the scene, shadows are removed from the newly visible patches by reshooting "positive" energy, and new shadows are correctly inserted to the newly shadowed patches by shooting "negative" energy. And for each patch which have shot before, its energy contribution to the scene based on the old geometry is removed and then reshoot based on the new geometry. Scene coherence is exploited by either hardware clipping to a hemicube or using a general shadow volume. An improvement to these methods was developed by Müller [60] by adding an intelligent data structure maintaining shadow-lists accelerating potentially modified interaction between surfaces. Due to the difficulty of controling the global energy balance in progressive radiosity, these methods remain very expensive, although they achieve impressive update times for direct illumination.

The limitations of progressive refinement can be lessened in the context of hierarchical radiosity [38]. The global energy balance is maintained in the link structure and the corresponding hierarchy. When an object moves, only a limited number of links and a limited part of the hierarchy are affected, which can be used to accelerate the update. Forsyth et al. [31] presented the idea of "promoting" and "demoting" links based on a refinement criterion which moves links up and down in the hierarchy, depending on the position of the objects at each frame. Shaw [72] used a similar idea of “ghost” links to do mesh unfolding for adaptive hierarchy. Also, she introduced special “shadow” links connected to the source and containing blocker information to approach the visibility change problem. To provide a unified mechanism which can rapidly identify the part of the system modified, Drettakis et al. [29] introduced a line-space hierarchy associated with the links between scene elements, which can be traversed efficiently using the implicit correspondence between it and the original hierarchy of surface and clusters in the scene. Moreover, a shaft structure is created for each link to represent a portion of line segment space, which speeds up the identification. Although these algorithms achieve significant improvement over the progressive refinement approaches, interactive update rates are still not possible using these methods, especially for complex scenes.

A primary limitation of all these radiosity methods is that they are restricted to ideal diffuse reflectors, and are best suited for polygonal environments. Extensions of radiosity to account for specular reflection and transmission effects will substantially increase the computational time.

1.1.3 Image-Based Rendering

More recently, *image-based rendering* has emerged as an emerging and competing rendering paradigm. In contrast to the conventional geometry-based rendering, image-based rendering (IBR) techniques rely on interpolation using the original set of input images or pixel reprojection from source images onto the target image in order to produce a novel virtual view. A significant advantage of image-based rendering is that the cost of rendering is independent of the scene complexity, which makes it very promising for interactive viewing and animation. Based on the nature of the scheme for pixel indexing or transfer, Sing [49] classified four distinct (but not necessarily mutually exclusive) categories of image-based rendering techniques. They are: *non-physically based image mapping*, *mosaicking*, *geometrically-valid pixel reprojection* and *interpolation from dense samples*.

Non-physically based image mapping does not consider 3-D geometry at all in the pixel location computation. Techniques used include Beier and Neely’s feature-based morphing [14], 2-D spline meshing mapping [87], and linear combination of a training set of images such as human faces [82]. These techniques are widely used in the advertising and entertainment industries for some special morphing effects, not suitable for those interactive applications where physically correct images are important. Mosaics [77, 76] refers to the combination of at least two different images to yield a higher resolution image or larger view, which is not our goal in interactive rendering. Therefore, we mainly discuss the image-based techniques in the last two categories and their applications in interactive rendering, particularly, interactive viewing.

Geometrically-valid pixel reprojection makes use of some geometrical information for pixel shifting, and the source image it starts with normally combines the depth value for each pixel, that is, a range image. Given a range image, the change of each pixel location from the reference view to the desired view position is constrained in a predictable way, which can be described by a 3D warping equation [21, 58, 57]. This warping is called *3D warp* since the range (depth) information is involved in the computation. In addition, McMillan et al. [58, 57] also proposed an occlusion-compatible warping ordering to solve the object occlusion and dis-occlusion problem. This warping technique has been used in many systems for view interpolation in interactive viewing. Chen and William [21] used a 3D warp in a pre-processing step to compute the warp vectors between images. They subsequently performed quadtree decomposition of the image by grouping these warp vectors and linearly interpolated them to create the intermediate views. Similar range image interpolation is used in a framework proposed by Nimeroff et al. [63] for efficient global illumination in complex, animated environments, they exploited coherence by computing direct and indirect illumination separately in both space and time and interpolated the sparsely sampled indirect illumination. Mark et al. [55] presented a reconstruction algorithm to remove holes, which treats the reference images as a mesh and warps the mesh triangles to the current image. They avoided occlusion-related artifacts by warping two different reference images and compositing them. However, although the image warping combined with interpolation has improved the rendering performance tremendously, it does not correctly handling non-diffuse scenes [53], such as specular mirrors, glossy surfaces, etc. Moreover, since image warping can only use the information appearing in the reference images, it can not correctly handle the case that an invisible part in the reference image becomes visible in the new view.

The Lumigraph [37] and Light field rendering systems [51] take a very different approach to image-based rendering from the warping systems we have just discussed. Rather than storing a limited number of images and then warping to interpolate between them. They store a set of dense, regular sampling of all possible light rays with a 4D parameterization of the ray spaces. Then the radiance function in the new viewpoint can be built from interpolation of the stored dense samples. The interpolation scheme used by Levoy and Hanrahan [51] approximates the resampling process by simply interpolating the 4D function from nearby samples, using the nearest neighbor, bilinear or quadrilinear interpolation. In Gortler et al.'s approach [37], a continuous Lumigraph is reconstructed by the linear sum of the product between a basis function and the value at each grid point in 4D. Geometric information can be used to guide the choice of basis function. Light field rendering or Lumigraph can perfectly solve the problems in 3D image warping, such as non-Lambertian shading. But it has a data acquisition pre-processing phase, not intended for interactively rendering dynamic scenes. Also the viewer is constrained to lie outside the convex hull of the scene. Finally, it requires a large amount of memory, despite agressive compression techniques.

1.2 Path Perturbation

In the previous section, we gave a brief review of previous and current research efforts towards the goal of interactive rendering in three different areas. It can be seen that incremental radiosity and image-based rendering (except light field rendering) both share the limitation of diffuse scenes. Although some radiosity algorithms have been adjusted to handle specular effects, they are essentially multi-pass rendering by combining another ray tracing pass, which makes them much slower for interactive use. On the other hand, it is very difficult for pure image based rendering techniques to generate a physically valid novel view without any geometric information. In the light of these observations and our goal of pursuing fast, accurate interactive rendering algorithms, we choose ray

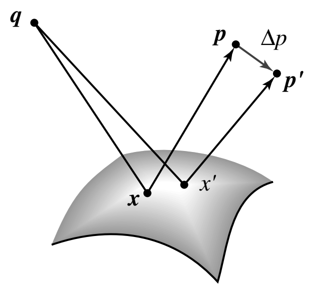


Figure 1.1: *Reflections tend to have a great degree of coherence, the corresponding reflection points* $\mathbf{x}$ *and* $\mathbf{x}'$ *will converge as* $\mathbf{p}$ *approaches* $\mathbf{p}'$*.*

tracing as our basic light transport algorithm, and attempt to build some rapid update scheme on top of it, accounting for the change of some prominent optical effects with respect to the movement of either the view point or the objects.

1.2.1 Path Coherence

Let us consider what happens to a given path connecting points $\mathbf{p}$ and $\mathbf{q}$ when $\mathbf{p}$ moves slightly. See Figure 1.1. Changes in the path can take one of two forms: hitting a different object; intersecting at a different point on the same surface. It is the latter case that suggests that there exists another type of coherence with the nature of path perturbation, which should be explored to update these two cases differently to improve efficiency.

For a scene consisting of both diffuse and pure specular objects, an object point can be seen from the view point somewhere on the image plane, via some reflection points on the curved reflecting surfaces. This mapping from a 3D point to its reflection points is non-linear, characterized by a particular specular reflection path from the scene point to the vantage point. For smooth reflecting surfaces, it is likely that the reflection points along the path will vary continuously as a function of the object point or the vantage point, except at the boundary. That is, reflections tend to have a great degree of coherence; as two objects grow nearer to each other, so will their reflections. As shown in Figure 1.1, as $\mathbf{p}$ approaches $\mathbf{p}'$, the corresponding reflection points $\mathbf{x}$ and $\mathbf{x}'$ will converge.

Therefore, the reflection paths corresponding to the neighborhood of the view point

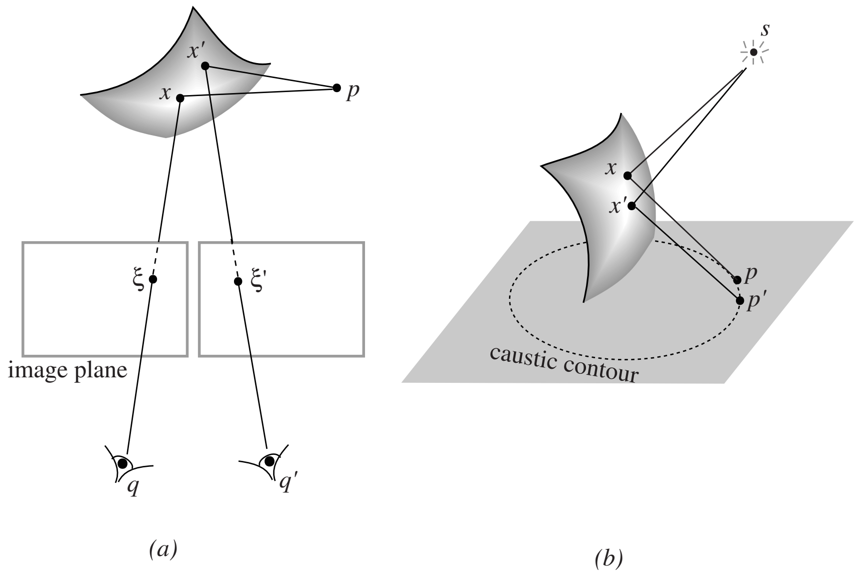


Figure 1.2: *Two examples of path coherence in image synthesis. (a) Image warping transforms a reference image to the desired image at the perturbed view position* $\mathbf{q}'$. *(b) Perturbation can be used to find indirect illumination path from the point light* $\mathbf{s}$ *to the nearby point on caustic contours.*

or the neighborhood of the object point are very similar. While exceptions occur near object boundaries, this coherence can be exploited. To make use of such path coherence inherent in neighboring images, we must find a connection between these similar paths so that we can rapidly infer one from the other. In doing this, some physical laws, especially light transport principles must be taken into account.

Path coherence has arisen in a number of guises in image synthesis. In image-based rendering, consider a collection of images of the same static scene with respect to nearly identical view positions. Even though each view position is associated with a different ray path from the same visible scene point, it is likely that nearly all these ray paths hit the same scene objects at slightly different positions, except that the rays hit the object boundaries. In particular, strong path coherence exists between stereoscopic image pairs. Thus, image warping can be used to transform a finite set of reference images to the desired image at the perturbed view point. See Figure 1.2(a). To render caustics on a plane due to a point light source and a reflecting surface, we must find the reflective illumination paths from the light to every plane point. If we define caustic contours as the level curves on the plane with constant irradiance from the light source, then similar caustic paths exist among neighboring points on the plane. See Figure 1.2(b). Without use of path coherence, Mitchell and Hanrahan [59] had to solve a non-linear system numerically for the reflection path at each plane point to compute its irradiance value.

While it has been observed from many image synthesis applications that great similarity exists among the ray paths from a common fixed point to a small neighborhood of a perturbed point, it has proven difficult to exploit this type of coherence. Although several mathematical formulations have been offered [1] for diffuse scenes, none apply to general reflecting paths. In this thesis, we introduce perturbation theory into image synthesis and propose a new approach to exploit path coherence through differential forms of a path.

1.2.2 Perturbation Methods

During the past two decades, many branches such as applied math, continuous mechanics, heat transfer, etc, have relied heavily on fully numerical procedures involving finite differences and finite elements. Despite this overwhelming trend, approximate analytical methods have continued to develop and provide useful solutions to a variety of problems. One such method is the method of *perturbation expansion.* Given the solution to a certain mathematical problem, how is the solution determined when the conditions of the problem are slightly altered? The systematic answer to this question forms the subject of *perturbation theory* [52].

The basic tools in the perturbation analysis is the Taylor expansion. To answer the question posed above we consider a solution to the given problem as a function of a *perturbation quantity* such that the solution corresponds to the function value when the perturbation is zero. The slightly altered conditions of the problem can be formulated as a small delta away from the zero value of the perturbation quantity. Under such mathematical formalization, the solution to be determined under the changed conditions can be approximated by expanding the parameterized solution function around the given solution with Taylor series. What is left then is how to compute the coefficients in the Taylor series, which varies depending on the application.

Analysis based on perturbation theory is approximate, since we must truncate the sequence in the expansion. However, the approach plays an important role in yielding fast solutions for equations that cannot be solved exactly. Perturbation methods have been applied to a wide range of disciplines. The first comprehensive book on perturbation methods was written by Van Dyke [30], with a focus on fluid mechanical applications. Cole [25] introduced a text from the point of view of applied mathematics. Aziz [10] gave a review of the various applications of perturbation methods in heat transfer area. Comprehensive material on perturbation methods can be found in the latest books by Nayfeh [62], Lin and Segel [52], Hinch [42], Bender and Orszag [15]. In this thesis, we expand the list to include image synthesis.

In the context of interactive rendering, the natural connection between path coherence property just discussed and perturbation theory suggests that path perturbation approach can be employed to interactively update a specular path with respect to the small movement of the eye or the object. That is, we can reformulate a general specular path as a high dimensional function relating the vantage point or the object position to all the intermediate reflection points along the way. Then, given an exact path connecting a viewpoint and an visible point in 3D, which is known from previous ray tracing or other channels, the problem of inferring a new path from this known path to handle the perturbations in the eye point or the visible point falls into the application domains of perturbation theory. Henchforth, all the techniques of perturbation methods can be re-examined here, this is an important contribution of this thesis.

1.3 Thesis Overview

This thesis introduces perturbation theory to image synthesis and presents new mathematical and computational tools for interactive rendering. The work is aimed at achieving interactive rates during the scene interaction and also correctly accounting for some prominent optical effects, such as specular, refraction, caustics, etc. The goal is met by exploiting path coherence among image sequences and generating families of closely related optical paths by expanding a given path into a high dimensional Taylor series, which is accomplished by deriving the closed-form expressions for linear and higher-order approximations of this series. This constitutes the theory foundation of this thesis. The practical aspect of our work focuses on applying the perturbation formula derived to solve some challenging rendering problems. New rendering algorithms are proposed to approximate two important optical effects rapidly and accurately, one is specular reflections, the other is caustics.

Chapter 2 is the core mathematical foundation of this thesis, it introduces a powerful tool for interactive rendering. The tool is an analytical perturbation formula for a general reflecting path with respect to the perturbation of either end point, which comes from the high dimensional Taylor expansion of a path up to the second order. The chapter introduces the concepts of *path Jacobian* and *path Hessian* to describe the linear and quadratic approximations of an optical path. After formulating a path mathematically in accordance with *Fermat's principle*, it provides a complete derivation for the closed-form expressions for path Jacobians and path Hessians from *Implicit Function Theorem* and tensor differentiation. The interesting aspect of these formulas is in correctly handling multiple bounce reflections, which are expressed in a recurrence relation.

Chapter 3 demonstrates a direct application of the path perturbation formula derived in Chapter 2 and presents a new approach to interactively approximate specular reflections in arbitrary curved surfaces. The technique employs local perturbations to interpolate point samples computed from pre-processing, and is applicable to any smooth implicitly-defined reflecting surface. The result shows that this approach is fast enough to compute reflections of tessellated objects in arbitrary curved surfaces at interactive rates using standard graphics hardware, and the quality is indistinguishable from ray traced images.

Chapter 4 shows another application of the analytic expressions for path Jacobians and path Hessians. By incorporating path Jacobian formula for a caustic path with wavefront tracing and curvature computation, we can compute the radiant intensity gradient due to a point light source, which allows us to directly trace the caustic contours formed by a curved surface and a point light source from numerically solving an ordinary differential equation for iso-contour curves on a plane. Some singular cases to make the numerical solver fail are also discussed.

The last chapter summarizes the research and discusses directions for future work. It is hoped that the theory of path perturbation may also find potential applications in global illumination with explicit paths involved, and provides new insights to accelerate some global illumination algorithms.

Chapter 2

Derivatives of Path

As a basic mathematical tool to characterize function differentiability, various kinds of derivatives have been studied in computer graphics in order to accurately, quantitively model features such as discontinuity, variations, etc. Derivative information has played an important role in mesh construction, guiding importance sampling and improving function interpolation. In previous work, Heckbert [40] and Lischinski et al. [54] identified derivative discontinuities in irradiance and used them to construct effective surface meshes. Irradiance gradient has been used to get higher order approximation of irradiance function. For example, Ward et al. [84] and Vedel [80, 81] estimated irradiance gradients by Monte Carlo path tracing and used them to improve the interpolation accuracy. Salesin et al. [68] and Bastos et al. [12] employed gradients to construct higher-order interpolants for irradiance functions. Drettakis et al. [28] estimated gradients as well as isolux contours from a collection of discrete samples and used them to guide subsequent sampling, placing more samples where the curvature of the isolux contours was large. Arvo [5] extended the derivative to a vector function and derived the closed-form expression for the irradiance Jacobian, the derivative of the vector representation of irradiance. The expression holds for any number of polyhedral blockers, and was used for direct computation of isolux contours, finding local irradiance extrema and iso-meshing. The similar idea was also employed by him to generate the iso-contour in volume rendering [8].

Using the similar derivative technique in a high dimensional setting, this chapter introduces a new computational tool for non-Lambertian environments which consists of diffuse or specular curved surfaces. The tool is very useful in incremental rendering where strong path coherence exists from frame to frame. The central contribution of our work is the closed-form expressions for the first-order and second-order approximations of a specular path. These perturbation formulas also hold for multiple bounce paths, in which a recurrence relation is presented.

2.1 Introduction

As we mentioned in Section 1.2.2, our perturbation method is motivated by the fact that reflections tend to have a great degree of coherence; as two object points grow near to each other, so will their reflections. To exploit such path coherence using perturbation theory, The key step is to formulate the problem of updating a reflection path in the context of interactive rendering as a perturbation problem. Specifically, the reflection point $\mathbf{x}$ of a given one-bounce reflection path connecting $\mathbf{q}$ and $\mathbf{p}$ can be considered as a function[1] of the two points, that is,

$$\phi : \mathrm{I\!R}^3 \times \mathrm{I\!R}^3 \to \mathrm{I\!R}^3,$$

where $\phi(\mathbf{q}, \mathbf{p})$ returns the scene coordinates of the point $\mathbf{x}$ on the reflecting surface. By fixing one endpoint $\mathbf{q}$, the function $\phi(\mathbf{q}, \cdot)$ can be viewed as a mapping from a 3D point to its reflection:

$$\Psi : \mathrm{I\!R}^3 \to \mathrm{I\!R}^3, \tag{2.1}$$

where $\Psi(x) \equiv \phi(\mathbf{q}, x)$. We call Ψ the *path function* with respect to the point $\mathbf{p}$. For a small ϵ, we now consider $\Psi(\mathbf{p} + \Delta\mathbf{p})$ where $||\,\Delta\mathbf{p}\,|| < \epsilon$, and obtain an asymptotic approximation to the new path function by means of a Taylor expansion.

The path function $\Psi : \mathrm{I\!R}^3 \to \mathrm{I\!R}^3$ relating a reflection point $\mathbf{x}$ to the perturbed point

[1]More precisely, ϕ is a relation since there may exist several reflection points for two given points, but locally it is a function.

$\mathbf{p}$ is a vector-valued function taking a vector argument. The Taylor series of Ψ can be expressed in Cartesian tensor notation as

$$\begin{aligned} \Psi_i(\mathbf{p} + \Delta\mathbf{p}) &= \Psi_i(\mathbf{p}) + \sum_j \Psi_{i,j}\epsilon_j \\ &+ \frac{1}{2!}\sum_{jk} \Psi_{i,jk}\epsilon_j\epsilon_k \cdots \\ &+ \frac{1}{n!}\sum_{jk\ldots r} \Psi_{i,jk\ldots r}\epsilon_j\epsilon_k \ldots \epsilon_r + \cdots \end{aligned}$$

where $\Delta\mathbf{p} = (\epsilon_1, \epsilon_2, \epsilon_3)$ is the perturbation of $\mathbf{p}$, $\Psi = (\Psi_1, \Psi_2, \Psi_3)$, and

$$\begin{aligned} \Psi_{i,j} &= \frac{\partial \Psi_i}{\partial p_j}, \\ \Psi_{i,jk} &= \frac{\partial^2 \Psi_i}{\partial p_j \partial p_k}, \\ &\cdots \end{aligned}$$

for $i, j, k = 1, 2, 3$. Here, p_j, p_k represent components of the independent variable $\mathbf{p}$, and all partial derivatives are evaluated at the given path through $\mathbf{p}$.

To obtain a second-order approximation of Ψ, we truncate the sequence after the first two dominant terms. Thus,

$$\Psi_i(\mathbf{p} + \Delta\mathbf{p}) = \Psi_i(\mathbf{p}) + \sum_j \Psi_{i,j}\epsilon_j + \frac{1}{2}\sum_{jk} \Psi_{i,jk}\epsilon_j\epsilon_k + \mathrm{O}(\|\, \Delta\mathbf{p} \,\|^3).$$

Collecting the coefficients appearing in the three expansions of Ψ_1, Ψ_2 and Ψ_3 and putting them into familiar matrix forms, we obtain the Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \Psi_{1,1} & \Psi_{1,2} & \Psi_{1,3} \\ \Psi_{2,1} & \Psi_{2,2} & \Psi_{2,3} \\ \Psi_{3,1} & \Psi_{3,2} & \Psi_{3,3} \end{bmatrix}, \tag{2.2}$$

and three Hessian matrices

$$\mathbf{H}_i = \begin{bmatrix} \Psi_{i,11} & \Psi_{i,12} & \Psi_{i,13} \\ \Psi_{i,21} & \Psi_{i,22} & \Psi_{i,23} \\ \Psi_{i,31} & \Psi_{i,32} & \Psi_{i,33} \end{bmatrix}, \tag{2.3}$$

for $i = 1, 2, 3$, which constitutes a third order tensor $\mathbf{H}$. In terms of $\mathbf{J}$ and $\mathbf{H}$, the second-order Taylor expansion of the path function Ψ about the given path through $\mathbf{p}$ can be expressed as:

$$\Psi(\mathbf{p} + \Delta\mathbf{p}) = \Psi(\mathbf{p}) + \mathbf{J}\Delta\mathbf{p} + \frac{1}{2} \begin{bmatrix} \Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}_1\Delta\mathbf{p} \\ \Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}_2\Delta\mathbf{p} \\ \Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}_3\Delta\mathbf{p} \end{bmatrix} + \mathrm{O}(\|\, \Delta\mathbf{p}\, \|^3), \tag{2.4}$$

which is the *second-order perturbation formula* to update a given path through $\mathbf{p}$ to a new path to the nearby point $\mathbf{p} + \Delta\mathbf{p}$. Thus, $\mathbf{J}$ and $\mathbf{H}$ are actually the first- and second-order derivatives of the path function Ψ, that is,

$$\mathbf{J} \equiv \frac{\partial \Psi(\mathbf{p})}{\partial \mathbf{p}},$$

and

$$\mathbf{H} \equiv \frac{\partial^2 \Psi(\mathbf{p})}{\partial \mathbf{p}^2}.$$

We shall refer to the first derivative $\mathbf{J}$ as the *path Jacobian* and the second derivative $\mathbf{H}$ as the *path Hessian*. The path Jacobian and path Hessian provide first- and second-order approximations to Ψ, respectively. Since Ψ finds the reflection point on a curved surface, the accuracy of this approximation depends on the local curvature of the reflecting surface. In general, the linear approximation suffices for nearly flat surfaces, while for more curved surfaces higher terms must be taken into consideration.

In order to apply the perturbation formula (2.4) to perturbed paths, we must compute $\mathbf{J}$ and $\mathbf{H}$ at a known path. To do this, we require tools from geometric optics and elementary classical analysis, which will be briefly reviewed in Section 2.2. This review covers Fermat's Principle, Lagrange Multiplier Theorem and Implicit Function Theorem. In Section 2.3, we derive the formula for path Jacobians with the aid of the Implicit Function Theorem and Fermat's principle. We start with a single-bounce reflection path, and then extend the result to a multiple-bounce path, present a recurrence relation between path Jacobians for a N-bounce path. Based on the path Jacobian formula derived in Section 2.3, Section 2.4 applies tensor differentiation to extend it to a higher order, and derives the closed-form expression for the third-order path Hessian tensor. Similarly, a recurrence relation is obtained for path Hessians of a multiple bounce path. Finally, the computation of path Jacobians and path Hessians for a N-bounce path has been summarized with pseudo code in Section 2.5.

2.2 Preliminaries

2.2.1 Fermat's Principle

In geometrical optics, the propagation of the light obeys *Fermat's principle*, also known as the principle of the *shortest optical path.* This principle asserts that the optical length of an actual light ray between any two points P_1 and P_2 is a local extremum among all paths between these points, within a small neighborhood [17, pp. 128–129]. Here, the *optical length* from one point on a ray to another is defined as the geometric path length weighted by the refractive index of the medium, η:

$$\int_C \eta ds, \tag{2.5}$$

where $C(s)$ is the parametric path from P_1 to P_2 and s is arc length. Since the light propagates with a velocity $v = c/\eta$ along the ray, we can write equation (2.5) in terms of time:

$$\int_C \eta ds = \int_C \frac{c}{v} ds = c \int_C dt.$$

Therefore, Fermat's principle is also known as the *principle of least time.*

Fermat's principle is a fundamental principle underlying geometrical optics. It stipulates the light travels along paths of stationary optical length, which implies that the paths followed by photons traveling through any medium should be locally maximal or minimal in terms of either distance or time. This property suggests that determining an actual path (or *Fermat path*) between any two 3D points can be reduced to a problem of variational calculus. In a ray tracing setting where regions of constant refractive index are separated by smooth boundaries, rays will travel along piecewise straight paths, reflecting or refracting at boundaries. The optical length function d between two arbitrary points $\mathbf{p}$ and $\mathbf{q}$ is simply the summation of piecewise line segment lengths weighted by their corresponding refractive indices, that is,

$$d(\mathbf{x}_0, \ldots, \mathbf{x}_{N+1}) = \sum_{i=0}^{N} \eta_i \sqrt{(\mathbf{x}_i - \mathbf{x}_{i+1})^2}. \tag{2.6}$$

where $\mathbf{x}_0 = \mathbf{p}$ and $\mathbf{x}_{N+1} = \mathbf{q}$. For a reflection path in a uniform medium, equation (2.6) simplifies further since the refractive indices are constant, and may be assumed to be one.

2.2.2 Lagrange Multiplier Theorem

The Fermat path problem represents a class of optimization problem with equality constraints, that is,

$$\text{minimize} \quad f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{h}(\mathbf{x}) = \mathbf{0},$$

where $\mathbf{x} \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$ and $m \leq n$. Here, f is the *objective function* and $\mathbf{h}$ is the *constraint function*. The *Lagrange Multiplier Theorem* provides a first-order necessary condition for the local extrema.

Theorem 1 (Lagrange Multiplier Theorem) *Let $\mathbf{x}^*$ be a local extremal point of $f : \mathbb{R}^n \to \mathbb{R}$, subject to $\mathbf{h}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$, $m \leq n$, and $\mathbf{x}^*$ is a regular point. Then, there exists $\lambda^* \in \mathbb{R}^{\mathbf{m}}$ such that*

$$\nabla f(\mathbf{x}^*) + \lambda^{*\mathrm{T}} D\mathbf{h}(\mathbf{x}^*) = \mathbf{0}^{\mathrm{T}}, \tag{2.7}$$

where

$$D\mathbf{h}(\mathbf{x}^*) = \begin{bmatrix} \nabla h_1(\mathbf{x}^*) \\ \vdots \\ \nabla h_m(\mathbf{x}^*) \end{bmatrix}$$

is the Jacobian matrix of $\mathbf{h} = [h_1, \ldots, h_m]^{\mathrm{T}}$ *at* $\mathbf{x}^*$.

We refer to the vector λ^* in the above theorem as the *Lagrange multiplier vector.* For a special case of only one constraint, where $n = 3$ and $m = 1$, the Lagrange condition (2.7) becomes

$$\nabla f(\mathbf{x}^*) + \lambda^* \nabla h(\mathbf{x}^*) = \mathbf{0}. \tag{2.8}$$

Here the scalar λ^* is referred as the *Lagrange multiplier.*

2.2.3 The Implicit Function Theorem

By means of Lagrange Multipliers, we can easily specify an implicit equation for the perturbed reflection problem; however, it is generally impossible to extract a closed-form solution from the resulting non-linear equations. Fortunately, the *Implicit Function Theorem* (IFT) provides a method for explicitly computing the *derivative* of such an implicitly-defined function without first knowing an explicit form of the function. It is this tool that allows us to derive the path Jacobian.

Theorem 2 (Implicit Function Theorem) *Let* $A \subset \mathbb{R}^n \times \mathbb{R}^m$ *be an open set and let* $F : A \to \mathbb{R}^m$ *be a function of class* C^p. *Suppose* $(\mathbf{x}_0, \mathbf{y}_0) \in A$ *such that* $F(\mathbf{x}_0, \mathbf{y}_0) = 0$ *and*

$$\det \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \cdots & \frac{\partial F_1}{\partial y_m} \\ \vdots & & \vdots \\ \frac{\partial F_m}{\partial y_1} & \cdots & \frac{\partial F_m}{\partial y_m} \end{bmatrix} \neq 0, \tag{2.9}$$

where $F = (F_1, \ldots, F_m)$, *and the Jacobian matrix is evaluated at the point* $(\mathbf{x}_0, \mathbf{y}_0)$. *Then there exists an open neighborhood* $\mathbf{x}_0 \in U \subset \mathbb{R}^n$, *a neighborhood* $\mathbf{y}_0 \in V \subset \mathbb{R}^m$, *and a unique function $f : U \to V$ such that*

$$F(\mathbf{x}, f(\mathbf{x})) = 0 \tag{2.10}$$

for all $\mathbf{x} \in U$. *Furthermore,* $f \in C^p$.

See Marsden and Hoffman [56, pp.211–213] for a proof of the Implicit Function Theorem. By differentiating both sides of equation (2.10) with respect to the independent variable $\mathbf{x} \in \mathbb{R}^n$, we obtain the following well-known corollary:

Corollary 1 *The Jacobian matrix of the implicit function f in Theorem 2 is given by*

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = - \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \cdots & \frac{\partial F_1}{\partial y_m} \\ \vdots & & \vdots \\ \frac{\partial F_m}{\partial y_1} & \cdots & \frac{\partial F_m}{\partial y_m} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix} \tag{2.11}$$

where the inverse on the right hand side exists, since its determinant is nonzero.

Note that the Jacobian matrix of the implicit function f is a linear combination of derivatives of the known function F.

2.3 The Path Jacobian

The path Jacobian, which is the first derivative of path function Ψ, can be derived with the aid of the Implicit Function Theorem and Fermat's principle. In the next sections, after translating this problem into a mathematical model from Fermat's principle and the method of Lagrange Multiplier, we derive the path Jacobian for the simple one-bounce case with the aid of the Implicit Function Theorem. Then, we show two approaches to extend the path Jacobian for one-bounce paths to N-bounce paths and derive a recurrence formula for path Jacobians of a multiple bounce path. Finally, we verify the equivalence of these two approaches from a three-bounce example.

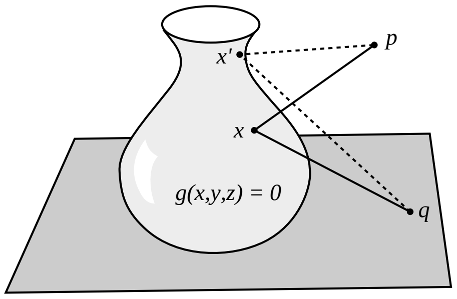


Figure 2.1: *Reflection paths obey Fermat's variational principle, stating that the length of the optical path connecting* **p** *and* **q** *is a local extremum.*

2.3.1 One-Bounce Path

Fermat's principle states that the propagation of light waves can be reduced to the study of paths along which the travel time is minimal. This principle is directly applicable to computing reflections from surfaces of arbitrary shape defined by implicit functions. Since a ray path assumes a local minima or maxima, it can be found via optimization. Moreover, the restriction that reflections lie on the implicitly-defined reflecting surfaces recasts the problem as a *constrained optimization problem.* Consequently, the method of Lagrange Multipliers can be applied, as demonstrated by Mitchell and Hanrahan [59].

Suppose $g(\mathbf{x}) = 0$ is the implicit definition of a reflecting surface $\mathbf{G}$, and $\mathbf{x}$ is a reflection point of a ray path connecting $\mathbf{p}$ and $\mathbf{q}$ in a homogeneous medium, as shown in Figure 2.1; by Fermat's principle, the path length assumes a local extremum. There may be many such paths connecting two points, such as the path $\mathbf{p} - \mathbf{x}' - \mathbf{q}$ shown in Figure 2.1. By applying Lagrange Multipliers to the length of the path, we obtain

$$\begin{aligned} \nabla d(\mathbf{p}, \mathbf{x}, \mathbf{q}) + \lambda \nabla g(\mathbf{x}) &= \mathbf{0} \\ g(\mathbf{x}) &= 0, \end{aligned} \tag{2.12}$$

where λ is a Lagrange multiplier, and $d(\mathbf{p}, \mathbf{x}, \mathbf{q})$ is the length of the optical path from $\mathbf{p}$ to $\mathbf{q}$ via $\mathbf{x}$ in a homogeneous medium. Thus,

$$d(\mathbf{p}, \mathbf{x}, \mathbf{q}) = ||\,\mathbf{p} - \mathbf{x}\,|| + ||\,\mathbf{q} - \mathbf{x}\,||\,.$$

By fixing one endpoint of the path, $\mathbf{q}$, and allowing $\mathbf{p}$ to vary, we may rewrite equation (2.12) as an implicit equation that relates $\mathbf{p}$, $\mathbf{x}$ and λ:

$$F(\widehat{\mathbf{p}}, \mathbf{x}, \lambda) = \mathbf{0}, \tag{2.13}$$

where $F : \mathrm{I\!R}^3 \times \mathrm{I\!R}^4 \to \mathrm{I\!R}^4$, and $\mathbf{p}$ is regarded as the independent variable, which we designate with a hat. We shall use this convention throughout. We refer to equation (2.13) as the *Fermat equation.* The explicit form of the Fermat equation can be derived by expanding the ∇ operator in equation (2.12), yielding

$$\begin{aligned} F_i(\mathbf{p}, \mathbf{x}, \lambda) &= -\frac{(p_i - x_i)}{||\,\mathbf{p} - \mathbf{x}\,||} - \frac{(q_i - x_i)}{||\,\mathbf{q} - \mathbf{x}\,||} + \lambda \frac{\partial g(\mathbf{x})}{\partial x_i} \\ F_4(\mathbf{p}, \mathbf{x}, \lambda) &= g(\mathbf{x}). \end{aligned} \tag{2.14}$$

where $i = 1, 2, 3$.

Unfortunately, it is generally impossible to solve these non-linear equations for the reflection point $\mathbf{x}$ in a closed form, even for trivial functions $g(\mathbf{x})$. Consequently, Mitchell and Hanrahan [59] resorted to various numerical methods, such as the interval Newton method and automatic differentiation techniques to solve for $\mathbf{x}$. However, the Implicit Function Theorem discussed in section 2.2.3 provides a tool to compute the derivative of the reflection point without knowing the explicit form of the function.

To show how the Implicit Function Theorem and its corollary can be applied to the path Jacobian computation, consider a ray reflected from a surface with implicit definition $g(\mathbf{x}) = 0$. Since this reflection path provides a solution $(\tilde{\mathbf{p}}, \tilde{\mathbf{x}}, \tilde{\lambda})$ to the Fermat equation (2.13), it follows from the condition (2.9) in Theorem 2 that if

$$\det \left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial(\mathbf{x}, \lambda)} \right] \neq 0$$

at $(\tilde{\mathbf{p}}, \tilde{\mathbf{x}}, \tilde{\lambda})$, then there exists a function $f : \mathrm{I\!R}^3 \to \mathrm{I\!R}^4$ such that

$$f(\mathbf{p}) = (\mathbf{x}, \lambda) \tag{2.15}$$

and

$$F(\mathbf{p}, f(\mathbf{p})) = \mathbf{0}$$

for all $\mathbf{p}$ sufficiently close to $\tilde{\mathbf{p}}$. That is, f solves the reflection problem in a neighborhood of the known reflection path. More importantly, from Corollary 1, we can solve for $\partial f/\partial \mathbf{p}$ in this neighborhood using equation (2.11), which yields

$$\left[\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}\right]_{4\times 3} = -\left[\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial(\mathbf{x},\lambda)}\right]^{-1}_{4\times 4}\left[\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial(\mathbf{p})}\right]_{4\times 3.} \tag{2.16}$$

Equation (2.15) shows that the path function $\Psi : \mathbf{p} \to \mathbf{x}$ is easily obtained from f by discarding its last component λ. By introducing an operator $\mathbf{sub} : \mathrm{Hom}(\mathbb{R}^3, \mathbb{R}^4) \to \mathrm{Hom}(\mathbb{R}^3, \mathbb{R}^3)$, which drops the last row of a 4×3 matrix, the 3×3 path Jacobian $\mathbf{J}$ can be expressed as

$$\boxed{\mathbf{J} = \left[\frac{\partial \Psi(\mathbf{p})}{\partial \mathbf{p}}\right]_{3\times 3} = \mathbf{sub}\left(-\left[\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial(\mathbf{x},\lambda)}\right]^{-1}_{4\times 4}\left[\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial \mathbf{p}}\right]_{4\times 3}\right),} \tag{2.17}$$

which characterizes the variation in $\mathbf{x}$ with respect to $\mathbf{p}$. Note that all quantities on the right of equation (2.17) can be obtained from the Fermat equation (2.14) and the implicit function g. For clarity, later we shall refer to f as the *Fermat function* and the 4×3 matrix on the left of equation (2.16) before dropping the last row as the *Fermat Jacobian*, and designate it by D.

2.3.2 N-Bounce Path

In this section we show how to compute the path Jacobians for the more general case of N bounces. Given a path from a varying point $\mathbf{p}$ to a fixed point $\mathbf{q}$ via N reflecting surfaces, we order the reflection points from $\mathbf{p}$ to $\mathbf{q}$ as $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with $\mathbf{x}_0 = \mathbf{p}$ and $\mathbf{x}_{N+1} = \mathbf{q}$. The corresponding reflecting surfaces $\mathbf{G_i}$ and their implicit functions g_i are ordered accordingly, as shown in Figure 2.2 for a three-bounce path. Each reflection

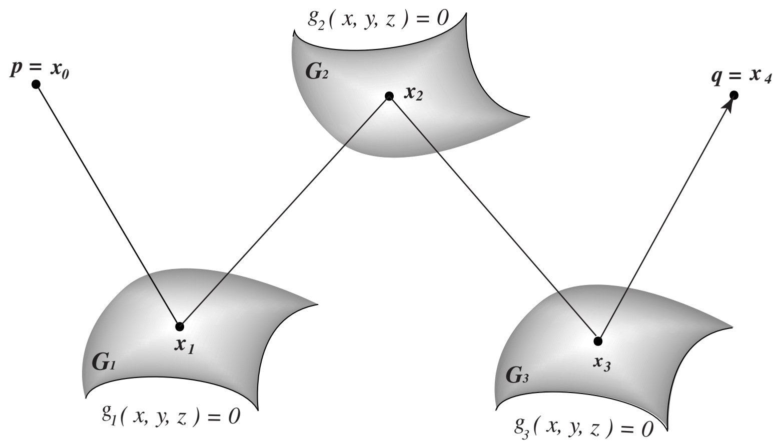


Figure 2.2: *A reflection path from* $\mathbf{p}$ *to* $\mathbf{q}$ *via three reflection points* $\mathbf{x}_1$*,* $\mathbf{x}_2$ *and* $\mathbf{x}_3$ *on three implicitly-defined surfaces* $\mathbf{G_1}$*,* $\mathbf{G_2}$ *and* $\mathbf{G_3}$*.*

point $\mathbf{x}_i$ is related to the varying endpoint $\mathbf{p}$ by a path function Ψ_i. The path Jacobian $\mathbf{J}_i$ at this point is defined as the derivative of Ψ_i with respect to $\mathbf{p}$.

As a ray path between $\mathbf{p}$ and $\mathbf{q}$, Fermat's principle states that the N reflection points are located in such a way that the optical length of the path $\mathbf{x}_0 - \mathbf{x}_1 - \cdots - \mathbf{x}_{N+1}$ is minimized or maximized. By analogy with the one-bounce path, we can apply the method of Lagrange Multipliers to the entire path to obtain a Fermat equation satisfied by all the $\mathbf{x}_i's$; the Implicit Function Theorem can then be used to compute the N path Jacobians. Consider a constraint vector $\mathbf{g} = [g_1, \ldots, g_N]^{\mathrm{T}}$ formed by the N implicit functions. We can apply the vector form (2.7) of the Lagrange condition to obtain a system of $4N$ equations satisfied by the N reflection points [59]:

$$\begin{aligned} \nabla_i d(\mathbf{x}_0, \ldots, \mathbf{x}_{N+1}) + \lambda_i \nabla_i g_i(\mathbf{x}_i) &= 0, \\ g_i(\mathbf{x}_i) &= 0, \end{aligned} \tag{2.18}$$

where $i = 1, \ldots, N$, and the gradient operator ∇_i is with respect to $\mathbf{x}_i$. Since each $\nabla_i d$

contains three points, equation (2.18) is equivalent to:

$$\begin{aligned} F_{g_1}(\mathbf{p}, \mathbf{x}_1, \mathbf{x}_2, \lambda_1) &= 0 \\ &\vdots \\ F_{g_N}(\mathbf{x}_{N-1}, \mathbf{x}_N, \lambda_N) &= 0 \end{aligned} \tag{2.19}$$

where F_{g_i} is the Fermat equation for each one-bounce path segment $(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$, as in equation (2.14). We now consider equation (2.19) as an implicit function $F : \mathbb{R}^3 \times \mathbb{R}^{4N} \rightarrow \mathbb{R}^{4N}$, with $\mathbf{p}$ as its independent variable:

$$F(\widehat{\mathbf{p}}, \mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N) = 0. \tag{2.20}$$

The Implicit Function Theorem and its corollary can then be applied to equation (2.20), provided that

$$\det\left(\frac{\partial F(\mathbf{p}, \mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N)}{\partial(\mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N)}\right) \neq 0$$

at the given path. This condition amounts to ensuring that the reflecting path does not include any points that are exactly on a boundary, and the rays of the path are nowhere tangent to a surface. In this case, there exists a function

$$f : \mathbb{R}^3 \rightarrow \mathbb{R}^4 \times \cdots \times \mathbb{R}^4, \tag{2.21}$$

that maps $\mathbf{p}$ to N pairs of reflection points and Lagrange multipliers, $(\mathbf{x}_i, \lambda_i)$, which holds for all points $\mathbf{p}'$ within a small neighborhood of $\mathbf{p}$. From Corollary 1, the derivative of this function is given by

$$\left[\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}\right]_{4N\times 3} = -\left[\frac{\partial F(\mathbf{p}, \mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N)}{\partial(\mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N)}\right]^{-1}_{4N\times 4N} \left[\frac{\partial F(\mathbf{p}, \mathbf{x}_1, \lambda_1, \ldots, \mathbf{x}_N, \lambda_N)}{\partial \mathbf{p}}\right]_{4N\times 3.} \tag{2.22}$$

According to equation (2.21), the N path Jacobians $\mathbf{J}_1, \mathbf{J}_2, \ldots, \mathbf{J}_N$ can be formed from the $4N \times 3$ matrix on the left by dropping the rows relating to the Lagrange multiplier vector.

In order to compute path Jacobians using equation (2.22), we must invert a $4N \times 4N$ matrix, which grows with the number of bounces. This computation makes the direct approach impractical. Instead, by taking each bounce separately, we approach the problem of computing path Jacobians for an N-bounce path by recursively applying formula (2.17) for a one-bounce path. This significantly reduces the computation for long reflecting paths.

We introduce some notation before deriving the recurrence relation for an N-bounce path. Instead of viewing each reflection point $\mathbf{x}_i$ as a path function Ψ_i of the common endpoint $\mathbf{p}$, we can also consider it as a function of the previous point $\mathbf{x}_{i-1}$ since each $\mathbf{x}_i$ is actually a one-bounce reflection with respect to the surrounding points $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$. We refer to this function as *reflection function*. Correspondingly, we introduce another 3×3 Jacobian matrix to denote the derivative of the ith reflection function with respect to the previous point $\mathbf{x}_{i-1}$ and call it the *one-bounce reflection Jacobian*. The reflection Jacobian characterizes how a reflection point $\mathbf{x}_i$ varies with respect to perturbations in its previous point $\mathbf{x}_{i-1}$. To distinguish the two Jacobians, we designate the path Jacobian as $\mathbf{J}^*$ and the one-bounce reflection Jacobian as $\mathbf{J}$ in a context of multiple reflections. Given the one-bounce reflection Jacobians for an N-bounce path, the corresponding path Jacobians can be easily computed via the recurrence

$$\begin{aligned} \mathbf{J}_1^* &= \mathbf{J}_1 \\ \mathbf{J}_i^* &= \mathbf{J}_i \cdot \mathbf{J}_{i-1}^* \end{aligned} \tag{2.23}$$

for $i = 2, \ldots, N$. The first identity follows from the fact that $\mathbf{x}_0 = \mathbf{p}$, while the recursive formula follows from the application of the chain rule:

$$\mathbf{J}_i^* = \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}} = \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \cdot \frac{\partial \mathbf{x}_{i-1}}{\partial \mathbf{p}}.$$

Next we show that the single-bounce path Jacobian (2.17) can be extended to a general multiple-bounce specular path by means of a recurrence relation expressing the reflection Jacobian $\mathbf{J}_i$ for each reflection point $\mathbf{x}_i$ in terms of the reflection Jacobian $\mathbf{J}_{i+1}$.

We start from the last bounce $(\mathbf{x}_{N-1}, \mathbf{x}_N, \mathbf{q})$, where $\mathbf{q}$ is the fixed endpoint. This case reduces to a simple one-bounce path, as discussed in section 2.3.1,

$$F_{g_N}(\widehat{\mathbf{x}}_{N-1}, \mathbf{x}_N, \lambda_N) = \mathbf{0}. \tag{2.24}$$

The Implicit Function Theorem implies that if

$$\det\left(\frac{\partial F_{g_N}(\mathbf{x}_{N-1}, \mathbf{x}_N, \lambda_N)}{\partial(\mathbf{x}_N, \lambda_N)}\right) \neq 0,$$

there exists a function $f_N : \mathbf{x}_{N-1} \rightarrow (\mathbf{x}_N, \lambda_N)$. The function f_N can be decomposed into two components: $f_{N1} : \mathrm{I\!R}^3 \rightarrow \mathrm{I\!R}^3$ and $f_{N2} : \mathrm{I\!R}^3 \rightarrow R$, mapping to $\mathbf{x}_N$ and λ_N respectively, where f_{N1} is the reflection function defined above. With the existence of f_N, the last reflection Jacobian $\mathbf{J}_N$ has been derived in equation (2.17). As a segment of an N-bounce path, the ith bounce still observes Fermat's principle. Applying Lagrange Multipliers to $(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1})$, where $i \neq N$, we obtain a similar Fermat equation:

$$F_{g_i}(\widehat{\mathbf{x}}_{i-1}, \mathbf{x}_i, \widehat{\mathbf{x}}_{i+1}, \lambda_i) = \mathbf{0}, \tag{2.25}$$

which differs from equation (2.13) by taking two explicit independent variables into account; one for each end point. By processing the N-bounce path *from back to front*, we construct a function $f_{i+1} : \mathrm{I\!R}^3 \rightarrow \mathrm{I\!R}^4$ mapping $\mathbf{x}_i$ to $(\mathbf{x}_{i+1}, \lambda_{i+1})$. By decomposing f_{i+1} into $f_{(i+1)1} : \mathbf{x}_i \rightarrow \mathbf{x}_{i+1}$ and $f_{(i+1)2} : \mathbf{x}_i \rightarrow \lambda_{i+1}$, we can express $\mathbf{x}_{i+1}$ implicitly in equation (2.25) in terms of $f_{(i+1)1}$. That is,

$$F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, f_{(i+1)1}(\mathbf{x}_i), \lambda_i) = \mathbf{0}. \tag{2.26}$$

Equation (2.26) can be treated as an implicit equation of three variables $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, and λ_i, with $\mathbf{x}_{i-1}$ as the independent variable. That is,

$$H_i(\widehat{\mathbf{x}}_{i-1}, \mathbf{x}_i, \lambda_i) = \mathbf{0} \tag{2.27}$$

where

$$H_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \lambda_i) = F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, f_{(i+1)1}(\mathbf{x}_i), \lambda_i).$$

Applying the Implicit Function Theorem to equation (2.27), we can show that the ith explicit function $f_i : \mathbf{x}_{i-1} \to (\mathbf{x}_i, \lambda_i)$ exists under the condition that

$$\det\left(\frac{\partial H_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \lambda_i)}{\partial(\mathbf{x}_i, \lambda_i)}\right) = \det\left(\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, f_{(i+1)1}(\mathbf{x}_i), \lambda_i)}{\partial(\mathbf{x}_i, \lambda_i)}\right) \neq 0. \tag{2.28}$$

In terms of f_i, equation (2.26) can be reformulated as

$$F_{g_i}(\mathbf{x}_{i-1},\ f_{i1}(\mathbf{x}_{i-1}),\ f_{(i+1)1}(f_{i1}(\mathbf{x}_{i-1})),\ f_{i2}(\mathbf{x}_{i-1})) = \mathbf{0}. \tag{2.29}$$

Differentiating both sides of equation (2.29) with respect to $\mathbf{x}_{i-1}$, we obtain

$$\frac{\partial F_{g_i}}{\partial \mathbf{x}_{i-1}} + \frac{\partial F_{g_i}}{\partial \mathbf{x}_i} \cdot \frac{\partial f_{i1}}{\partial \mathbf{x}_{i-1}} + \frac{\partial F_{g_i}}{\partial \mathbf{x}_{i+1}} \cdot \frac{\partial f_{(i+1)1}}{\partial \mathbf{x}_i} \cdot \frac{\partial f_{i1}}{\partial \mathbf{x}_{i-1}} + \frac{\partial F_{g_i}}{\partial \lambda_i} \cdot \frac{\partial f_{i2}}{\partial \mathbf{x}_{i-1}} = \mathbf{0}. \tag{2.30}$$

Rearranging the terms in equation (2.30) and substituting $\mathbf{J}_{i+1}$ computed previously for $\dfrac{\partial f_{(i+1)1}}{\partial \mathbf{x}_i}$, we obtain

$$\frac{\partial F_{g_i}}{\partial \mathbf{x}_{i-1}} = -\begin{bmatrix} \dfrac{\partial F_{g_i}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} + \dfrac{\partial F_{g_i}}{\partial \mathbf{x}_i} & \dfrac{\partial F_{g_i}}{\partial \lambda_i} \end{bmatrix} \begin{bmatrix} \dfrac{\partial f_{i1}(\mathbf{x}_{i-1})}{\partial \mathbf{x}_{i-1}} \\ \\ \dfrac{\partial f_{i2}(\mathbf{x}_{i-1})}{\partial \mathbf{x}_{i-1}} \end{bmatrix}.$$

By inverting the matrix, we solve for $\partial f_i / \partial \mathbf{x}_{i-1}$:

$$\left[\frac{\partial f_i}{\partial \mathbf{x}_{i-1}}\right]_{4\times 3} = -\begin{bmatrix} \dfrac{\partial F_{g_i}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} + \dfrac{\partial F_{g_i}}{\partial \mathbf{x}_i} & \dfrac{\partial F_{g_i}}{\partial \lambda_i} \end{bmatrix}^{-1} \left[\frac{\partial F_{g_i}}{\partial \mathbf{x}_{i-1}}\right], \tag{2.31}$$

where the existence of the inverse is guaranteed by the condition in (2.28). Introducing an operator $\mathbf{aug} : \mathrm{Hom}(\mathbb{R}^3, \mathbb{R}^4) \to \mathrm{Hom}(\mathbb{R}^4, \mathbb{R}^4)$, which expands a 4×3 matrix with a zero column, we can rewrite equation (2.31) as

$$\left[\frac{\partial f_i}{\partial \mathbf{x}_{i-1}}\right]_{4\times 3} = -\left[\frac{\partial F_{g_i}}{\partial(\mathbf{x}_i, \lambda_i)} + \mathbf{aug}\left(\frac{\partial F_{g_i}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1}\right)\right]^{-1} \left[\frac{\partial F_{g_i}}{\partial \mathbf{x}_{i-1}}\right]. \qquad (2.32)$$

Finally, taking the submatrix of the left hand side of equation (2.32), we have a recurrence relation for the ith reflection Jacobian $\mathbf{J}_i$:

$$\boxed{\mathbf{J}_i \;=\; \mathbf{sub}\left(-\left[A_i + \mathbf{aug}(B_i \cdot \mathbf{J}_{i+1})\right]^{-1} T_i\right),} \qquad (2.33)$$

where $i = N-1, \ldots, 1$ and

$$\begin{aligned} A_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial(\mathbf{x}_i, \lambda_i)}\right]_{4\times 4} \\ B_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial \mathbf{x}_{i+1}}\right]_{4\times 3} \qquad (2.34) \\ T_i &= \left[\frac{\partial F_{g_i}(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i)}{\partial \mathbf{x}_{i-1}}\right]_{4\times 3}. \end{aligned}$$

Thus, the reflection Jacobians $\mathbf{J}_i$ are computed *backward*, from the fixed point towards the perturbed point. The starting Jacobian $\mathbf{J}_N$ for the last reflection point $\mathbf{x}_N$, which is calculated first, can be viewed as a special case of this recurrence relation where $B_N = 0$ and $\mathbf{J}_{N+1} = \mathbf{0}$.

By expressing the perturbation formula for each reflection point $\mathbf{x}_i$ as a Taylor series expanded about the previous point instead of the varying endpoint $\mathbf{p}$, we can directly use reflection Jacobians to perturb a given N-bounce path to the first order. That is, starting from the first reflection Jacobian $\mathbf{J}_1$ (equivalent to the first path Jacobian $\mathbf{J}_1^*$), all the new reflection points can be approximated to the first order successively, from the perturbed point $\mathbf{p}$ towards the fixed point $\mathbf{q}$:

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta \mathbf{x}_{i-1} \tag{2.35}$$

where $i = 1, 2, 3, \ldots, N$, $\Delta\mathbf{x}_{i-1} = \mathbf{x}'_{i-1} - \mathbf{x}_{i-1}$, and $\Delta\mathbf{x}_0 = \Delta\mathbf{p}$. Therefore, it is not necessary to compute the complete path Jacobians $\mathbf{J}^*$'s from reflection Jacobians $\mathbf{J}$'s using (2.23).

Compared to the direct method, the recursive method suggests an obvious recurrence relation, is much more efficient and easier to implement, as it involves only 4×4 matrices. Also, it can be verified that these two methods yield the same answer for the first path Jacobian $\mathbf{J}_1^*$.

2.3.3 Equivalence of Multiple Bounce Methods

Consider a three-bounce path shown in Figure 2.2, in this appendix, we will compute the first path Jacobian $\mathbf{J}_1^*$ for the reflection point $\mathbf{x}_1$ with both recursive method and direct method discussed in Section 2.3.2, and verify that two approaches are equivalent.

Recursive Method

Computing three reflection Jacobians ($\mathbf{J}_3$, $\mathbf{J}_2$, $\mathbf{J}_1$), or equivalently, three Fermat Jacobians (D_3, D_2, D_1), from back to forth according to the recursive formula (2.33), we will get:

$$D_3 = -\left[\frac{\partial F_{g_3}}{\partial(\mathbf{x}_3, \lambda_3)}\right]^{-1}\left[\frac{\partial F_{g_3}}{\partial \mathbf{x}_2}\right] \tag{2.36}$$

$$D_2 = -\left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} + \mathbf{aug}\left(\frac{\partial F_{g_2}}{\partial \mathbf{x}_3}\cdot \mathbf{J}_3\right)\right]^{-1}\left[\frac{\partial F_{g_2}}{\partial \mathbf{x}_1}\right] \tag{2.37}$$

$$D_1 = -\left[\frac{\partial F_{g_1}}{\partial(\mathbf{x}_1, \lambda_1)} + \mathbf{aug}\left(\frac{\partial F_{g_1}}{\partial \mathbf{x}_2}\cdot \mathbf{J}_2\right)\right]^{-1}\left[\frac{\partial F_{g_1}}{\partial \mathbf{p}}\right] \tag{2.38}$$

Since λ_3 doesn't appear in the expression for F_{g_2}, we have $\dfrac{\partial F_{g_2}}{\partial \lambda_3} = 0$. Similarly, $\dfrac{\partial F_{g_1}}{\partial \lambda_2} = 0$. Let

$$\begin{aligned} M_2 &= \begin{bmatrix} D_3 & \mathbf{0} \end{bmatrix} \\ M_1 &= \begin{bmatrix} D_2 & \mathbf{0} \end{bmatrix} \end{aligned}, \tag{2.39}$$

we can replace the **aug** operator in equation (2.37) and (2.38) by:

$$D_2 = -\left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} + \frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \cdot M_2\right]^{-1} \left[\frac{\partial F_{g_2}}{\partial \mathbf{x}_1}\right] \tag{2.40}$$

$$D_1 = -\left[\frac{\partial F_{g_1}}{\partial(\mathbf{x}_1, \lambda_1)} + \frac{\partial F_{g_1}}{\partial(\mathbf{x}_2, \lambda_2)} \cdot M_1\right]^{-1} \left[\frac{\partial F_{g_1}}{\partial \mathbf{p}}\right] \tag{2.41}$$

By modifying the expressions for D_3 and D_2 shown in equation (2.36) and (2.40) with the identities $\dfrac{\partial F_{g_3}}{\partial \lambda_2} = 0$ and $\dfrac{\partial F_{g_2}}{\partial \lambda_1} = 0$, we can express M_2 and M_1 defined in equation (2.39) as:

$$M_2 = -\left[\frac{\partial F_{g_3}}{\partial(\mathbf{x}_3, \lambda_3)}\right]^{-1} \left[\frac{\partial F_{g_3}}{\partial(\mathbf{x}_2, \lambda_2)}\right] \tag{2.42}$$

$$M_1 = -\left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} + \frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \cdot M_2\right]^{-1} \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_1, \lambda_1)}\right] \tag{2.43}$$

By expanding equation (2.43) with equation (2.42) and then substituting the result into equation (2.41), we get

$$D_1 = -M^{-1} \left[\frac{\partial F_{g_1}}{\partial \mathbf{p}}\right], \tag{2.44}$$

where

$$M = \left[\frac{\partial F_{g_1}}{\partial(\mathbf{x}_1, \lambda_1)} - \frac{\partial F_{g_1}}{\partial(\mathbf{x}_2, \lambda_2)} \left[\frac{\partial F_{g_1}}{\partial(\mathbf{x}_2, \lambda_2)} - \frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \left[\frac{\partial F_{g_3}}{\partial(\mathbf{x}_3, \lambda_3)}\right]^{-1} \frac{\partial F_{g_3}}{\partial(\mathbf{x}_2, \lambda_2)}\right]^{-1} \frac{\partial F_{g_2}}{\partial(\mathbf{x}_1, \lambda_1)}\right] \tag{2.45}$$

The first path Jacobian $\mathbf{J}_1^*$ (equivalent to $\mathbf{J}_1$) is formed from the first three rows of D_1.

Direct Method

For a three-bounce path, the direct method discussed in Section 2.3.2 attains a system of 12 implicit equations, $F(\widehat{\mathbf{p}}, \mathbf{x}_1, \lambda_1, \mathbf{x}_2, \lambda_2, \mathbf{x}_3, \lambda_3) = 0$, which can be written as

$$\begin{aligned} F_{g_1}(\widehat{\mathbf{p}}, \mathbf{x}_1, \mathbf{x}_2, \lambda_1) &= \mathbf{0} \\ F_{g_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \lambda_2) &= \mathbf{0} \;. \\ F_{g_3}(\mathbf{x}_2, \mathbf{x}_3, \mathbf{q}, \lambda_3) &= \mathbf{0} \end{aligned}$$

As derived in equation (2.22), the Jacobian matrix for the function $f : \mathbf{p} \rightarrow (\mathbf{x}_1, \lambda_1, \mathbf{x}_2, \lambda_2, \mathbf{x}_3, \lambda_3)$ can be computed as:

$$\left[\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}\right]_{12\times 3} = -\left[\frac{\partial F(\mathbf{p}, \mathbf{x}_1, \lambda_1, \mathbf{x}_2, \lambda_2, \mathbf{x}_3, \lambda_3)}{\partial(\mathbf{x}_1, \lambda_1, \mathbf{x}_2, \lambda_2, \mathbf{x}_3, \lambda_3)}\right]^{-1}_{12\times 12} \left[\frac{\partial F(\mathbf{p}, \mathbf{x}_1, \lambda_1, \mathbf{x}_2, \lambda_2, \mathbf{x}_3, \lambda_3)}{\partial \mathbf{p}}\right]_{12\times 3.} \quad (2.46)$$

If we pair $\mathbf{x}_i$ and λ_i together, and define the components of high dimensional function f as

$$f(\mathbf{p}) = \begin{bmatrix} f_1(\mathbf{p}) \\ f_2(\mathbf{p}) \\ f_3(\mathbf{p}) \end{bmatrix} = \begin{bmatrix} (\mathbf{x}_1, \lambda_1) \\ (\mathbf{x}_2, \lambda_2) \\ (\mathbf{x}_3, \lambda_3) \end{bmatrix},$$

equation (2.46) can be reformulated in a block matrix form

$$\begin{bmatrix} \frac{\partial F_{g_1}(\mathbf{p}, \mathbf{x}_1, \mathbf{x}_2, \lambda_1)}{\partial(\mathbf{x}_1, \lambda_1)} & \frac{\partial F_{g_1}(\mathbf{p}, \mathbf{x}_1, \mathbf{x}_2, \lambda_1)}{\partial(\mathbf{x}_2, \lambda_2)} & 0 \\ \frac{\partial F_{g_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \lambda_2)}{\partial(\mathbf{x}_1, \lambda_1)} & \frac{\partial F_{g_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \lambda_2)}{\partial(\mathbf{x}_2, \lambda_2)} & \frac{\partial F_{g_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \lambda_2)}{\partial(\mathbf{x}_3, \lambda_3)} \\ 0 & \frac{\partial F_{g_3}(\mathbf{x}_2, \mathbf{x}_3, \mathbf{q}, \lambda_3)}{\partial(\mathbf{x}_2, \lambda_2)} & \frac{\partial F_{g_3}(\mathbf{x}_2, \mathbf{x}_3, \mathbf{q}, \lambda_3)}{\partial(\mathbf{x}_3, \lambda_3)} \end{bmatrix} \begin{bmatrix} \frac{\partial f_1(\mathbf{p})}{\partial \mathbf{p}} \\ \frac{\partial f_2(\mathbf{p})}{\partial \mathbf{p}} \\ \frac{\partial f_3(\mathbf{p})}{\partial \mathbf{p}} \end{bmatrix} .$$

$$= - \begin{bmatrix} \dfrac{\partial F_{g_1}(\mathbf{p}, \mathbf{x}_1, \mathbf{x}_2, \lambda_1)}{\partial \mathbf{p}} \\ \\ 0 \\ \\ 0 \end{bmatrix} . \quad (2.47)$$

Note that the blocks are 4×4 Jacobian matrices. Let

$$\begin{aligned} y_1 &= \frac{\partial f_1(\mathbf{p})}{\partial \mathbf{p}} \\ y_2 &= \frac{\partial f_2(\mathbf{p})}{\partial \mathbf{p}} \\ y_3 &= \frac{\partial f_3(\mathbf{p})}{\partial \mathbf{p}}, \end{aligned}$$

Equation (2.47) can be considered as a linear system of three unknowns y_1, y_2 and y_3:

$$\begin{cases} \dfrac{\partial F_{g_1}}{\partial(\mathbf{x}_1, \lambda_1)} y_1 + \dfrac{\partial F_{g_1}}{\partial(\mathbf{x}_2, \lambda_2)} y_2 = \dfrac{\partial F_{g_1}}{\partial \mathbf{p}} & (1) \\ \dfrac{\partial F_{g_2}}{\partial(\mathbf{x}_1, \lambda_1)} y_1 + \dfrac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} y_2 + \dfrac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} y_3 = 0 & (2) \\ \dfrac{\partial F_{g_3}}{\partial(\mathbf{x}_2, \lambda_2)} y_2 + \dfrac{\partial F_{g_3}}{\partial(\mathbf{x}_3, \lambda_3)} y_3 = 0 & (3) \end{cases}$$

Using substitution, we can solve for y_3 in terms of y_2 from (3):

$$y_3 = - \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \right]^{-1} \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} \right] y_2 \quad (2.48)$$

Plugging equation (2.48) into (2) to solve for y_2:

$$y_2 = - \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} - \frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_3, \lambda_3)} \right]^{-1} \frac{\partial F_{g_2}}{\partial(\mathbf{x}_2, \lambda_2)} \right]^{-1} \left[\frac{\partial F_{g_2}}{\partial(\mathbf{x}_1, \lambda_1)} \right] y_1 \quad (2.49)$$

Finally, substitute equation (2.49) into (1) and solve for y_1:

$$y_1 = -M^{-1} \left[\frac{\partial F_{g1}}{\partial \mathbf{p}} \right]$$

where

$$M = \left[\frac{\partial F_{g1}}{\partial(\mathbf{x}_1, \lambda_1)} - \frac{\partial F_{g1}}{\partial(\mathbf{x}_2, \lambda_2)} \left[\frac{\partial F_{g1}}{\partial(\mathbf{x}_2, \lambda_2)} - \frac{\partial F_{g2}}{\partial(\mathbf{x}_3, \lambda_3)} \left[\frac{\partial F_{g3}}{\partial(\mathbf{x}_3, \lambda_3)} \right]^{-1} \frac{\partial F_{g3}}{\partial(\mathbf{x}_2, \lambda_2)} \right]^{-1} \frac{\partial F_{g2}}{\partial(\mathbf{x}_1, \lambda_1)} \right] \tag{2.50}$$

Notice that $f_1(\mathbf{p}) = (\mathbf{x}_1, \lambda_1)$ and $y_1 = \dfrac{\partial f_1(\mathbf{p})}{\partial \mathbf{p}}$, the first path Jacobian $\mathbf{J}_1^*$ is also formed from the first three rows of y_1. Comparing equation (2.50) and equation (2.45), we verify that both approaches produce the same results for the first path Jacobian $\mathbf{J}_1^*$. In addition, by comparing equations (2.48), (2.49) and (2.50) with (2.36), (2.40) and (2.45), it can be easily verified that the conditions for the existence of the above inverse matrices used in solving the linear system are exactly those non-singularity conditions implied while applying the Implicit Function Theorem to each bounce in the recursive procedure.

2.4 The Path Hessian

As a linear approximation of the path function $\Psi : \mathbb{R}^3 \to \mathbb{R}^3$, the path Jacobian matrix $\mathbf{J}$ maps from tangent space to tangent space, which is equivalent to locally fitting the function Ψ with a linear form. To obtain better accuracy, we may compute one more term in equation (2.4), that is, the path Hessian $\mathbf{H}$, and fit the function Ψ locally with a quadratic form. As a second derivative of the path function, the path Hessian computation involves matrix differentiation and yields a tensor of the third order, which we can represent as a collection of three 3×3 matrices, as shown in equation (2.3).

2.4.1 One-Bounce Path

Let $f : \mathbf{p} \to (\mathbf{x}, \lambda)$ denote the function implicitly defined by the Fermat equation (2.13) in the one-bounce case, then the path function $\Psi = (\Psi_1, \Psi_2, \Psi_3)$ is related to f by

$$\Psi_i = f_i \tag{2.51}$$

for $i = 1, 2, 3$. According to equation (2.3), all the second partial derivatives $f_{i,jk}$ $(i, j, k = 1, 2, 3)$ must be evaluated for the path Hessian $\mathbf{H}$. Observe that the first-order approximation in equation (2.17) yields:

$$T(\mathbf{p}, \mathbf{x}, \lambda) = -\Gamma(\mathbf{p}, \mathbf{x}, \lambda) D(\mathbf{p}), \tag{2.52}$$

where T, Γ and D are given by

$$\begin{aligned} T(\mathbf{p}, \mathbf{x}, \lambda) &= \left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial \mathbf{p}}\right]_{4\times 3} \\ \Gamma(\mathbf{p}, \mathbf{x}, \lambda) &= \left[\frac{\partial F(\mathbf{p}, \mathbf{x}, \lambda)}{\partial (\mathbf{x}, \lambda)}\right]_{4\times 4} \\ D(\mathbf{p}) &= \left[\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}\right]_{4\times 3.} \end{aligned}$$

To compute the derivatives of matrices T, Γ and D in equation (2.52), it is convenient to write them in Cartesian tensor form. Thus, equation (2.52) becomes

$$T_{ij}(\mathbf{p}, \mathbf{x}, \lambda) = -\sum_{k=1}^{4} \Gamma_{ik}(\mathbf{p}, \mathbf{x}, \lambda) D_{kj}(\mathbf{p}),$$

where $i = 1, 2, 3, 4$ and $j = 1, 2, 3$. Replacing $(\mathbf{x}, \lambda)$ with the function f, we obtain

$$T_{ij}(\mathbf{p}, f(\mathbf{p})) = -\sum_{k=1}^{4} \Gamma_{ik}(\mathbf{p}, f(\mathbf{p})) D_{kj}(\mathbf{p}). \tag{2.53}$$

Differentiating both sides of equation (2.53) with respect to $\mathbf{p}$, we have

$$T_{ij,m} = -\sum_{k=1}^{4} (\Gamma_{ik,m} D_{kj}(\mathbf{p}) + \Gamma_{ik} D_{kj,m}(\mathbf{p})),$$

which can also be expressed as the gradient of these matrices [70, pp.62]. That is,

$$(\nabla T)_{mij} = -\sum_{k=1}^{4} ((\nabla \Gamma)_{mik} D_{kj}(\mathbf{p}) + \Gamma_{ik} (\nabla D)_{mkj}), \tag{2.54}$$

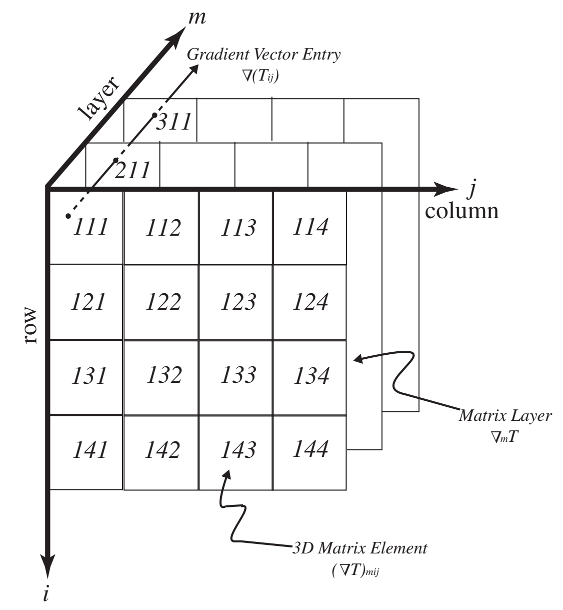


Figure 2.3: *The gradient of a matrix T, ∇T, is a third-order array with "layer", "row" and "column".*

where the gradient of matrices T, Γ and D are defined by

$$\begin{aligned}
(\nabla T)_{mij} &= T_{ij,m} = \frac{\partial T_{ij}(\mathbf{p}, f(\mathbf{p}))}{\partial p_m} \\
(\nabla \Gamma)_{mik} &= \Gamma_{ik,m} = \frac{\partial \Gamma_{ik}(\mathbf{p}, f(\mathbf{p}))}{\partial p_m} \qquad (2.55) \\
(\nabla D)_{mkj} &= D_{kj,m} = \frac{\partial D_{kj}(\mathbf{p})}{\partial p_m},
\end{aligned}$$

for $i, k = 1, 2, 3, 4,\ j, m = 1, 2, 3$.

As shown in Figure 2.3, the gradient of a matrix T is a third-order array; the three ordered subscripts can be interpreted as "layer", "row", "column", respectively. The row and column (the second and third indices) correspond to the row and column in the matrix T, while the layer (the first index) corresponds to each coordinate of the differential variable. As a convention, we always use m to index layer and i, j, k, l to index row or column. For example, the component $(\nabla T)_{mij}$ in equation (2.55) denotes the partial derivative of T_{ij} with respect to p_m (the mth coordinate component of $\mathbf{p}$).

On the one hand, each layer of ∇T constitutes a matrix $\{t_{ij}\}$ with $t_{ij} = (\nabla T)_{mij}$ for a fixed m, which we designate by $\nabla_m T$; on the other hand, ∇T itself can be considered as a special "matrix"[2] $\{s_{ij}\}$ with $s_{ij} = \nabla(T_{ij})$, that is, each "matrix" entry is a gradient vector of its corresponding entry in T.

In terms of the mth layers of ∇T, $\nabla\Gamma$ and ∇D, the summation on the right hand side of equation (2.54) can be interpreted as two matrix products, that is,

$$\nabla_m T = -\nabla_m \Gamma \cdot D - \Gamma \cdot \nabla_m D.$$

Solving for each layer of ∇D, we obtain

$$\boxed{\nabla_m D = -\Gamma^{-1}\left(\nabla_m T + \nabla_m \Gamma \cdot D\right),} \tag{2.56}$$

for $m = 1, 2, 3$.

As the Fermat Jacobian we defined in Section 2.3, D has a matrix form

$$D = \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \\ f_{4,1} & f_{4,2} & f_{4,3} \end{bmatrix}.$$

The mth layer of ∇D is obtained by differentiating each component of D with respect to the mth coordinate of $\mathbf{p}$,

$$\nabla_m D = \begin{bmatrix} f_{1,1m} & f_{1,2m} & f_{1,3m} \\ f_{2,1m} & f_{2,2m} & f_{2,3m} \\ f_{3,1m} & f_{3,2m} & f_{3,3m} \\ f_{4,1m} & f_{4,2m} & f_{4,3m} \end{bmatrix}.$$

[2] Strictly speaking, it is not a matrix since its entry is a gradient vector instead of a scalar.

Using equation (2.51), the gradient of the path Jacobian $\mathbf{J}$ can be formed from ∇D by dropping the last row for each layer,

$$\nabla_m \mathbf{J} = \mathbf{sub}\left(\nabla_m D\right) = \begin{bmatrix} f_{1,1m} & f_{1,2m} & f_{1,3m} \\ f_{2,1m} & f_{2,2m} & f_{2,3m} \\ f_{3,1m} & f_{3,2m} & f_{3,3m} \end{bmatrix}. \tag{2.57}$$

Alternatively, we may write each component of $\nabla \mathbf{J}$ as:

$$(\nabla \mathbf{J})_{mij} = f_{i,jm}, \tag{2.58}$$

where $i, j, m = 1, 2, 3$. From the definition of path Hessians in equation (2.3) and equation (2.51), we have

$$\mathbf{H}_{ijm} = f_{i,jm}, \tag{2.59}$$

where $i, j, m = 1, 2, 3$. Notice that we combine three Hessian matrices into a third order Hessian tensor. Combining equation (2.57), (2.58) and (2.59), we obtain the path Hessian $\mathbf{H}$ by reorganizing ∇D:

$$\boxed{\mathbf{H}_{ijm} = (\nabla D)_{mij},} \tag{2.60}$$

where $i, j, m = 1, 2, 3$.

Given a matrix gradient as a special matrix with vector entries, we compute ∇T and $\nabla \Gamma$ in equation (2.56) by evaluating the gradient of each matrix component:

$$\begin{aligned} \nabla(T_{ij}) &= \frac{\partial T_{ij}(\mathbf{p}, f(\mathbf{p}))}{\partial \mathbf{p}} \\ &= \frac{\partial T_{ij}(\mathbf{p}, \mathbf{x}, \lambda)}{\partial \mathbf{p}} + \frac{\partial T_{ij}(\mathbf{p}, \mathbf{x}, \lambda)}{\partial (\mathbf{x}, \lambda)} \cdot D \\ \nabla(\Gamma_{ik}) &= \frac{\partial \Gamma_{ik}(\mathbf{p}, f(\mathbf{p}))}{\partial \mathbf{p}} \end{aligned} \tag{2.61}$$

$$= \frac{\partial \Gamma_{ik}(\mathbf{p},\mathbf{x},\lambda)}{\partial \mathbf{p}} + \frac{\partial \Gamma_{ik}(\mathbf{p},\mathbf{x},\lambda)}{\partial(\mathbf{x},\lambda)} \cdot D, \tag{2.62}$$

which follows from the chain rule. In equations (2.61) and (2.62), the first term on the right hand side is a gradient vector and the second term is a product of a gradient row vector with a 4×3 Jacobian matrix; so each equation is a sum of two row vectors.

Expressing the matrix components T_{ij} and Γ_{ik} in terms of the Fermat equation F, we have

$$\begin{aligned} T_{ij}(\mathbf{p},\mathbf{x},\lambda) &= \left(\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial \mathbf{p}}\right)_{ij} = \frac{\partial F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial p_j} \\ \Gamma_{ik}(\mathbf{p},\mathbf{x},\lambda) &= \left(\frac{\partial F(\mathbf{p},\mathbf{x},\lambda)}{\partial(\mathbf{x},\lambda)}\right)_{ik} = \frac{\partial F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial x_k}, \end{aligned}$$

where $x_4 = \lambda$. Thus, equations (2.61) and (2.62) can be obtained by computing the second partial derivatives of the four explicit equations F_i ($i = 1, 2, 3, 4$) shown in (2.14) as:

$$\begin{aligned} \nabla(T_{ij}) &= \frac{\partial^2 F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial p_j \partial \mathbf{p}} + \frac{\partial^2 F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial p_j \partial(\mathbf{x},\lambda)} \cdot D \\ \nabla(\Gamma_{ik}) &= \frac{\partial^2 F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial x_k \partial \mathbf{p}} + \frac{\partial^2 F_i(\mathbf{p},\mathbf{x},\lambda)}{\partial x_k \partial(\mathbf{x},\lambda)} \cdot D \end{aligned} \tag{2.63}$$

where $i, k = 1, 2, 3, 4$ and $j = 1, 2, 3$.

2.4.2 N-Bounce Path

As in the first-order approximation, we introduce the *reflection Hessian* for the second derivative of the reflection function at each reflection point $\mathbf{x}_i$ with respect to the previous point $\mathbf{x}_{i-1}$. To be consistent with the notation used for reflection Jacobians and path Jacobians, we designate the reflection Hessian by $\mathbf{H}$ and the path Hessian by $\mathbf{H}^*$ for a multiple-bounce path. In this section, we show that the recurrence relation for reflection Jacobians $\mathbf{J}_i' s$ in an N-bounce path can be extended to the second order, yielding a corresponding recurrence relation for reflection Hessians $\mathbf{H}_i' s$.

While deriving path Jacobians for an N-bounce path, we have shown that there exists an implicitly-defined function $f_i : \mathbf{x}_{i-1} \rightarrow (\mathbf{x}_i, \lambda_i)$ associated with each reflection point $\mathbf{x}_i$ and it consists of two components: the reflection function $f_{i1} : \mathbf{x}_{i-1} \rightarrow \mathbf{x}_i$ and $f_{i2} : \mathbf{x}_{i-1} \rightarrow \lambda_i$. Analogous to (2.52), the first-order approximation for an N-bounce path in the recursive formula (2.33) yields:

$$T_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i) = -\Gamma_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \lambda_i) D_i(\mathbf{x}_{i-1}), \tag{2.64}$$

where T_i, Γ_i and D_i are defined as,

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

A_i and B_i are defined in equation (2.34). Since equation (2.64) has the same form as equation (2.52), following the same derivation will result in a similar formula for the gradient of Fermat Jacobian D_i:

$$\nabla_m D_i = -\Gamma_i^{-1} \left(\nabla_m T_i + \nabla_m \Gamma_i \cdot D_i\right), \tag{2.65}$$

for $m = 1, 2, 3$ and $i = 1, \ldots, N$. The gradient operator ∇ is with respect to the previous point $\mathbf{x}_{i-1}$. Furthermore, the relation between the function f_i and the reflection function f_{i1} suggests that equation (2.60) shown in the one-bounce case still holds for the conversion between the reflection Hessian $\mathbf{H}_i$ and the gradient of the corresponding Fermat Jacobian, ∇D_i. That is,

$$(\mathbf{H}_i)_{jkm} = (\nabla \mathbf{J}_i)_{mjk} = (\nabla D_i)_{mjk} \tag{2.66}$$

for $j, k, m = 1, 2, 3$.

Due to the existence of the functions $f_i : \mathbf{x}_{i-1} \to (\mathbf{x}_i, \lambda_i)$ and $f_{i+1} : \mathbf{x}_i \to (\mathbf{x}_{i+1}, \lambda_{i+1})$, the matrix variables T_i, Γ_i can be considered as functions of $\mathbf{x}_{i-1}$. That is,

$$(T_i)_{jk}(\mathbf{x}_{i-1},\ f_{i1}(\mathbf{x}_{i-1}),\ f_{(i+1)1}(f_{i1}(\mathbf{x}_{i-1})),\ f_{i2}(\mathbf{x}_{i-1})) \tag{2.67}$$

$$(\Gamma_i)_{jl}(\mathbf{x}_{i-1},\ f_{i1}(\mathbf{x}_{i-1}),\ f_{(i+1)1}(f_{i1}(\mathbf{x}_{i-1})),\ f_{i2}(\mathbf{x}_{i-1})), \tag{2.68}$$

for $j, l = 1, 2, 3, 4$ and $k = 1, 2, 3$. The gradient ∇T_i can be computed by differentiating each element $(T_i)_{jk}$ with respect to the independent variable $\mathbf{x}_{i-1}$,

$$\begin{aligned}
\nabla((T_i)_{jk}) &= \frac{\partial (T_i)_{jk}(\mathbf{x}_{i-1},\ f_{i1}(\mathbf{x}_{i-1}),\ f_{(i+1)1}(f_{i1}(\mathbf{x}_{i-1})),\ f_{i2}(\mathbf{x}_{i-1}))}{\partial \mathbf{x}_{i-1}} \\
&= \frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \frac{\partial f_{(i+1)1}}{\partial \mathbf{x}_i} \cdot \frac{\partial f_{i1}}{\partial \mathbf{x}_{i-1}} + \frac{\partial (T_i)_{jk}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot \frac{\partial f_i}{\partial \mathbf{x}_{i-1}} \\
&= \frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial (T_i)_{jk}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot D_i. \qquad (2.69)
\end{aligned}$$

Thus, each element of ∇T_i can be obtained from the component of the gradient vector on the left of equation (2.69):

$$(\nabla T_i)_{mjk} = \left(\frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial (T_i)_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial (T_i)_{jk}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot D_i \right)_m .$$

In computing $\nabla \Gamma_i$, we must consider the two cases:

$$(\Gamma_i)_{jl} = \begin{cases} (A_i)_{jl} + \sum_{k=1}^{3} (B_i)_{jk} (\mathbf{J}_{i+1})_{kl} & l = 1, 2, 3 \\ (A_i)_{jl} & l = 4 \end{cases} \tag{2.70}$$

When $l = 1, 2, 3$, using tensor differentiation notation, we differentiate the first equation in (2.70) with respect to $\mathbf{x}_{i-1}$:

$$(\Gamma_i)_{jl,m} = (A_i)_{jl,m} + \sum_{k=1}^{3} \left((B_i)_{jk,m} (\mathbf{J}_{i+1})_{kl} + (B_i)_{jk} (\mathbf{J}_{i+1})_{kl,m} \right).$$

Like equation (2.54), it can also be expressed with the gradient of matrices as:

$$(\nabla\Gamma_i)_{mjl} = (\nabla A_i)_{mjl} + \sum_{k=1}^{3}((\nabla B_i)_{mjk}(\mathbf{J}_{i+1})_{kl} + (B_i)_{jk}(\nabla C_i)_{mkl}). \tag{2.71}$$

Since the gradient operator ∇ above is with respect to $\mathbf{x}_{i-1}$, another symbol ∇C_i is introduced for the gradient of $\mathbf{J}_{i+1}$ with respect to $\mathbf{x}_{i-1}$, which is different from $\nabla\mathbf{J}_{i+1}$. Thus,

$$\nabla\left((\mathbf{J}_{i+1})_{kl}\right) = \frac{\partial(\mathbf{J}_{i+1})_{kl}}{\partial\mathbf{x}_i} \tag{2.72}$$

$$\nabla\left((C_i)_{kl}\right) = \frac{\partial(\mathbf{J}_{i+1})_{kl}}{\partial\mathbf{x}_{i-1}} = \nabla\left((\mathbf{J}_{i+1})_{kl}\right)\cdot\mathbf{J}_i. \tag{2.73}$$

In equation (2.72), we interpret $\mathbf{J}_{i+1} = \partial f_{(i+1)1}(\mathbf{x}_i)/\partial\mathbf{x}_i$ as a function of $\mathbf{x}_i$ and take its derivative with respect to $\mathbf{x}_i$. However, with $f_i : \mathbf{x}_{i-1} \to (\mathbf{x}_i, \lambda_i)$, we can also consider $\mathbf{J}_{i+1}$ as a function dependent on $\mathbf{x}_{i-1}$ and calculate its derivative with respect to $\mathbf{x}_{i-1}$ in equation (2.73). $\nabla\mathbf{J}_{i+1}$ and ∇C_i are related by the last identity of (2.73) from the chain rule. Writing equation (2.71) in terms of the mth layers of $\nabla\Gamma_i$, ∇A_i, ∇B_i and ∇C_i, we obtain a matrix form

$$\nabla_m\Gamma_i = \nabla_m A_i + \nabla_m B_i\cdot\mathbf{J}_{i+1} + B_i\cdot\nabla_m C_i \tag{2.74}$$

for $m = 1, 2, 3$. By considering A_i, B_i as functions of $\mathbf{x}_{i-1}$ like (2.67), we can compute ∇A_i, ∇B_i in the same way as ∇T_i. Thus,

$$\begin{aligned}
\nabla\left((A_i)_{jr}\right) &= \frac{\partial(A_i)_{jr}}{\partial\mathbf{x}_{i-1}} + \frac{\partial(A_i)_{jr}}{\partial\mathbf{x}_{i+1}}\cdot\mathbf{J}_{i+1}\cdot\mathbf{J}_i + \frac{\partial(A_i)_{jr}}{\partial(\mathbf{x}_i,\lambda_i)}\cdot D_i \\
\nabla\left((B_i)_{jk}\right) &= \frac{\partial(B_i)_{jk}}{\partial\mathbf{x}_{i-1}} + \frac{\partial(B_i)_{jk}}{\partial\mathbf{x}_{i+1}}\cdot\mathbf{J}_{i+1}\cdot\mathbf{J}_i + \frac{\partial(B_i)_{jk}}{\partial(\mathbf{x}_i,\lambda_i)}\cdot D_i \qquad (2.75) \\
\nabla\left((C_i)_{kl}\right) &= \nabla\left((\mathbf{J}_{i+1})_{kl}\right)\cdot\mathbf{J}_i.
\end{aligned}$$

where $j, r = 1, 2, 3, 4$ and $k, l = 1, 2, 3$. Combining equation (2.74) with the case of $l = 4$, we can write each component of $\nabla\Gamma_i$ as

$$(\nabla\Gamma_i)_{mjl} = \begin{cases} (\nabla_m A_i + \nabla_m B_i \cdot \mathbf{J}_{i+1} + B_i \cdot \nabla_m C_i)_{jl} & l = 1,2,3 \\ (\nabla A_i)_{mjl} & l = 4 \end{cases} \tag{2.76}$$

for $m = 1,2,3$ and $j = 1,2,3,4$. Note that all partial derivatives on the right hand side of equations (2.69) and (2.75) are available by evaluating the second partial derivatives of the Fermat equation F_{g_i}, just as we did in equation (2.63).

Similarly, the reflection Hessian $\mathbf{H}_i$ is obtained by reorganizing ∇D_i using equation (2.60). As seen from equation (2.75), the reflection Hessian at a reflection point $\mathbf{x}_i$ is not only dependent on its reflection Jacobian $\mathbf{J}_i$, but also dependent on the reflection Jacobian $\mathbf{J}_{i+1}$ and the reflection Hessian (implied in $\nabla\mathbf{J}_{i+1}$) computed for the following point $\mathbf{x}_{i+1}$. This dependence suggests that for the second-order approximation of a multiple-bounce path, both reflection Jacobians $\mathbf{J}$ and reflection Hessians $\mathbf{H}$ must be computed recursively *from back to front*, with the reflection Jacobian $\mathbf{J}_i$ computed before reflection Hessian $\mathbf{H}_i$ at each bounce.

With reflection Jacobians and reflection Hessians computed for an N-bounce path, a second-order perturbation formula similar to equation (2.35) can be expressed as:

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} + \frac{1}{2}(\Delta\mathbf{x}_{i-1})^{\mathrm{T}} \mathbf{H}_i \Delta\mathbf{x}_{i-1} \tag{2.77}$$

for $i = 1,2,3,\ldots,N$ and $\Delta\mathbf{x}_{i-1} = \mathbf{x}'_{i-1} - \mathbf{x}_{i-1}$, where $\Delta\mathbf{x}_0 = \Delta\mathbf{p}$. The tensor multiplication in equation (2.77) returns a vector

$$\begin{bmatrix} (\Delta\mathbf{p})^{\mathrm{T}}\mathbf{H}_{11}\Delta\mathbf{p} \\ (\Delta\mathbf{p})^{\mathrm{T}}\mathbf{H}_{12}\Delta\mathbf{p} \\ (\Delta\mathbf{p})^{\mathrm{T}}\mathbf{H}_{13}\Delta\mathbf{p} \end{bmatrix} \quad \textit{or} \quad \begin{bmatrix} (\Delta\mathbf{x}_{i-1})^{\mathrm{T}}\mathbf{H}_{i1}\Delta\mathbf{x}_{i-1} \\ (\Delta\mathbf{x}_{i-1})^{\mathrm{T}}\mathbf{H}_{i2}\Delta\mathbf{x}_{i-1} \\ (\Delta\mathbf{x}_{i-1})^{\mathrm{T}}\mathbf{H}_{i3}\Delta\mathbf{x}_{i-1} \end{bmatrix}$$

where $\mathbf{H}_{i1}$, $\mathbf{H}_{i2}$ and $\mathbf{H}_{i3}$ refer to three Hessian matrices of the path Hessian tensor $\mathbf{H}_i$.

2.5 Evaluation of Path Derivatives

In Section 2.3 and 2.4, we have derived 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 N reflection points $\mathbf{x}_i$ and the corresponding reflecting surfaces $g_i (i = 1, \ldots, N)$, the steps to computethe path Jacobian for each $\mathbf{x}_i$ can be summarized as follows:

```
PathJacobians( Point p, Point q, Point x[ ], Surface g[ ], int N )

1   Matrix J[ ]
2   $\mathbf{x}_0 \leftarrow \mathbf{p}$
    Compute the last path Jacobian
3   $\mathbf{J}_N \leftarrow$ OneBouncePathJacobian( $\mathbf{x}_{N-1}$, $\mathbf{x}_N$, $\mathbf{q}$, $\mathbf{0}$, $g_N$ )
4   Compute the other path Jacobians from back to front
5   for $i = N-1, \ldots, 1$
6       $\mathbf{J}_i \leftarrow$ OneBouncePathJacobian( $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $\mathbf{J}_{i+1}$, $g_i$ )
7   endfor
8   return J
```

<u>OneBouncePathJacobian(**Point** $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, **Matrix** $\mathbf{J}_{i+1}$, **Surface** g_i)</u>

```
1   **Matrix** $A, B, C, \Gamma, T, D_i, \mathbf{J}_i$
    *Compute lagrange multiplier* $\lambda_i$ *for the ray segment* $\mathbf{x}_{i-1} - \mathbf{x}_i - \mathbf{x}_{i+1}$
2   **Gradient** $h \leftarrow$ gradient of $g_i$ at $\mathbf{x}_i$
3   $h_j \leftarrow$ Nonzero component of $h$
4   **Scalar** $\lambda_i \leftarrow \left( \frac{((\mathbf{x}_{i-1})_j - (\mathbf{x}_i)_j)}{||\,\mathbf{x}_{i-1} - \mathbf{x}_i\,||} + \frac{((\mathbf{x}_{i+1})_j - (\mathbf{x}_i)_j)}{||\,\mathbf{x}_{i+1} - \mathbf{x}_i\,||} \right) \frac{1}{h_j}$

    *Compute the Jacobian matrices of Fermat equation with respect to different variables*
    *using equation (2.34), 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       $B \leftarrow [\partial F / \partial \mathbf{x}_{i+1}]$
9       $C \leftarrow B * \mathbf{J}_{i+1}$
10      *expand* $C$ *with a zero column*
11      $\Gamma \leftarrow A + C$
12  **else**
13      $\Gamma \leftarrow A$
14  **endif**
    *Apply recursive formula (2.33)*
15  $D_i \leftarrow \Gamma^{-1} T$
16  $\mathbf{J}_i \leftarrow$ the first three rows of $D_i$
17  **return** $\mathbf{J}_i$
```

The Jacobian matrices A, B and T of the Fermat equation F in line 5, 6 and 8 are computed by evaluating the first partial derivatives of the four explicit equations F_i shown in equation (2.14). Since they are based on the gradient of the implicit function, the gradient vector and the Hessian matrix of the corresponding implicit function g must be first derived in order to compute the path Jacobian. For any smooth function g, its mixed partial derivatives are order-independent, which reduces the computation cost of computing the required partials.

As shown in Section 2.4, the ith path Hessian is not only dependent on the ith path Jacobian, but also 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 and then 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. It calls a recursive routine OneBouncePathHessian to compute the path Hessian for each intermediate bounce. To evaluate the partial derivatives in line 7, 8 and 9 of OneBouncePathHessian, the second partial derivatives of the four explicit equations $F_1, \cdots, F_4$ in equation (2.14) are required, which requires the third 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 derivatives of the implicit function, or by using finite differences.

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

Matrix $\mathbf{J}[\,]$
Tensor $\mathbf{H}[\,]$
$\mathbf{x}_0 \leftarrow \mathbf{p}$
Compute the last path Jacobian and the last Hessian.
$\mathbf{J}_N \leftarrow$ OneBouncePathJacobian( $\mathbf{x}_{N-1}$, $\mathbf{x}_N$, $\mathbf{q}$, $\mathbf{0}$, $g_N$ )
$\mathbf{H}_N \leftarrow$ OneBouncePathHessian( $\mathbf{x}_{N-1}$, $\mathbf{x}_N$, $\mathbf{q}$, $g_N$, $\mathbf{J}_N$, $\mathbf{0}$, $\mathbf{0}$ )
Compute the other path Jacobians and path Hessians from back to front
for $i = N-1, \ldots, 1$
    $\mathbf{J}_i \leftarrow$ OneBouncePathJacobian( $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $\mathbf{J}_{i+1}$, $g_i$ )
    $\mathbf{H}_i \leftarrow$ OneBouncePathHessian( $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, $g_i$, $\mathbf{J}_i$, $\mathbf{J}_{i+1}$, $\mathbf{H}_{i+1}$ )
endfor
return $\mathbf{J}$, $\mathbf{H}$
```

OneBouncePathHessian(**Point** $\mathbf{x}_{i-1}$, $\mathbf{x}_i$, $\mathbf{x}_{i+1}$, **Surface** g_i, **Matrix** $\mathbf{J}_i$, $\mathbf{J}_{i+1}$, **Tensor** $\mathbf{H}_{i+1}$)

```
Matrix $A, B, T, D_i$ $\leftarrow$ obtained from OneBouncePathJacobian
Matrix Gradient $\nabla T$, $\nabla A$, $\nabla B$, $\nabla C$, $\nabla E$, $\nabla \Gamma$, $\nabla D_i$, $\nabla \mathbf{J}_{i+1}$
Tensor $\mathbf{H}_i$
$\nabla \mathbf{J}_{i+1}$ $\leftarrow$ reorganize $\mathbf{H}_{i+1}$ using equation (2.66)
Compute the gradient $\nabla T$, $\nabla A$, $\nabla B$, $\nabla C$ with equation (2.69) and (2.75).
for $j, r = 1, 2, 3, 4$
  for $k, l = 1, 2, 3$
    $\nabla(T_{jk}) \leftarrow \frac{\partial T_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial T_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial T_{jk}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot D_i$
    $\nabla(A_{jr}) \leftarrow \frac{\partial A_{jr}}{\partial \mathbf{x}_{i-1}} + \frac{\partial A_{jr}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial A_{jr}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot D_i$
    $\nabla(B_{jk}) \leftarrow \frac{\partial B_{jk}}{\partial \mathbf{x}_{i-1}} + \frac{\partial B_{jk}}{\partial \mathbf{x}_{i+1}} \cdot \mathbf{J}_{i+1} \cdot \mathbf{J}_i + \frac{\partial B_{jk}}{\partial (\mathbf{x}_i, \lambda_i)} \cdot D_i$
    $(\nabla C)_{kl} \leftarrow (\nabla \mathbf{J}_{i+1})_{kl} \cdot \mathbf{J}_i$
  endfor
endfor
Compute the gradient $\nabla \Gamma$ with equation (2.76).
for $m = 1, 2, 3$
  $\nabla_m E \leftarrow \nabla_m A + \nabla_m B \cdot \mathbf{J}_{i+1} + B \cdot \nabla_m C$
endfor
for $m = 1, 2, 3$
  for $j = 1, 2, 3, 4$
    for $l = 1, 2, 3$
      $(\nabla \Gamma)_{mjl} \leftarrow (\nabla_m E)_{jl}$
    endfor
    $(\nabla \Gamma)_{mj4} \leftarrow (\nabla A)_{mj4}$
  endfor
endfor
```

```
    *Compute the gradient $\nabla D_i$ with equation (2.65).*
24  **for** $m = 1, 2, 3$
25      $\nabla_m D_i \leftarrow -\Gamma^{-1} (\nabla_m T + \nabla_m \Gamma \cdot D_i)$
26  **endfor**
    *Compute path Hessian tensor from $\nabla D_i$ using equation (2.66).*
27  **for** $j, k, m = 1, 2, 3$
28      $(\mathbf{H}_i)_{jkm} \leftarrow (\nabla D_i)_{mjk}$
29  **endfor**
30  **return** $\mathbf{H}_i$
```

There is another property worthy to mention here. Although we have assumed in the derivation that **q** is fixed while **p** varies, the two endpoints of a reflection path are clearly symmetric. This property can be used to quickly update a path while moving an object or changing a view point.

Chapter 3

Interactive Specular Reflections

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 [59, 79]. In Chapter 2, we have derived a closed-form expression to analytically describe such coherent reflections from the perspective of path perturbation. In the next two chapters, we will demonstrate two applications to illustrate its use, both applications involves explicit reflection paths. One is aimed at rapidly approximating geometrically accurate specular reflections, the other is about simulating caustics, which will be covered in Chapter 4.

In this chapter, we describe a new approach for interactively approximating specular reflections in arbitrary curved surfaces. The technique is quite efficient as it employs local perturbations to interpolate point samples analytically and is applicable to any smooth implicitly-defined reflecting surface that is equipped with a ray intersection procedure. 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 based on the closed-form expressions for 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.

3.1 Introduction

Reflection is an very important view-dependent phenomenon which adds a new dimension to the realism of computer generated scenes. Interactive rendering of reflections on curved objects is not supported well by current rendering techniques. In this section, we first review some previous work on simulating reflections. Then, we classify three categories of reflection problems and propose our perturbation approach as the third category of reflection problem.

3.1.1 Background

The most general and accurate means of simulating reflections is ray tracing [86], 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 [79]. However, ray tracing is generally impractical for interactive applications in spite of extensive work on ray tracing acceleration schemes mentioned in Chapter 1.

A popular alternative to ray tracing is environment mapping [16], which can produce convincing visual effects that mimic specular reflection with far less computation. 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 [59] 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. As we discussed in Chapter 2, their work was essentially the starting point of our perturbation theory.

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 considering the virtual objects 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 [66] described how several hidden surface algorithms could be enhanced to include reflections in curved surfaces, essentially by accommodating virtual objects. Rushmeier and Torrance [67] 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 [64] 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.

3.1.2 Perturbation Approach

Figure 3.1 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? Similarly, 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? 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.

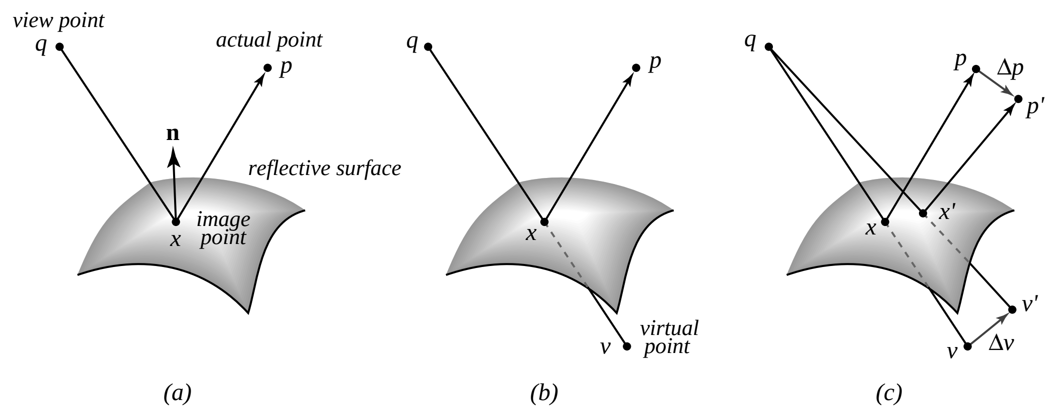


Figure 3.1: *Three different reflection problems. (a) Computing the point p as seen reflected at x. (b) Computing the virtual point v corresponding to the actual point p. (c) Computing how the position of the virtual point v changes as the actual point p is perturbed.*

See Figure 2.1. Secondly, while the reflected point can be found easily using only the surface normal at the reflection point (and standard ray-intersection procedures), the point of reflection, or the virtual point, are generally much more difficult to express as a function of the viewpoint and the reflected point. While the reflection seen at a given point is typically easy to express analytically, finding the position of a virtual object usually requires numerical approximation, even for very simple reflecting geometries. For example, Mitchell et al. [59] had to use interval Newton method and automatic differentiation to find the positions of all the reflection points to compute the irradiance.

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. The basic tool is the closed-form expressions for the first-order and higher order approximations of a reflection ray path, which we have derived in Chapter 2. In the next section, we describe an algorithm for rapidly computing highly accurate specular reflections in arbitrary curved surfaces by exploiting this third category of reflection problem.

By characterizing how specular reflections change as a result of perturbing the actual object, 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. The technique uses first-order or second-order perturbations of known reflection paths, which are ray traced in a pre-processing step, to compute virtual objects for each reflection. Depending on the local curvature estimate of curved reflecting surfaces, perturbations of different orders can be chosen flexibly to trade for more efficiency. The virtual objects are then rendered using standard graphics hardware supporting both alpha-blending and z-buffering. Thus, from a sparse set of known reflections, we can quickly construct any reflection at all. This amounts to an interpolation method for reflections, as it allows a continuous reflection to be constructed from discrete samples in such a way that the exact behavior at those samples is preserved. From these interpolated reflections, virtual objects are constructed and rendered with real objects using graphics hardware supporting alpha blending and z-buffering. The entire process is fast enough for real-time interaction, and the simulated reflections are of very high accuracy; our test results were nearly identical to ray traced images.

Our proposed method is most similar in spirit to the work of Ofek and Rappoport. In both instances, reflections in curved surfaces are simulated by constructing virtual objects that can be efficiently rendered. Our most significant point of departure is in the use of perturbation theory [52] to accurately locate the position of the virtual object. In order to take advantage of hardware benefits, our approach reduces the problem of rendering both real and virtual objects with correct depth information to one of z-buffering, which is similar to Panduranga's modified hidden surface algorithms. By using second-order approximation, we can attain extremely faithful reflection geometry. See Figure 3.4 for a comparison of our results with ray tracing. The reflections in the vase created by the two methods are visually indistinguishable, yet the perturbation method is several orders of magnitude faster than ray tracing.

The rest of this chapter is organized as follows: Section 3.2 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 formulae derived in Chapter 2. Some convincing results of this perturbation approach are demonstrated in Section 3.3. We conclude this chapter with a discussion of future directions in Section 3.4.

3.2 A Perturbation Approach for Specular Reflections

The analytic formula (2.77) for perturbing specular reflections shown in the previous section can be used to approximate reflections in arbitrary curved surfaces by interpolation from known reflections. In this section we present an algorithm for rapidly computing highly accurate reflections in curved specular surfaces. By combining ray tracing and hardware alpha-blending and z-buffering, the entire process is fast enough for real-time interaction, and the reflection images are highly accurate.

3.2.1 Overview

An overview of the process is shown in Figure 3.2. 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, 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 Figure 3.2. Initially, for each tessellated vertex of a reflected object, these steps compute its virtual vertices to generate specular reflections of the scene. During an interaction session, they recompute the virtual vertices for modified vertices in the environment. This is done in four steps for each vertex: *finding the closest reflection rays using the octree*, *computing path Jacobians and path Hessians*, *interpolating new reflection points using perturbation*, and *computing new virtual vertices.* Finally, both the real objects and the virtual objects are rendered using standard graphics hardware supporting both alpha-blending and z-buffering. After tessellation, the specular reflectors are rendered as transparent surfaces, with transparency determined by their reflectivity.

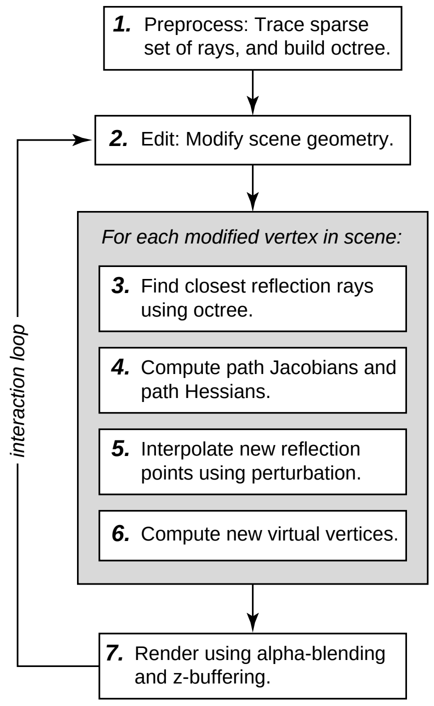

1. Preprocess: Trace sparse set of rays, and build octree.
2. Edit: Modify scene geometry.
For each modified vertex in scene:
3. Find closest reflection rays using octree.
4. Compute path Jacobians and path Hessians.
5. Interpolate new reflection points using perturbation.
6. Compute new virtual vertices.
7. Render using alpha-blending and z-buffering.
interaction loop

Figure 3.2: *Overview of the algorithm for interactively computing specular reflections using local perturbations.*

The following sections describe these steps in detail.

3.2.2 Preprocessing

Sparse Sampling

Our perturbation formula requires the existence of true reflection paths in order to interpolate new reflections. Owing to the second-order accuracy of our interpolation, 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 through specular reflectors in the scene, 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. Depending on the maximum level of specular reflections to be handled, we store all reflection paths with a depth less than or equal to this level. All reflection positions and associated reflecting surfaces along the path are retained. The reflection paths are distinguished into different types according to the number and the order of these bounces. For example, for a scene of 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, 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 the example of two refelctors, 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 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 is used to reduce the computation cost.

Constructing the Ray Octree

Because our approach interpolates by perturbing several nearest-neighbors, it is important to find k 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 pre-computed 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 following algorithm 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 [69].

```
RayCandidateSet( Sphere $S$, Rays $\mathcal{R}$, integer $k$ )
    Point $A$ ← center of sphere $S$
    Scalar $r$ ← radius of sphere $S$
    Rays $\mathcal{R}^*$ ← empty set
    Compute distance interval for each ray. Keep all rays that intersect $S$.
    for each ray $\varrho \in \mathcal{R}$ do
        $d$ ← distance from ray $\varrho$ to the point $A$
        $\varrho$.min ← $d - r$
        $\varrho$.max ← $d + r$
        if $\varrho$.min $\leq 0$ then add $\varrho$ to $\mathcal{R}^*$
    endfor
    Ensure that the set $\mathcal{R}^*$ contains at least $k$ rays.
    if $|\mathcal{R}^*| < k$ then
        Rays $\mathcal{S}$ ← elements of $\mathcal{R} - \mathcal{R}^*$ sorted by increasing min distance
        add to first $k - |\mathcal{R}^*|$ elements of $\mathcal{S}$ to $\mathcal{R}^*$
    endif
    Include all rays that are among the $k$ closest to some point in $S$.
    Scalar $\alpha$ ← maximum of $\varrho$.max among all $\varrho \in \mathcal{R}^*$
    for each $\varrho \in \mathcal{R} - \mathcal{R}^*$ do
        if $\varrho$.min $< \alpha$ then add $\varrho$ to $\mathcal{R}^*$
    endfor
    For any point $\mathbf{p} \in S$ the subset $\mathcal{R}^* \subset \mathcal{R}$ now contains the $k$ rays of $\mathcal{R}$
    that are closest to $\mathbf{p}$ (and possibly others).
    return $\mathcal{R}^*$
    end
```

Given a sphere S, a set of rays $\mathcal{R}$, and an integer k, the above algorithm 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 $\mathcal{R}$.

3.2.3 Computing Virtual Objects

After the preprocessing, the sparsely-sampled reflection rays are retained in a hierarchical structure. Each ray is tagged to record its position along the reflection path. Then, whenever an object manipulation changes the scene geometry, 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 tessellated object vertex. 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 have several virtual objects, with respect to different reflectors or of different levels. As stated in Section 3.2.2, different types of virtual objects can be perturbed from different types of sample rays. In the following sections, we describe how to compute a virtual vertex for an object vertex in detail.

Finding Closest Reflection Rays

Once the known reflections 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 points in a concave surface, Figure 3.4 illustrates such an example. To generate multiple reflections in a non-convex surface, we need a mechanism to 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 experimentation, we use 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 have close reflection points and similar reflection directions. We compute the angle between two reflection directions to measure the direction similarity. The threshholds for distance and angle are currently specified by the user.

For a given point $\mathbf{p}'$, we first find 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 predefined 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, it 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.

Computing Path Jacobian and Path Hessian

For a given point $\mathbf{p}'$, once the closest reflection ray is found, we can calculate 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 simply reduced to the third reflection problem shown in Figure 3.1(c), that is, computing how the position of the known reflection point $\mathbf{x}$ changes as the actual point $\mathbf{p}$ is perturbed by $\Delta\mathbf{p} = \mathbf{p}' - \mathbf{p}$. In order to apply the perturbation formula (2.77) to approximate the new reflection point, we compute the path Jacobians and/or path Hessians for the closest reflection path found. For an N-bounce path, the details to compute the path Jacobians and the path Hessians have been shown in the pseudo code of Section 2.5.

Interpolating Reflection Points

Let $\mathbf{p}'$ be a point near $\mathbf{p}$. Suppose $\mathbf{p} - \mathbf{x}_1 - \cdots - \mathbf{x}_N - \mathbf{q}$ 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 $\mathbf{x}_N$ 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 reflection of $\mathbf{p}'$ to different accuracy. That is, the new reflection point $\mathbf{x}'_N$ can be approximated to first-order accuracy by

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} \tag{3.1}$$

or to second-order accuracy by

$$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{J}_i \Delta\mathbf{x}_{i-1} + \frac{1}{2}(\Delta\mathbf{x}_{i-1})^{\mathrm{T}} \mathbf{H}_i \Delta\mathbf{x}_{i-1} \tag{3.2}$$

where $i = 1, 2, 3, \ldots, N$, $\Delta\mathbf{x}_{i-1} = \mathbf{x}'_{i-1} - \mathbf{x}_{i-1}$ and $\Delta\mathbf{x}_0 = \Delta\mathbf{p}$. Shown in Figure 3.3 are several known single-level reflection paths associated with the given view point and reflector. All other single-level reflections are approximated 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.

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 (software shading based on a chosen shading model) 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

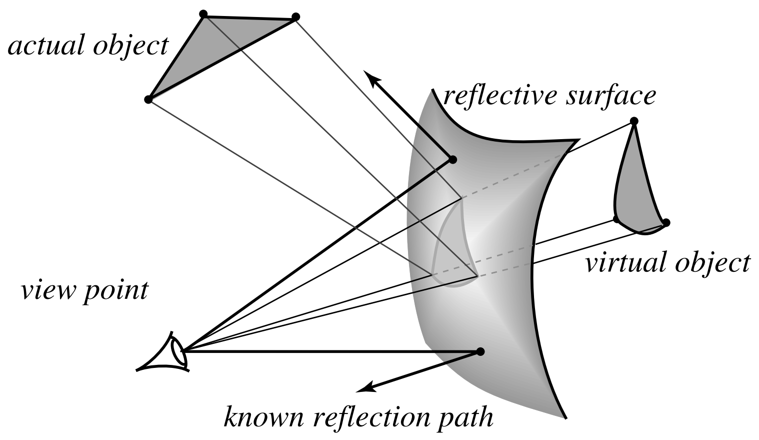


Figure 3.3: *The reflection of an actual object is computed by perturbing several known reflection paths. All virtual objects are then z-buffered with the rest of the environment. The reflecting surface is alpha-blended, with transparency corresponding to reflectivity of the actual surface.*

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 pseudo-code below. Here, the virtual vertex computed is for a single-level specular reflection, the point **q** passed in is the view position where the sample rays originate and the second-order perturbation is used.

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

    Cube $C \leftarrow$ cell containing $\mathbf{p}'$
    Rays $\mathcal{R} \leftarrow$ candidate set of $C$
    Find the nearest ray to $p'$ with a linear search.
    Ray $\varrho \leftarrow$ the ray of $C$ closest to $\mathbf{p}'$
    Perturb the path with equation (3.2).
    Point $\mathbf{x}, \mathbf{p}, \mathbf{x}'$
    $\mathbf{x} \leftarrow$ origin of the ray $\varrho$
    $\mathbf{p} \leftarrow$ the point on $\varrho$ closest to $p'$
    Vector $\Delta\mathbf{p} \leftarrow \mathbf{p}' - \mathbf{p}$
    Matrix $\mathbf{J} \leftarrow$ OneBouncePathJacobian( $\mathbf{p}, \mathbf{x}, \mathbf{q}, \mathbf{0}, g$ )
    Tensor $\mathbf{H} \leftarrow$ OneBouncePathHessian( $\mathbf{p}, \mathbf{x}, \mathbf{q}, g, \mathbf{J}, \mathbf{0}, \mathbf{0}$ )
    $\mathbf{x}' \leftarrow \mathbf{x} + \mathbf{J}\Delta\mathbf{p} + \frac{1}{2}\Delta\mathbf{p}^{\mathrm{T}}\mathbf{H}\Delta\mathbf{p}$
    Create the virtual vertex .
    Point $\mathbf{v}' \leftarrow \mathbf{x}' + \frac{||\mathbf{x}' - \mathbf{p}'||}{||\mathbf{x}' - \mathbf{q}||}(\mathbf{x}' - \mathbf{q})$
    return $\mathbf{v}'$
```

3.2.4 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 standard graphics hardware in this context is made possible by the following facts:

- Virtual objects are positioned behind the reflective surface with respect to the view point. This allows reflectivity of a surface to be computed using alpha-blended transparency to "merge" specular reflections onto real objects. This is similar to image merging method proposed by Ofek and Rappoport [64].

- 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.

3.3 Implementation and 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.

For a scene with a reflecting vase and a diffuse polygonal lizard, Figure 3.4 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 is 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.

As an example of generating multiple-bounce specular reflections by the perturbation

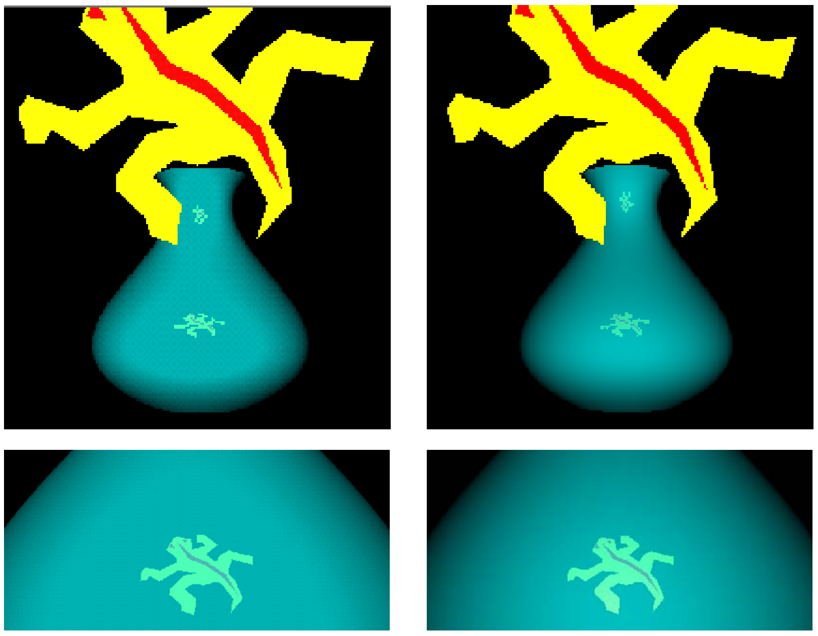


Figure 3.4: *Side-by-side comparison of one-bounce reflection images generated by the perturbation method (left) and ray tracing (right). The results are nearly identical, yet the images in the left column can be computed very rapidly (approximately 0.1 seconds per update) as the lizard-shaped polygon is moved interactively.*

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 Figure 3.5. Similarly, the bottom images show a closeup view of the two-bounce reflections. 80×60 sample rays were used for this 640×480 image. The reflections of the vase and the sphere are constructed by tessellating them into 1250 triangles, but the directly-viewed vase is rendered with a finer tessellation (20000 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.

Excluding the preprocessing time from our perturbation method and the time for

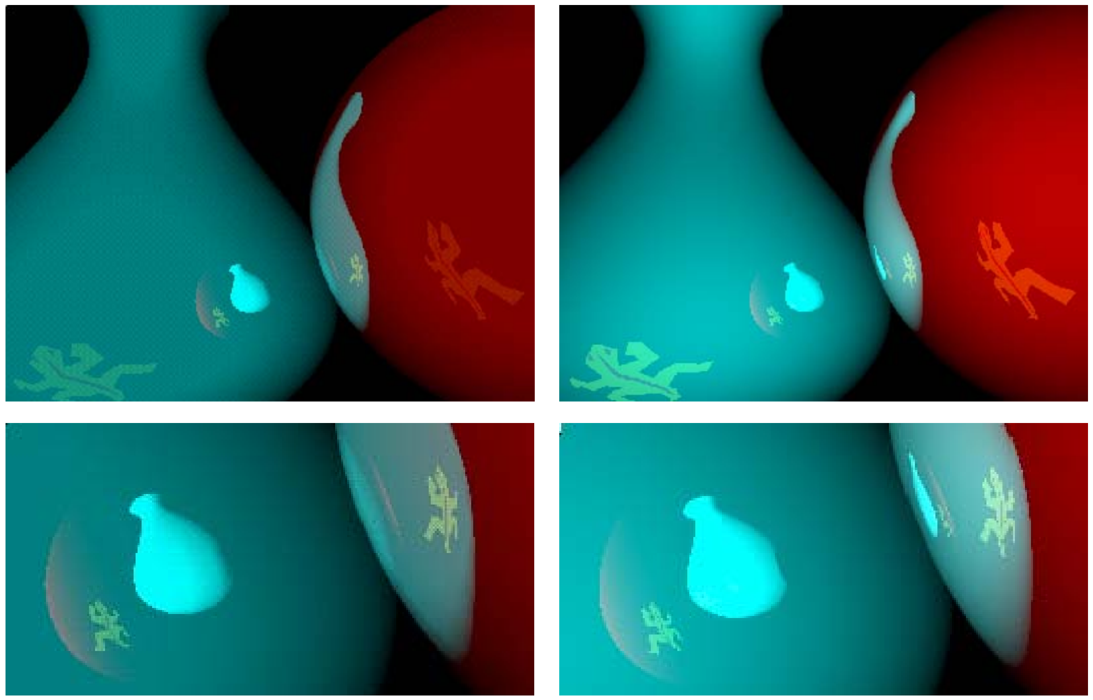


Figure 3.5: *Side-by-side comparison of multiple-bounce reflection images generated by the perturbation method (left) and ray tracing (right). Perturbed image in the left computes the reflections up to two bounces, ray traced image in the right is generated with maximum depth of* 5.

parsing and creating bounding slab tree from ray tracer, the performance on these two scenes is summarized in Table 3.1. 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. Figures 3.4 and 3.5 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 Figures 3.4 and 3.5, each update takes *less than one second*.

In Figure 3.6, 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

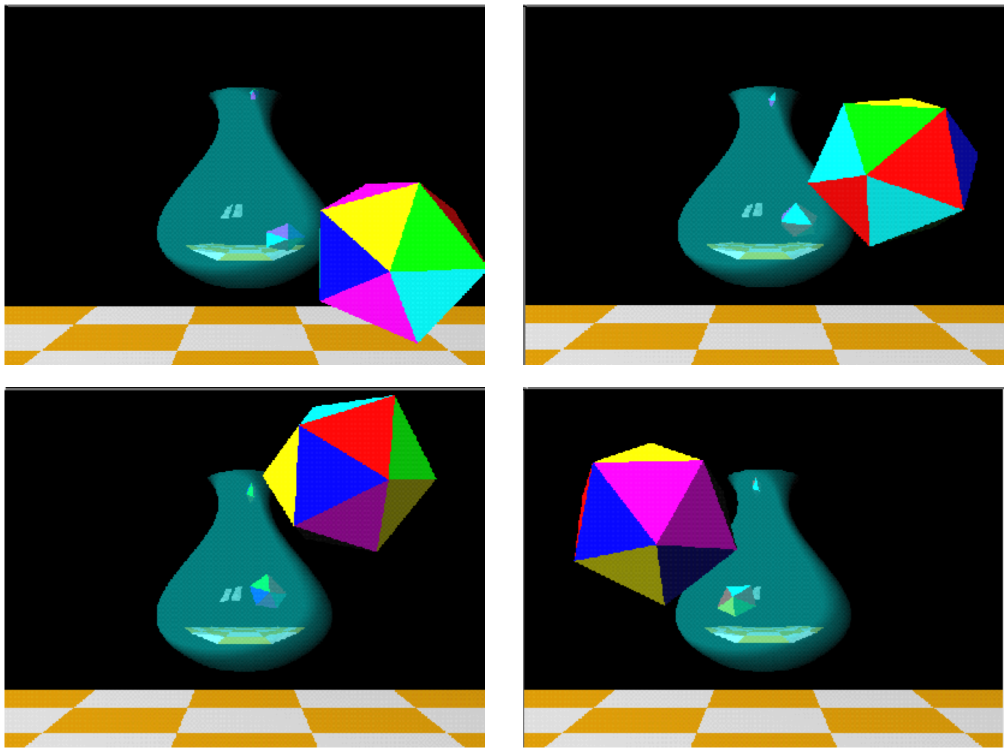
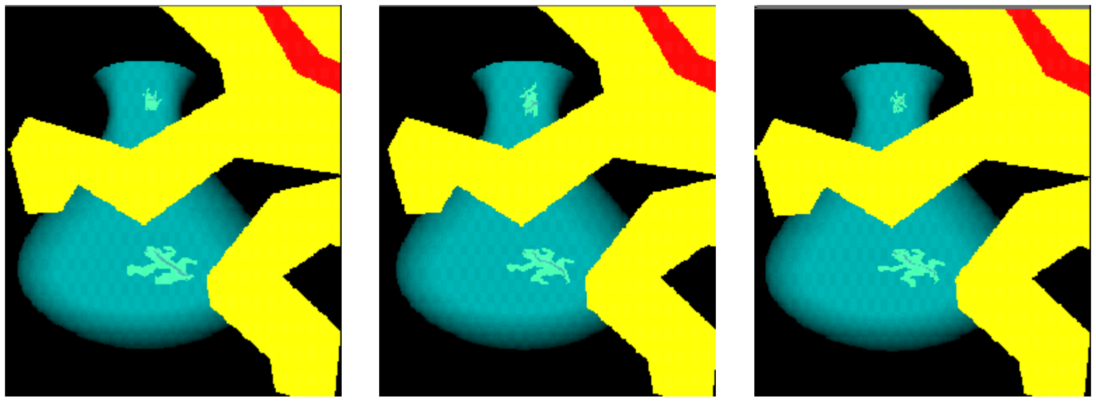


Figure 3.6: *Images generated interactively by repositioning, rotating, and scaling the icosahedron. The reflection of the icosahedron is updated in real time as it is moved. Hidden surface removal is performed in one pass of z-buffering for the entire scene, including the reflections.*

	initial scene		moving lizard	
	vase	vase-sphere	vase	vase-sphere
Perturbation(sec)	0.43	9.6	0.1	0.7
RayTracer(sec)	41	67	41	67

Table 3.1: *Timing comparison between perturbation method and ray tracing.*

interpolated to second-order. Hidden surface removal is performed by z-buffering the entire scene, including the reflections simulated by virtual objects. Figure 3.6 shows several frames in an animation sequence, which are generated interactively at 30 frames/sec.



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

Figure 3.7: *Varying sample density affects reflection quality and rendering time.*

3.4 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 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.

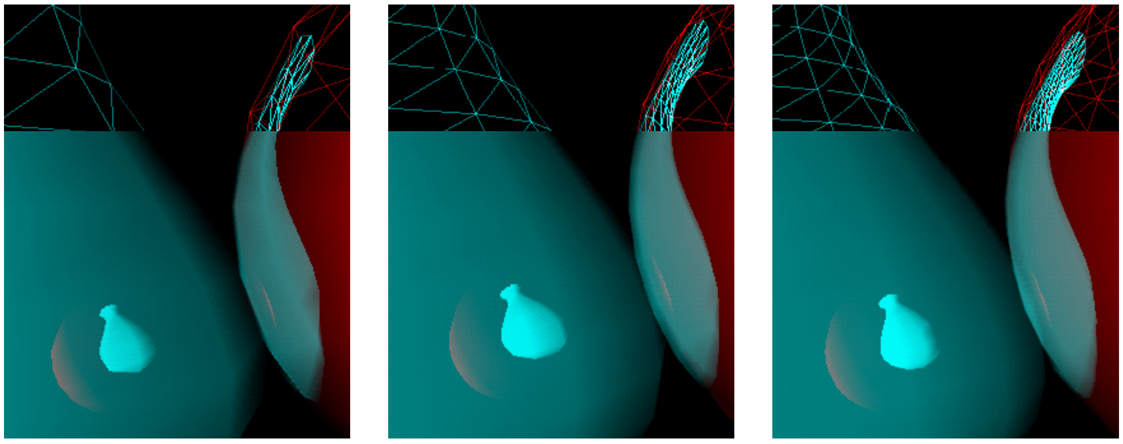
Next we discuss the limitations of our perturbation approach and several aspects affecting its reflection quality and performance, and compare it with other related work.

Limitations: While computing virtual objects in Section 3.2.3, 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 [64] 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 regards to concave reflectors, which can produce complicated reflections. Ofek and Rappoport [64] 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), the resulting reflection image is “chaotic”, different object vertices may have a different number of virtual vertices. Our virtual object algorithm cannot handle such a crossing since it is hard to find the correspondence between these virtual vertices to correctly connect the virtual objects. 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.

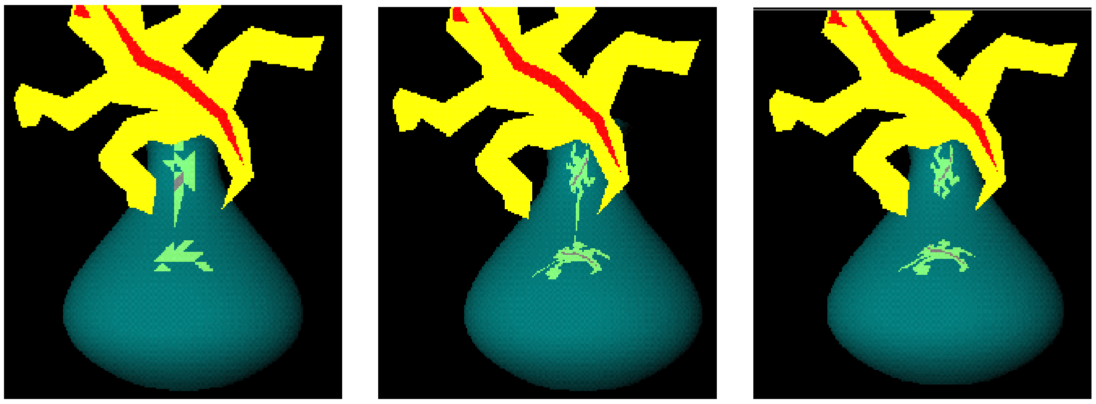
Sample Density: Since the second-order perturbation has an error of order $\mathrm{O}(||\,\Delta\mathbf{p}\,||^3)$, the accuracy of our perturbation method depends upon the underlying sample rays. Denser sampling leads to smaller $||\,\Delta\mathbf{p}\,||$, and thus a higher accuracy. Figure 3.7 shows 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 the second-order approximation, the sampling over the image plane can be relatively sparse. For the scene in Figure 3.7, 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. Figure 3.8 compares specular reflections as well as rendering times correponding to different levels of tessellation. Each



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

Figure 3.8: *Varying tessellation levels cause different qualities and rendering times.*



0.36s 0.39s 0.53s

Figure 3.9: *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.*

reflection image is generated by tessellating the vase and the sphere with the number of triangles shown beneath each image.

Approximation Order: Figure 3.9 shows three images generated 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 [64]. As mentioned in Section 3.1, environment mapping fails for relatively close dynamic objects. 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[1]. 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

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

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 order), 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 effcient for some special types of reflectors, such as planar surfaces, linear extrusions, and spherical-like surfaces, while our method is more accurate for more 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 perturbation or the explosion map, depending on the reflective geometry and accuracy requirements.

Chapter 4

Direct Computation of Caustic Contours

Caustics provide some of the most visually spectacular patterns of light in nature. Caustics are formed when light reflected from or transmitted through a specular surfaces strikes a diffuse surface. Examples of caustics are the light patterns on the bottom of a swimming pool and light focused onto a table through a glass of wine.

Radiosity is not good at rendering caustics which are indirect (reflected or transmitted) illumination on the diffuse surfaces. Due to the natural connection between caustics and reflection (or transmission) paths, some researchers have explored to simulate caustics based on ray tracing. Cook [27], Kajiya [48] used stochastic ray tracing methods to approximate caustics to low accuracy. Arvo [4] introduced a preprocessing step to compute caustics. The preprocessing step uses backward ray tracing (also known as light ray tracing, photon tracing) to emit photons towards the specular surfaces, photons get reflected and stored on the Lambertian surfaces as illumination maps, which are used as texture maps for caustics. Heckbert [39] adaptively subdivided the illumination map into area elements with a size corresponding to the local density of the photon-hits. However, Illumination map can't be used with all objects since it requires describing the surfaces parametrically. Instead, Jensen et al. [45] stored all photon-hits explicitly in a photon map and introduced a new technique to extract radiance information from photon maps. The idea of photon map was also extended by Jensen [44] to incorporate BRDF information for rendering caustics on non-Lambertian surfaces. The problem of photon maps is the large memory request and inefficiency of its two-pass rendering. Finally, Mitchell et al. [59] integrated the computation of caustics into standard ray tracing by computing illumination from light reflected or transmitted via specular surfaces. They solved the ray paths from Fermat's principle and then computed indirect illumination by wavefront tracing these paths. Their method is unfortunately very complicated and time consuming.

In this chapter, we propose a new technique to generate caustics on diffuse surfaces by directly computing *caustic contours*, which are curves on the surface with constant irradiance due to indirect illumination. The basic idea of this approach is to formulate the caustic contour as an ordinary differential equation (ODE) and solve it with classical predictor-corrector methods. The ODE formulation is based on the closed-form expression for caustic irradiance gradient, which can be derived from wavefront tracing and the closed-form formula for the path Jacobian of a specular path derived in Chapter 2. Some examples of caustic contours generated with this method shows it a promising step towards rendering caustics efficiently and separately. The rest of this chapter is organized as follows: Section 4.1 clarifies the concept of caustics we use here and reviews some elements in differential geometry and the basics of wavefront tracing. These preliminaries are the building blocks for the complicated derivations followed. Next, in Section 4.2, we present an algorithm to directly compute caustic contours on a diffuse plane due to a point light source. Particularly, we give a detailed derivation for *caustic irradiance* and *caustic irradiance gradient*, which are the key elements for the whole process. Finally, some examples of caustic contours are demonstrated and problems with our caustic contour algorithm are discussed in Section 4.3.

4.1 Preliminaries

4.1.1 Caustic Contour

In geometrical optics, the *caustic* is the *envelop* to a family of rays transmitted by a lens or reflected off a mirror. By an *envelop* we mean a curve or surface (in 3D) tangent to a member of the family at each point. The word "caustic" is from the diminutive form of the Greek word for "burning iron", since a caustic can be seen as a bright spot with infinite brightness formed when light refracts through a transparent object or reflects off a curved surface in such a way that it is focused into a small area.

The term "caustic" is always used very loosely within computer graphics. What we colloquially call "caustics" refers to indirect illumination effects on diffuse surfaces, they are formed by light that is reflected or transmitted by a number of specular surfaces before interacting with a diffuse surface. In this sense, caustics are not necessary to be infinite bright. In this chapter, we adopt the graphical meaning of this term and call any reflected or refracted illumination on a diffuse surface "caustics".

Specifically, we are interested in the case where a point light source shines upon an implicitly-defined curved reflector and the reflected light creates caustics on a diffuse surface, for example, a plane for simplicity. Different locations on the plane will receive different irradiance from the point light source through reflective paths via the reflector. Analoguous to *isolux contours*, curves of constant irradiance on the plane due to reflected illumination are defined as *caustic contours*. In this thesis, we are focusing on generating caustic contours by direct computation rather than by post-processing an image, therefore, they may be used in the image generation process to render caustics on diffuse surfaces rapidly. Besides, caustic contours can also find many other potential use, for example, depicting caustic irradiance distribution, guiding adaptive subdivision of illumination maps, simplifying shading computations, and so on.

4.1.2 Elements of Differential Geometry

Shape Operator

The shape of a surface M in Euclidean space $\mathbb{R}^3$ can be described infinitesimally by a certain linear operator $\mathbf{S}$ defined on each of the tangent planes of M.

Definition 1 (Shape Operator) *If* $\mathbf{p}$ *is a point of a regular surface* M*, then for each vector* $\mathbf{v}$ *that is tangent to* M *at* $\mathbf{p}$*, let*

$$S_p(\mathbf{v}) = -\mathbf{D_v n}$$

where $\mathbf{n}$ *is a unit normal vector field on a neighborhood of* $\mathbf{p}$ *in* M *and* $\mathbf{D_v}$ *is the directional derivative in the* $\mathbf{v}$ *direction.* S_p *is called the Shape operator of* M *at* $\mathbf{p}$*.*

The rate of change of the surface normal $\mathbf{n}$ in the $\mathbf{v}$ direction tells how the tangent planes of M are varying in the $\mathbf{v}$ direction, and thus *shape operator* gives an infinitesimal description of the way M itself is curving in the space. If the surface normal $\mathbf{n}$ changes sign, then S_p changes to $-S_p$. As a linear operator $S_p : T_p(M) \rightarrow T_p(M)$ on the tangent space of M at $\mathbf{p}$[65, pp.190–191], shape operator can be expressed as a Jacobian matrix

$$\mathbf{S} = -\frac{d\mathbf{n}}{d\mathbf{x}}, \tag{4.1}$$

which is refered later as *shape operator matrix.*

For an implicit surface defined by $f(x, y, z) = 0$, the surface normal along the ray propagation is

$$\mathbf{n} = -\frac{\mathbf{g}}{||\,\mathbf{g}\,||},$$

where

$$\mathbf{g} = \nabla f,$$

and

$$||\,\mathbf{g}\,|| = \sqrt{f_x^2 + f_y^2 + f_z^2}.$$

Let $\mathbf{P}(\mathbf{w})$ denote the projection matrix around any given vector $\mathbf{w}$, which is defined as

$$\mathbf{P}(\mathbf{w}) = \mathbf{I} - \frac{\mathbf{w}\mathbf{w}^{\mathrm{T}}}{\mathbf{w}^{\mathrm{T}}\mathbf{w}} \tag{4.2}$$

where $\mathbf{I}$ is the identity matrix. Then, we can compute the derivative of $\mathbf{n}$ as:

$$\frac{d\mathbf{n}}{d\mathbf{x}} = -\frac{d\mathbf{g}/d\mathbf{x}}{||\,\mathbf{g}\,||} + \frac{\mathbf{g}(\mathbf{g}^{\mathrm{T}} \cdot d\mathbf{g}/d\mathbf{x})}{||\,\mathbf{g}\,||^3} = -\frac{\mathbf{P}(\mathbf{g})}{||\,\mathbf{g}\,||}\frac{d\mathbf{g}}{d\mathbf{x}}, \tag{4.3}$$

where

$$\mathbf{g}\mathbf{g}^{\mathrm{T}} = \begin{bmatrix} f_x f_x & f_x f_y & f_x f_z \\ f_y f_x & f_y f_y & f_y f_z \\ f_z f_x & f_z f_y & f_z f_z \end{bmatrix}$$

and $d\mathbf{g}/d\mathbf{x}$ is the Hessian matrix

$$\mathbf{H} = \frac{d\mathbf{g}}{d\mathbf{x}} = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{bmatrix}$$

Combining equation (4.3) and the definition in (4.1), we finally get the shape operator matrix for an implicit surface f:

$$\mathbf{S} = -\frac{d\mathbf{n}}{d\mathbf{x}} = \frac{\mathbf{P}(\mathbf{g})}{||\,\mathbf{g}\,||}\mathbf{H}. \tag{4.4}$$

Normal Curvature and Principal Curvature

The local geometric properties of a surface can be derived by computing its curvatures in different directions. Given a unit tangent direction $\mathbf{t}$ and a curve C on the surface passing through P with the tangent $\mathbf{t}$, suppose $\mathbf{p}$ is the principal normal vector to the curve C at P and $\mathbf{n}$ is the surface normal at P , then the curvature vector $\mathbf{k} = \kappa\mathbf{p}$ of C can be decomposed into a component normal to the surface and a component tangential to the surface, that is,

$$\mathbf{k} = \kappa_n \mathbf{n} + \kappa_g(\mathbf{n} \times \mathbf{t}).$$

where κ_n is called *normal curvature* and κ_g is called *geodesic curvature.*

All curves passing through P with the same direction have the same normal curvature vector, therefore, normal curvature is well-defined for all directions tangent to the surface. With the shape operator formula in (4.4), normal curvature in a given tangent direction can be computed from the following lemma.

Lemma 1 *Let* $\mathbf{t}$ *be a unit tangent vector at a point* $\mathbf{x}$ *of a regular surface* $M \subset \mathbb{R}^3$. *Then the normal curvature of* M *in the direction* $\mathbf{t}$ *at* $\mathbf{x}$ *is*

$$\kappa_n(\mathbf{t}) = \mathbf{t} \cdot \mathbf{S}\mathbf{t}. \tag{4.5}$$

<u>Proof:</u> Suppose an arc length parameterized curve $C : \mathbf{x}(s)$ passing through $\mathbf{x}$ on the surface M and has the unit tangent vector $\mathbf{t}$. $\mathbf{n}$ is the surface normal at $\mathbf{x}$. Then, from the definition of normal curvature, the curvature vector of C at $\mathbf{x}$ can be decomposed as

$$\mathbf{k} = \frac{d\mathbf{t}}{ds} = \kappa_n \mathbf{n} + \kappa_g(\mathbf{n} \times \mathbf{t}).$$

Hence, applying inner product on both sides

$$\kappa_n = \mathbf{k} \cdot \mathbf{n} = \frac{d\mathbf{t}}{ds} \cdot \mathbf{n} = -\mathbf{t} \cdot \frac{d\mathbf{n}}{ds} = \mathbf{t} \cdot \left(-\frac{d\mathbf{n}}{d\mathbf{x}} \frac{d\mathbf{x}}{ds} \right) = \mathbf{t} \cdot \mathbf{S}\mathbf{t}$$

where $\mathbf{S}$ is the shape operator matrix at $\mathbf{x}$. □□

Among the tangent directions of the surface, normal curvature attains a minimum and a maximum value along two perpendicular directions called the *line of curvature*, or *principal directions.* These extrema are called *principal curvatures* and *Gaussian* *curvature* is defined as the product of these two principal curvatures. Given two principal directions, the normal curvatures in two other perpendicular directions can computed from two principal curvatures with *Euler's theorem.*

Theorem 3 (Euler's Theorem) *The normal curvatures in two perpendicular directions* $\mathbf{u}$ *and* $\mathbf{v}$ *are given by*

$$\begin{aligned} \kappa_u &= \kappa_\xi \cos^2\theta + \kappa_\eta \sin^2\theta \\ \kappa_v &= \kappa_\xi \sin^2\theta + \kappa_\eta \cos^2\theta \\ \kappa_{uv} &= (\kappa_\xi - \kappa_\eta)\cos\theta\sin\theta, \end{aligned} \tag{4.6}$$

where θ *is the angle from the direction* $\mathbf{u}$ *to the principal direction* ξ *in a right-handed orientation of the tangent plane.*

The equation (4.6) can be inverted to find the principal curvatures, given the normal curvatures in two other orthogonal directions. We state this inverse relationship as *inverse Euler's formula*,

$$\begin{aligned} \tan 2\theta &= 2\kappa_{uv}/(\kappa_u - \kappa_v) \\ \kappa_\xi &= \kappa_u \cos^2\theta + 2\kappa_{uv}\cos\theta\sin\theta + \kappa_v \sin^2\theta \\ \kappa_\eta &= \kappa_u \sin^2\theta - 2\kappa_{uv}\cos\theta\sin\theta + \kappa_v \cos^2\theta. \end{aligned} \tag{4.7}$$

Geodesics and Geodesic torsion

κ_{uv} in Euler's formula (4.6) is related to *geodesic torsion.* A curve C on a surface M is called *geodesic curve* or *geodesic* if its geodesic curvature κ_g vanishes identically. Here, we will state some useful theorems about geodesic curves, which can be proved with *the first fundamental form* or *the second fundamental form.* Please see Kreyszig [50] for proofs.

Theorem 4 *Straight lines on any surfaces are geodesics. A curve* C *of class* $r \geq 2$*, not a straight line, is a geodesic on a surface* M *if and only if at each point of* C *at which the curvature* κ *of* C *is not zero , the unit normal vector* $\mathbf{n}$ *to* M *lies in the osculating plane* O *of* M*.*

Theorem 5 *At any point P on a surface M, there is exactly one geodesic passing through P in a given tangent direction.*

Theorem 6 *The torsion of two geodesic curves passing through a point and perpendicular to each other are equal in magnitude and opposite in sign.*

Given a geodesic curve $G : \mathbf{x}(s)$ with a unit tangent direction $\mathbf{t}$ on a surface M, according to Theorem 4, the principal normal vector $\mathbf{p}$ of G and the surface normal $\mathbf{n}$ of M coincide. By choosing a suitable orientation of the surface M, we have

$$\mathbf{p} = \mathbf{n}.$$

Hence, G has a unit binormal vector

$$\mathbf{b} = \mathbf{t} \times \mathbf{p} = \mathbf{t} \times \mathbf{n}.$$

Also, Fresnet formula gives us

$$\dot{\mathbf{n}} = -\kappa \mathbf{t} + \tau \mathbf{b}.$$

Applying inner product with $\mathbf{b}$ on both sides, we attain

$$\tau = \dot{\mathbf{n}} \cdot \mathbf{b}, \tag{4.8}$$

where $\dot{\mathbf{n}}$ denotes the derivative of $\mathbf{n}$ with respect to the arc length parameter s, that is,

$$\dot{\mathbf{n}} = \frac{d\mathbf{n}}{ds} = \frac{d\mathbf{n}}{d\mathbf{x}}\frac{d\mathbf{x}}{ds} = \frac{d\mathbf{n}}{d\mathbf{x}}\mathbf{t} = \mathbf{S}\mathbf{t}$$

and the shape operator matrix $\mathbf{S}$ can be computed with equation (4.4). Equation (4.8) shows that the expression for the torsion of a geodesic G at a point P of M depends only on the surface point P and on the direction tangent to G at P. Combining Theorem 5, we may thus define the *geodesic torsion* at a point P of a surface M in a given tangent direction is the torsion of the geodesic curve G passing through P in this direction. With the geodesic torsion formula (4.8), we can also prove

Lemma 2 *Let* $\mathbf{u}$*,* $\mathbf{v}$ *be two orthonormal directions on the tangent plane at a point* $\mathbf{x}$ *of a regular surface* M*,* $\mathbf{n}$ *is the surface normal at* $\mathbf{x}$ *and* $\mathbf{n}$*,* $\mathbf{u}$*,* $\mathbf{v}$ *forms a right-handed coordinate basis, then the geodesic torsions along the directions* $\mathbf{u}$ *and* $\mathbf{v}$ *are given by*

$$\begin{aligned} \tau_u &= \mathbf{v}\cdot\mathbf{S}\mathbf{u} \\ \tau_v &= -\mathbf{u}\cdot\mathbf{S}\mathbf{v}. \end{aligned} \tag{4.9}$$

<u>**Proof:**</u> Using the identities

$$\begin{aligned} \mathbf{u}\times\mathbf{n} &= -\mathbf{v} \\ \mathbf{v}\times\mathbf{n} &= \mathbf{u}, \end{aligned}$$

Equation (4.9) can be easily derived from equation (4.8). □□

Now, it is time to explain the meaning of the quantity κ_{uv} in Euler's formula (4.6). Given two perpendicular directions $\mathbf{u}$,$\mathbf{v}$ and a chosen surface orientation which determines the direction of the surface normal $\mathbf{n}$, κ_{uv} can refer to the geodesic torsion τ_u or τ_v. If the orientation of the surface is chosen in such a way that the surface normal $\mathbf{n} = \mathbf{u}\times\mathbf{v}$, then

$$\kappa_{uv} = \tau_u = \mathbf{u}\cdot\mathbf{S}\mathbf{v}.$$

The last identity follows from Theorem 6 and Lemma 2. If $\mathbf{n}$ calculated as such is not consistent with the one we used in deriving the shape operator in (4.4), we only need to negate the result.

4.1.3 Wavefront Tracing

In geometric optics, *wavefront* is defined relating to orthometric systems of rays. In this section, we will emphasize on the intuitive concepts rather than formal derivations.

Interested readers maybe check Stavroudis [75] for formal definition. Intuitively, a given point light will shot many rays, and we define a wavefront W to be the locus of points on each ray at a given optical path length. So, wavefronts can be thought of as isosurfaces in space. One important property of the wavefront surfaces are orthogonal to the rays. Just like a normal surface, a wavefront in the neighborhood of the ray will have well-defined geometric properties. In general, it will have two principle directions (ξ, η) and two principle curvatures (κ_ξ, κ_η). It can be locally parameterized by two orthogonal tangent directions $\mathbf{u}$ and $\mathbf{v}$, which combined with the wavefront normal $\mathbf{n}$ defines a local orthogonal coordinate system ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{n}$) for a wavefront. Corresponding to the two perpendicular directions $\mathbf{u}$ and $\mathbf{v}$, normal curvatures κ_u,κ_v and geodesic torsions τ_u, τ_v and κ_{uv} can be defined accordingly.

We are all familiar with how ray tracing works, however, we may regard each ray traced being associated with a wavefront that propagates through the medium, accounting for reflection, refraction and direct transfer operations occuring along the way. Typically, the shape of a wavefront will get changed after such optical phenomena. The evolution equations to characterize how each operation changes an incoming wavefront were derived in detail by Kneisly [43] and Stavroudis [75], and the results are listed here. Suppose α and t are the incident angle and the angle of refraction, respectively.

For transfer:

$$\begin{aligned} \kappa'_u &= \frac{\kappa_u}{1 - d\kappa_u} \\ \kappa'_v &= \frac{\kappa_v}{1 - d\kappa_v} \end{aligned} \tag{4.10}$$

where d is the distance the ray travels. Transfer doesn't change the principal directions.

For refraction:

$$\begin{aligned} \kappa_u^{(t)} &= \mu\kappa_u^{(i)} + \gamma\kappa_u^{(s)} \\ \kappa_v^{(t)} &= \mu(\cos\alpha/\cos t)^2\kappa_v^{(i)} + (\gamma/\cos^2 t)\kappa_v^{(s)} \\ \kappa_{uv}^{(t)} &= \mu(\cos\alpha/\cos t)\kappa_{uv}^{(i)} + (\gamma/\cos t)\kappa_{uv}^{(s)} \end{aligned} \tag{4.11}$$

where μ is the ratio of refractive indices in the two media and $\gamma = -\mu \cos\alpha + \cos t$.

For reflection:

$$\begin{aligned} \kappa_u^{(r)} &= \kappa_u^{(i)} - 2\cos\alpha \kappa_u^{(s)} \\ \kappa_v^{(r)} &= \kappa_v^{(i)} - \frac{2}{\cos\alpha}\kappa_v^{(s)} \\ \kappa_{uv}^{(r)} &= -\kappa_{uv}^{(i)} + 2\kappa_{uv}^{(s)} \end{aligned} \tag{4.12}$$

Actually, reflection is a special case of refraction in which $\mu = 1$ and $\cos t = -\cos\alpha$. Therefore, given an incident ray, the principal directions and curvatures of its associated wavefront at some point, the principal directions and curvatures of that wavefront at some subsequent location after transfer, reflection and refraction can be determined from equation (4.10), (4.12), (4.11) and Euler's formula in (4.6), (4.7). Initially, the wavefront for a point light source is spherical. Such a generalized ray tracing involving curvature computation is usually called *wavefront tracing*. In applying these evolution equations, we assume the surface normal is along the ray propagation. With regards to the curvature sign, we follow the convention that looking in the ray direction, all concave surfaces (or diverging wavefront) have negative principal curvatures, and all convex surfaces (or converging wavefront) have positive principal curvatures.

By keeping tracking of wavefront curvatures during wavefront tracing, we can also study how much light propagate along the ray. For a point light source, *radiant intensity*, I, is defined to be the radiant power per wavefront area at any point along the ray. Consider a set of rays which represent light moving forward over time, and two wavefront patches defined by these rays at different points in time, see Figure 4.1. For small enough patches, the area of the patch is defined by the radius of two principal curvatures and the angles subtended at the center of curvatures f_1, f_2, that is, $dA = r_1 d\theta_1 r_2 d\theta_2$. We can think of these two patches of area as the ends of a tube containing the set of rays. Although the area of the two patches is different, the total power transmitted through the tube is constant. Then, by conservation of energy, we have the following *intensity*

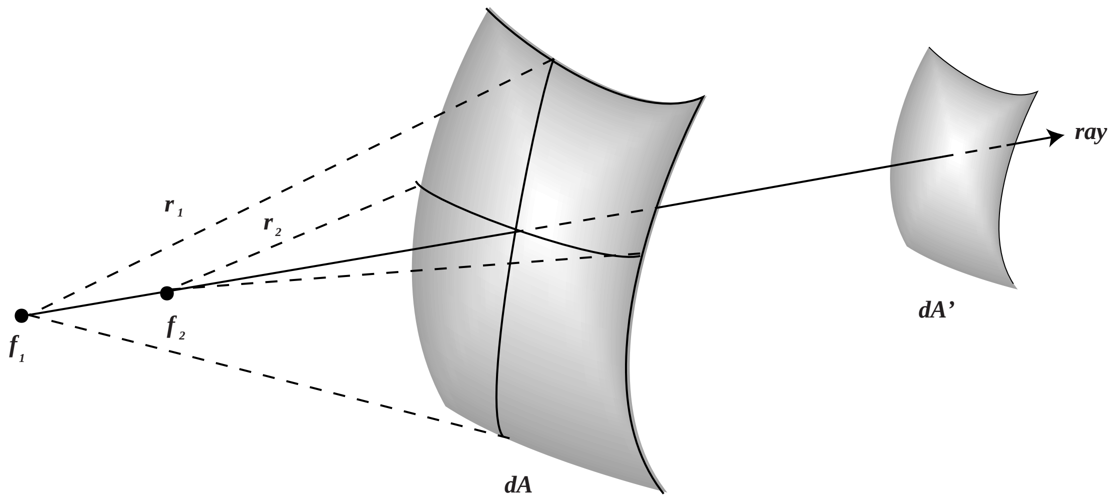


Figure 4.1: *A ray and two small patches of area, dA and dA' on two wavefronts associated with the ray. For small enough patches, the area of the patch $dA = r_1 d\theta_1 r_2 d\theta_2$. Intensity law shows that the radiant intensity will increase along the ray for such a convergent wavefront.*

law[59]:

$$\frac{I'}{I} = \frac{dA}{dA'} = \frac{r_1 r_2 d\theta_1 d\theta_2}{r_1' r_2' d\theta_1 d\theta_2} = \frac{r_1 r_2}{r_1' r_2'}.$$

This illustrates the important fact that the radiant intensity along the ray is proportional to the Gaussian curvature of the wavefront $1/(r_1 r_2)$.

4.2 An Algorithm for Generating Caustic Contours

The idea of tracing isolux contours by constructing an ordinary differential equation (ODE) has been explored by some researchers. Arvo [5] directly computed isolux contours from area light sources by combining a closed-form expression for irradiance Jacobian into an ODE. Similar idea was also used by him in volume rendering to generate iso-contours in image plane[8]. In this section, we extend this idea and propose an algorithm to directly generate caustic contours on a plane. The key element for constructing an

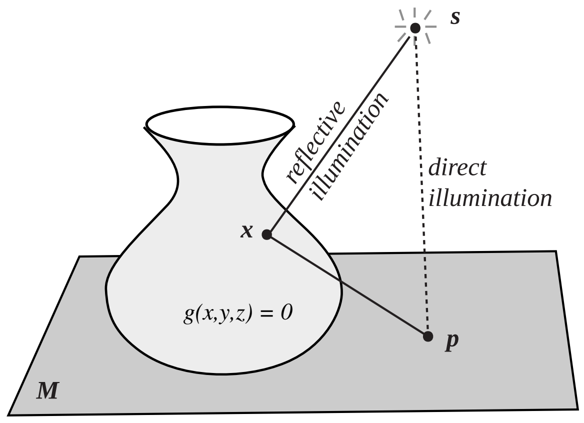


Figure 4.2: *The caustics on the plane M are formed from indirect illumination represented by reflection paths (caustic paths) from the point light source* $\mathbf{s}$ *to the plane.*

ODE for caustic contours is to have an analytical formula for the gradient of irradiance due to caustics at each plane point, which is derived from wavefront tracing and the path Jacobian formula presented in Chapter 2. Furthermore, the perturbation formula using path Jacobian is employed to update a caustic path to exploit path coherence between neighboring contour points. In the next sections, we first formulate the caustic contour problem as an ordinary differential equation in terms of caustic irradiance gradient. Then, we compute the caustic irradiance at a plane point by wavefront tracing a caustic path and using intensity law, and derive the formula for caustic irradiance gradient using the formula for the path Jacobian. Finally, a modified predictor-corrector method and a path perturbation approach are presented as an ODE solver to trace the caustic contours directly.

4.2.1 ODE Formulation

Suppose a pure specular surface $\mathbf{G}$ lies on a lambertian plane M, with a point light source $\mathbf{s}$ hanging over the scene. As shown in Figure 4.2, the caustics on the plane are formed by indirect illumination from the point light source through reflection paths. We call such reflection paths causing caustics *caustic paths*. Depending on the caustic paths reaching a plane point, the plane points at different locations will receive different irradiance from caustics. Particularly, we call such irradiance due to caustics *caustic irradiance*. With the light $\mathbf{s}$ fixed, the distribution of caustic irradiance on the plane can be described by a caustic irradiance function $\phi : \mathbb{R}^3 \rightarrow R$, where $\phi(\mathbf{p})$ returns the caustic irradiance value[1] at a plane point $\mathbf{p}$. Accordingly, caustic contours, curves of constant irradiance on the plane M beneath the curved reflector, can be represented by a function of one variable $\mathbf{r} : [0, \infty) \rightarrow M$ which satisfies

$$\phi(\mathbf{r}(s)) = \phi(\mathbf{r}(0))$$

for all s along the curve. To compute caustic contours on the plane, we first phrase this problem as a first order differential equation (ODE) to which $\mathbf{r}$ is the solution and then try to solve it numerically.

As we know, the direction of most rapid increase in $\phi(\mathbf{r})$ at a point $\mathbf{r} \in M$ is given by the gradient $\nabla\phi(\mathbf{r})$, which in general does not lie in the plane M. According to the definition of caustic contours, the projection of the gradient $\nabla\phi(\mathbf{r})$ onto the plane should be orthogonal to the caustic contour passing through $\mathbf{r}$. By rotating the projected gradient by 90 degrees, we obtain a direction in which the caustic irradiance remains constant to the first order. Thus, the differential equation can be defined as

$$\dot{\mathbf{r}} = \mathbf{P}(\mathbf{r})\nabla\phi^{\mathrm{T}}(\mathbf{r}), \tag{4.13}$$

where matrix $\mathbf{P}(\mathbf{r})$ is the product of a projection matrix and a rotation matrix

$$\mathbf{P}(\mathbf{r}) = \mathbf{R}(\mathbf{n}(\mathbf{r}))\left[\mathbf{I} - \mathbf{n}(\mathbf{r})\mathbf{n}^{\mathrm{T}}(\mathbf{r})\right].$$

Here, $\mathbf{R}(\mathbf{n}(\mathbf{r}))$ is the rotation by $-\frac{\pi}{2}$ about the normal vector at $\mathbf{r}$. Since M is a plane, its normal $\mathbf{n}(\mathbf{r})$ is constant everywhere, the matrix $\mathbf{P}$ keeps constant.

[1]We assume the irradiance as a scalar for now.

4.2.2 Caustic Irradiance

In order to compute the caustic contours by solving the ODE (4.13) numerically, we must be able to evaluate the caustic irradiance ϕ and its gradient $\nabla\phi$. According to intensity law described in Section 4.1.3, we can find the caustic irradiance $\phi(\mathbf{p})$ at a plane point $\mathbf{p}$ by keeping track of Gaussian curvature of the wavefront along a caustic path from the point light source $\mathbf{s}$ to $\mathbf{p}$ via $\mathbf{x}$ on the specular reflector. The whole tracing process can be divided into four steps:

- **Transfer Operation from the Light Source**

  Initially, the point light source $\mathbf{s}$ emits a spherical wavefront, which has no principal direction and thus the geodesic torsions $\kappa_{\xi\eta} = \kappa_{uv} = 0$. Light travels along a straight line from $\mathbf{s}$ until it hits the reflecting surface at $\mathbf{x}$. By applying transfer equation (4.10), we attain an incident wavefront at the reflection point $\mathbf{x}$:

$$\begin{aligned} \kappa_{\xi}^{(i)} &= -1/d_1 \\ \kappa_{\eta}^{(i)} &= -1/d_1 \\ \kappa_{\xi\eta}^{(i)} &= 0 \end{aligned}$$

  where $d_1 = ||\,\mathbf{x} - \mathbf{s}\,||$.

- **Reflection Operation at the Reflector**

  When the ray hits the reflector, it gets reflected. Before we apply reflection equation (4.12) to examine how the shape of the reflected wavefront changes, we should define proper local coordinate systems (or parameterizations) for the incident wavefront, the reflecting surface and the reflected wavefront.

  Suppose $\mathbf{n}^{(i)}$, $\mathbf{n}^{(s)}$ and $\mathbf{n}^{(r)}$ are the normals to the incident wavefront, the reflector and the reflected wavefront, then the unit vector

$$\mathbf{u} = \frac{\mathbf{n}^{(i)} \times \mathbf{n}^{(s)}}{||\,\mathbf{n}^{(i)} \times \mathbf{n}^{(s)}\,||} \tag{4.14}$$

  represents a direction tangent to incident wavefront, reflector surface, and reflected

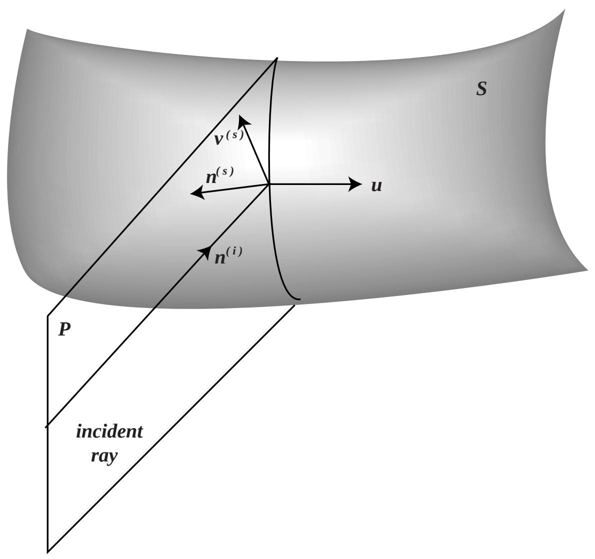


Figure 4.3: *The* $\mathbf{u}$*,* $\mathbf{v}$*,* $\mathbf{n}$ *local coordinate system.* $\mathbf{n}^{(s)}$ *is the unit normal vector to the reflecting surface* $\mathbf{S}$*,* $\mathbf{u}$ *is a unit vector perpendicular to* $\mathbf{n}^{(s)}$ *and the incident ray* $\mathbf{n}^{(i)}$ *and therefore perpendicular to* $\mathbf{M}$*, the plane of incidence.* $\mathbf{v}^{(s)}$ *is the unit vector perpendicular to* $\mathbf{u}$ *and* $\mathbf{n}^{(s)}$*.*

wavefront. See Figure 4.3. By defining

$$\begin{aligned} \mathbf{v}^{(i)} &= \mathbf{n}^{(i)} \times \mathbf{u} \\ \mathbf{v}^{(s)} &= \mathbf{n}^{(s)} \times \mathbf{u} \\ \mathbf{v}^{(r)} &= \mathbf{n}^{(r)} \times \mathbf{u}, \end{aligned} \tag{4.15}$$

we have three orthogonal unit vectors associated with each surface, that is,
incident wavefront ($\mathbf{n}^{(i)}$, $\mathbf{u}$, $\mathbf{v}^{(i)}$),
reflecting surface ($\mathbf{n}^{(s)}$, $\mathbf{u}$, $\mathbf{v}^{(s)}$),
reflected wavefront ($\mathbf{n}^{(r)}$, $\mathbf{u}$, $\mathbf{v}^{(r)}$),
which forms a natural local coordinate systems (or parameterization) for these three surfaces. Now we have two parameterization for each surface or wavefront, one is corresponding to the principal directions (ξ, η), the other is corresponding to the orthogonal directions ($\mathbf{u}$, $\mathbf{v}$) we just choose.

From equation (4.12), the reflected wavefront can be described as:

$$\begin{aligned}
\kappa_u^{(r)} &= \kappa_u^{(i)} - 2\cos\alpha\kappa_u^{(s)} &&= -\frac{1}{d_1} - 2\cos\alpha\kappa_u^{(s)} \\
\kappa_v^{(r)} &= \kappa_v^{(i)} - \frac{2}{\cos\alpha}\kappa_v^{(s)} &&= -\frac{1}{d_1} - \frac{2}{\cos\alpha}\kappa_v^{(s)} \\
\kappa_{uv}^{(r)} &= -\kappa_{uv}^{(i)} + 2\kappa_{uv}^{(s)} &&= 2\kappa_{uv}^{s},
\end{aligned} \tag{4.16}$$

where α is the incident angle, defined as

$$\cos\alpha = \mathbf{n}^{(i)} \cdot \mathbf{n}^{(s)}.$$

From Lemma 1 and Lemma 2, the surface curvatures $\kappa_u^{(s)}$, $\kappa_v^{(s)}$ and torsion $\kappa_{uv}^{(s)}$ can be computed as

$$\begin{aligned}
\kappa_u^{(s)} &= \mathbf{u}^{(s)} \cdot \mathbf{S}\mathbf{u}^{(s)} \\
\kappa_v^{(s)} &= \mathbf{v}^{(s)} \cdot \mathbf{S}\mathbf{v}^{(s)} \\
\kappa_{uv}^{(s)} &= \mathbf{u}^{(s)} \cdot \mathbf{S}\mathbf{v}^{(s)},
\end{aligned} \tag{4.17}$$

and the shape operator matrix $\mathbf{S}$ for an implicit surface has been derived in equation (4.4).

For a general reflection operation of wavefront tracing, the principal curvatures are known for incident wavefront from the previous operation. In order to apply the reflection equation, we should calculate the normal curvatures along $\mathbf{u}$ and $\mathbf{v}$ directions $(\kappa_u^{(i)}, \kappa_v^{(i)})$ from the principal curvatures $(\kappa_\xi^{(i)}, \kappa_\eta^{(i)})$ of the incident wavefront with Euler formula. However, this step is not necessary for a spherical incident wavefront here since sphere has no principal directions.

- **Principal Curvatures of Reflected Wavefront**

  Given two normal curvatures $\kappa_u^{(r)}$ and $\kappa_v^{(r)}$ of the reflected wavefront, the principal directions and the principal curvatures can be determined from Inverse Euler Formula (4.7),

$$\begin{aligned}\tan 2\theta &= 2\kappa_{uv}^{(r)}/(\kappa_u^{(r)} - \kappa_v^{(r)}) \\ \kappa_\xi^{(r)} &= \kappa_u^{(r)}\cos^2\theta + 2\kappa_{uv}^{(r)}\cos\theta\sin\theta + \kappa_v^{(r)}\sin^2\theta \\ \kappa_\eta^{(r)} &= \kappa_u^{(r)}\sin^2\theta - 2\kappa_{uv}^{(r)}\cos\theta\sin\theta + \kappa_v^{(r)}\cos^2\theta\end{aligned}$$

where θ is the angle from the direction $\mathbf{u}$ to the principal direction ξ in a right-handed orientation. From equation (4.16), we have

$$\begin{aligned}\tan 2\theta &= \frac{2\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)} - \cos^2\alpha\kappa_u^{(s)}} \\ \kappa_\xi^{(r)} &= \left(-\frac{1}{d_1} - 2\cos\alpha\kappa_u^{(s)}\right)\cos^2\theta + 2\kappa_{uv}^{(s)}\sin 2\theta + \left(-\frac{1}{d_1} - \frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\sin^2\theta \\ \kappa_\eta^{(r)} &= \left(-\frac{1}{d_1} - 2\cos\alpha\kappa_u^{(s)}\right)\sin^2\theta - 2\kappa_{uv}^{(s)}\sin 2\theta + \left(-\frac{1}{d_1} - \frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\cos^2\theta.\end{aligned} \tag{4.18}$$

- **Final Transfer to the Plane**

  Finally, the ray reaches the plane point $\mathbf{p}$ through another straight line, the principal directions are not changed. From transfer equation (4.10), we get the principal curvatures of the wavefront at the point $\mathbf{p}$:

$$\begin{aligned}\kappa_\xi &= \frac{\kappa_\xi^{(r)}}{1 - d_2\kappa_\xi^{(r)}} \\ \kappa_\eta &= \frac{\kappa_\eta^{(r)}}{1 - d_2\kappa_\eta^{(r)}},\end{aligned} \tag{4.19}$$

  where $d_2 = ||\,\mathbf{x} - \mathbf{p}\,||$.

From intensity law, the irradiance at the plane point $\mathbf{p}$ due to caustics is proportional to the Gaussian Curvature of the wavefront at $\mathbf{p}$, therefore, by assuming the intensity at the point light source is unit, we get the caustic irradiance function on the plane:

$$\phi(\mathbf{p}) = |\kappa_\xi\kappa_\eta|\,. \tag{4.20}$$

Since the wavefront arriving at a plane point **p** can be converging or diverging, which implies that the Gaussian curvature may be negative, we take its absolute value on the right hand side to make the caustic irradiance value meaningful.

4.2.3 Caustic Irradiance Gradient

By keeping track of the Gaussian curvature in the wavefront tracing, we have shown the formula for the caustic irradiance function ϕ on a plane. *Caustic irradiance gradient*, the derivative of ϕ with respect to the plane point **p**, indicates the direction in which the caustic irradiance changes rapidly, which we will build upon in the next section to directly trace the caustic contour. The important property of the caustic irradiance gradient $\nabla\phi$ is its connection with the path Jacobian **J** at the reflection point **x**, which is suggested by the dependence of the caustic irradiance $\phi(\mathbf{p})$ on the caustic path associated. As a one-bounce reflection path, the closed-form expression for **J** obtained in Section 2.3 can be directly applied to the caustic path.

Generally, a function of a form like ϕ is not differentiable. However, when we restrict the domain of the caustic irradiance function to the points lying in the same caustic contour or nearby, the wavefronts reaching these points should be similar and thus their Gaussian curvatures are of the same sign, then we will get a differentiable irradiance function $\phi = \kappa_\xi\kappa_\eta$ (or $\phi = -\kappa_\xi\kappa_\eta$) restricted along caustic contours. Next, we derive the caustic irradiance gradient under an assumption of positive Gaussian curvature.

Differentiating equation (4.20) with respect to the plane point **p**, we get

$$\begin{aligned}
\nabla\phi &= \nabla(\kappa_\xi\kappa_\eta) \\
&= \kappa_\eta\nabla\kappa_\xi + \kappa_\xi\nabla\kappa_\eta \\
&= \kappa_\eta\nabla\frac{\kappa_\xi^{(r)}}{1-d_2\kappa_\xi^{(r)}} + \kappa_\xi\nabla\frac{\kappa_\eta^{(r)}}{1-d_2\kappa_\eta^{(r)}} \\
&= \frac{\kappa_\eta}{(1-d_2\kappa_\xi^{(r)})^2}(\nabla\kappa_\xi^{(r)} + (\kappa_\xi^{(r)})^2\nabla d_2) + \frac{\kappa_\xi}{(1-d_2\kappa_\eta^{(r)})^2}(\nabla\kappa_\eta^{(r)} + (\kappa_\eta^{(r)})^2\nabla d_2) \\
&= \frac{\kappa_\eta}{(1-d_2\kappa_\xi^{(r)})^2}\nabla\kappa_\xi^{(r)} + \frac{\kappa_\xi}{(1-d_2\kappa_\eta^{(r)})^2}\nabla\kappa_\eta^{(r)} + (\kappa_\eta\kappa_\xi^2 + \kappa_\xi\kappa_\eta^2)\nabla d_2,
\end{aligned}$$

where ∇d_2 can be expressed with path Jacobian $\mathbf{J}$ as

$$\nabla d_2 \quad = \quad \nabla(||\,\mathbf{x}-\mathbf{p}\,||) = \frac{(\mathbf{x}-\mathbf{p})^{\mathrm{T}}}{||\,\mathbf{x}-\mathbf{p}\,||}(\mathbf{J}-\mathbf{I}) = \mathbf{n}^{(r)\mathrm{T}}(\mathbf{I}-\mathbf{J}).$$

From equation (4.18), $\nabla\kappa_\xi^{(r)}$ and $\nabla\kappa_\eta^{(r)}$ are expanded as:

$$\begin{aligned}
\nabla\kappa_\xi^{(r)} &= \nabla\left[\left(-\frac{1}{d_1}-2\cos\alpha\kappa_u^{(s)}\right)\cos^2\theta\right]+2\nabla\left[\kappa_{uv}^{(s)}\sin 2\theta\right]+\nabla\left[\left(-\frac{1}{d_1}-\frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\sin^2\theta\right] \\
&= \cos^2\theta\left[\nabla(-1/d_1)-2\kappa_u^{(s)}\nabla\cos\alpha-2\cos\alpha\nabla\kappa_u^{(s)}\right]-\left(-\frac{1}{d_1}-2\cos\alpha\kappa_u^{(s)}\right)\sin 2\theta\nabla\theta+ \\
&\quad \sin^2\theta\left[\nabla(-1/d_1)+2\kappa_v^{(s)}\frac{\nabla\cos\alpha}{\cos^2\alpha}-(2/\cos\alpha)\nabla\kappa_v^{(s)}\right]+\left(-\frac{1}{d_1}-\frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\sin 2\theta\nabla\theta+ \\
&\quad 4\kappa_{uv}^{(s)}\cos 2\theta\nabla\theta+2\sin 2\theta\nabla\kappa_{uv}^{(s)} \\
&= \frac{\nabla d_1}{d_1^2}-2\left(\kappa_u^{(s)}\cos^2\theta-\kappa_v^{(s)}\frac{\sin^2\theta}{\cos^2\alpha}\right)\nabla\cos\alpha+E\nabla\theta \\
&\quad -2\cos\alpha\cos^2\theta\nabla\kappa_u^{(s)}-2\frac{\sin^2\theta}{\cos\alpha}\nabla\kappa_v^{(s)}+2\sin 2\theta\nabla\kappa_{uv}^{(s)},
\end{aligned}$$

$$\begin{aligned}
\nabla\kappa_\eta^{(r)} &= \nabla\left[\left(-\frac{1}{d_1}-2\cos\alpha\kappa_u^{(s)}\right)\sin^2\theta\right]-2\nabla\left[\kappa_{uv}^{(s)}\sin 2\theta\right]+\nabla\left[\left(-\frac{1}{d_1}-\frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\cos^2\theta\right] \\
&= \sin^2\theta\left[\nabla(-1/d_1)-2\kappa_u^{(s)}\nabla\cos\alpha-2\cos\alpha\nabla\kappa_u^{(s)}\right]+\left(-\frac{1}{d_1}-2\cos\alpha\kappa_u^{(s)}\right)\sin 2\theta\nabla\theta+ \\
&\quad \cos^2\theta\left[\nabla(-1/d_1)+2\kappa_v^{(s)}\frac{\nabla\cos\alpha}{\cos^2\alpha}-(2/\cos\alpha)\nabla\kappa_v^{(s)}\right]-\left(-\frac{1}{d_1}-\frac{2}{\cos\alpha}\kappa_v^{(s)}\right)\sin 2\theta\nabla\theta- \\
&\quad 4\kappa_{uv}^{(s)}\cos 2\theta\nabla\theta-2\sin 2\theta\nabla\kappa_{uv}^{(s)} \\
&= \frac{\nabla d_1}{d_1^2}-2\left(\kappa_u^{(s)}\sin^2\theta-\kappa_v^{(s)}\frac{\cos^2\theta}{\cos^2\alpha}\right)\nabla\cos\alpha-E\nabla\theta \\
&\quad -2\cos\alpha\sin^2\theta\nabla\kappa_u^{(s)}-2\frac{\cos^2\theta}{\cos\alpha}\nabla\kappa_v^{(s)}-2\sin 2\theta\nabla\kappa_{uv}^{(s)}.
\end{aligned}$$

where E is the common coefficient for $\nabla\kappa_\xi^{(r)}$ and $\nabla\kappa_\eta^{(r)}$,

$$E = 4\kappa_{uv}^{(s)}\cos 2\theta+2\kappa_u^{(s)}\cos\alpha\sin 2\theta-2\kappa_v^{(s)}\frac{\sin 2\theta}{\cos\alpha}.$$

To calculate $\nabla\kappa_\xi^{(r)}$ and $\nabla\kappa_\eta^{(r)}$, we first compute the derivatives of d_1, $\cos\alpha$ and θ. Similar to d_2, we have

$$\nabla d_1 = \nabla(||\,\mathbf{x}-\mathbf{s}\,||) = \frac{(\mathbf{x}-\mathbf{s})^{\mathrm{T}}}{||\,\mathbf{x}-\mathbf{s}\,||}\mathbf{J} = \mathbf{n}^{(i)\mathrm{T}}\mathbf{J}.$$

In terms of the projection matrix $\mathbf{P}(\mathbf{w})$ defined in equation (4.2), the gradient of surface normals can be expressed as

$$\begin{aligned}\nabla\mathbf{n}^{(i)} &= \nabla\left(\frac{\mathbf{x}-\mathbf{s}}{||\,\mathbf{x}-\mathbf{s}\,||}\right)\\ &= \frac{||\,\mathbf{x}-\mathbf{s}\,||\,\mathbf{J} - \frac{(\mathbf{x}-\mathbf{s})(\mathbf{x}-\mathbf{s})^{\mathrm{T}}}{||\,\mathbf{x}-\mathbf{s}\,||}\mathbf{J}}{(||\,\mathbf{x}-\mathbf{s}\,||)^2}\\ &= \left[\frac{\mathbf{P}(\mathbf{x}-\mathbf{s})}{||\,\mathbf{x}-\mathbf{s}\,||}\right]\mathbf{J},\end{aligned}$$

$$\nabla\mathbf{n}^{(s)} = \frac{d\mathbf{n}^{(s)}}{d\mathbf{x}}\cdot\frac{d\mathbf{x}}{d\mathbf{p}} = -\mathbf{S}\mathbf{J}.$$

Then,

$$\nabla\cos\alpha = \nabla(\mathbf{n}^{(i)}\cdot\mathbf{n}^{(s)}) = (\mathbf{n}^{(s)})^{\mathrm{T}}\nabla\mathbf{n}^{(i)} + (\mathbf{n}^{(i)})^{\mathrm{T}}\nabla\mathbf{n}^{(s)}.$$

Differentiating the expression for $\tan\theta$ shown in equation (4.18) with respect to $\mathbf{p}$,

$$\frac{\nabla\theta}{\cos^2 2\theta} = \nabla\left(\frac{\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha}\right).$$

Thus, $\nabla\theta$ can be solved as

$$\begin{aligned}\nabla\theta &= \cos^2 2\theta\nabla\left(\frac{\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha}\right)\\ &= \frac{1}{1+\tan^2 2\theta}\nabla\left(\frac{\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha}\right)\\ &= \frac{1}{1+\left(\frac{2\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)}-\kappa_u^{(s)}\cos^2\alpha}\right)^2}\nabla\left(\frac{\cos\alpha\kappa_{uv}^{(s)}}{\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha}\right)\\ &= \frac{(\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha)\nabla(\cos\alpha\kappa_{uv}^{(s)}) - \cos\alpha\kappa_{uv}^{(s)}\nabla(\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha)}{(\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha)^2 + 4(\cos\alpha\kappa_{uv}^{(s)})^2},\end{aligned}$$

where

$$\begin{aligned}\nabla(\cos\alpha\kappa_{uv}^{(s)}) &= \kappa_{uv}^{(s)}\nabla\cos\alpha + \cos\alpha\nabla\kappa_{uv}^{(s)},\\ \nabla(\kappa_v^{(s)} - \kappa_u^{(s)}\cos^2\alpha) &= \nabla\kappa_v^{(s)} - 2\cos\alpha\kappa_u^{(s)}\nabla\cos\alpha - \cos^2\alpha\nabla\kappa_u^{(s)}.\end{aligned}$$

Now, it is clear that the expressions for $\nabla\kappa_\xi^{(r)}$ and $\nabla\kappa_\eta^{(r)}$ finally depends on the gradients of surface curvatures: $\nabla\kappa_u^{(s)}$, $\nabla\kappa_v^{(s)}$ and $\nabla\kappa_{uv}^{(s)}$. Since $\kappa_u^{(s)}$,$\kappa_v^{(s)}$ and $\kappa_{uv}^{(s)}$ involve multiplication with the shape operator matrix $\mathbf{S}$, it is more convenient to compute their gradients with Cartesian tensor notation. Thus, surface curvatures in equation (4.17) can be rewritten as

$$\begin{aligned}\kappa_u^{(s)} &= \mathbf{u}\cdot(\mathbf{Su}) &= \textstyle\sum_{i,j=1}^3 u_i S_{ij} u_j,\\ \kappa_v^{(s)} &= \mathbf{v}\cdot(\mathbf{Sv}) &= \textstyle\sum_{i,j=1}^3 v_i S_{ij} v_j,\\ \kappa_{uv}^{(s)} &= \mathbf{u}\cdot(\mathbf{Sv}) &= \textstyle\sum_{i,j=1}^3 u_i S_{ij} v_j.\end{aligned}$$

where $\mathbf{u}$ and $\mathbf{v}$ refer to the two orthogonal directions defined for the reflecting surface in equation (4.14) and (4.15). Without loss of generalization, we derive the mth component of the gradient vector $\nabla\kappa_{uv}^{(s)}$,

$$\begin{aligned}[\nabla\kappa_{uv}^{(s)}]_m &= \frac{\partial(\sum_{i,j=1}^3 u_i S_{ij} v_j)}{\partial p_m}\\ &= \sum_{i,j=1}^{3}(u_{i,m}S_{ij}v_j + u_i S_{ij,m} v_j + u_i S_{ij} v_{j,m})\\ &= \sum_{i,j=1}^{3}((\nabla\mathbf{u})_{mi}S_{ij}v_j + u_i(\nabla\mathbf{S})_{mij}v_j + u_i S_{ij}(\nabla\mathbf{v})_{mj})\\ &= (\nabla_m\mathbf{u})\cdot(\mathbf{Sv}) + \mathbf{u}\cdot((\nabla_m\mathbf{S})\mathbf{v}) + \mathbf{u}\cdot(\mathbf{S}(\nabla_m\mathbf{v})).\end{aligned}$$

In the above equation, what we want to compute on the left is a component of a gradient vector, that is, a scalar. Correspondingly, by considering the gradient of a direction vector as a matrix, the first term on the right is an inner product of a row vector and a column vector, so as the last term. Taking the mth layer of the three dimensional array $\nabla\mathbf{S}$ as a matrix, the second term is also an inner product of two vectors. Therefore, the right hand side represents the summation of three inner products, that is , a scalar.

Similarly, we can get

$$[\nabla \kappa_u^{(s)}]_m = (\nabla_m \mathbf{u}) \cdot (\mathbf{S}\mathbf{u}) + \mathbf{u} \cdot ((\nabla_m \mathbf{S})\mathbf{u}) + \mathbf{u} \cdot (\mathbf{S}(\nabla_m \mathbf{u}),$$

$$[\nabla \kappa_v^{(s)}]_m = (\nabla_m \mathbf{v}) \cdot (\mathbf{S}\mathbf{v}) + \mathbf{v} \cdot ((\nabla_m \mathbf{S})\mathbf{v}) + \mathbf{v} \cdot (\mathbf{S}(\nabla_m \mathbf{v})).$$

By defining $\mathbf{w} = \mathbf{n}^{(i)} \times \mathbf{n}^{(s)}$, $\nabla \mathbf{u}$ and $\nabla \mathbf{v}$ can be calculated from equation (4.14) and (4.15),

$$\begin{aligned}
\nabla \mathbf{u} &= \nabla \left(\frac{\mathbf{w}}{||\,\mathbf{w}\,||} \right) \\
&= \frac{\mathbf{P}(\mathbf{w})}{||\,\mathbf{w}\,||} \nabla \mathbf{w} \\
&= \frac{\mathbf{P}(\mathbf{w})}{||\,\mathbf{w}\,||} \left[\mathbf{Q}(\mathbf{n}^{(i)}) \nabla \mathbf{n}^{(s)} - \mathbf{Q}(\mathbf{n}^{(s)}) \nabla \mathbf{n}^{(i)} \right], \\
\nabla \mathbf{v} &= \nabla (\mathbf{n}^{(s)} \times \mathbf{u}), \\
&= \mathbf{Q}(\mathbf{n}^{(s)}) \nabla \mathbf{u} - \mathbf{Q}(\mathbf{u}) \nabla \mathbf{n}^{(s)}
\end{aligned}$$

where $\mathbf{Q}(\mathbf{p})$ is the skew matrix for a vector $\mathbf{p}$, defined as

$$\mathbf{Q}(\mathbf{p}) = \begin{bmatrix} 0 & p_z & -p_y \\ -p_z & 0 & p_x \\ p_y & -p_x & 0 \end{bmatrix}$$

and the gradient rule for a cross product is

$$\nabla(F \times G) = \mathbf{Q}(F)\nabla(G) - \mathbf{Q}(G)\nabla(F).$$

Each element of the shape operator matrix is evaluated at the surface point $\mathbf{x}$, which is determined by the plane point $\mathbf{p}$ for a fixed light source $\mathbf{s}$, so it can be taken as a function of $\mathbf{p}$. With the chain rule and the formula for path Jacobian $\mathbf{J}$, we can compute the gradient of $\mathbf{S}$ with respect to $\mathbf{p}$ by computing the gradient vector of each element

S_{ij}:

$$\begin{aligned}(\nabla \mathbf{S})_{ij} &= \frac{\partial S_{ij}}{\partial \mathbf{p}} \\ &= \frac{\partial S_{ij}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{p}} \\ &= \frac{\partial S_{ij}}{\partial \mathbf{x}} \mathbf{J}. \quad i, j = 1, 2, 3\end{aligned}$$

Although the formulae for caustic irradiance and its gradient are very complicated for a general implicit surface, it becomes much simpler for some reflecting surfaces with special geometric properties. For example, the fact that a spherical reflector keeps constant normal curvature along any direction and has no principal direction simplifies the computation a lot. See Appendix A.1 and Appendix A.2.

4.2.4 Direct Computation of Caustic Contours

Any techniques for solving first-order ordinary differential equations can be applied to solve the caustic contour differential equation shown in equation (4.13). We use the classical *predictor-corrector method.* It consists of two parts: *predictor* and *corrector.* The simplest predictor begins by taking a small step from a given starting point in the plane in a direction perpendicular to the gradient and gets a new *predicted* point, which is likely near the iso-contour, although not exactly on it. Of course, multi-step methods can be chosen to make closer prediction. For example, Milne's predictor extrapolates the new point from the three most recent gradients and function values using a parabola. That is, when the project and rotation matrix $\mathbf{P}$ is fixed, Milne's predictor is given by

$$\mathbf{r}_{k+1}^0 = \mathbf{r}_{k-3} + \frac{4h}{3} \mathbf{P}(2g_{k-2} - g_{k-1} + 2g_k),$$

where g_k denotes the gradient at the point $\mathbf{r}_k$ and h is the chosen step size. Given a predicted point from the predictor, a corrector is invoked to locate a nearby point within a specified tolerance distance to the curve. Because the contour is the zero set of the function $\phi(\mathbf{r}) - \phi_0$, ϕ_0 is the constant irradiance value, the correction can be performed very efficiently using the gradient descent method. Beginning with the predicted point

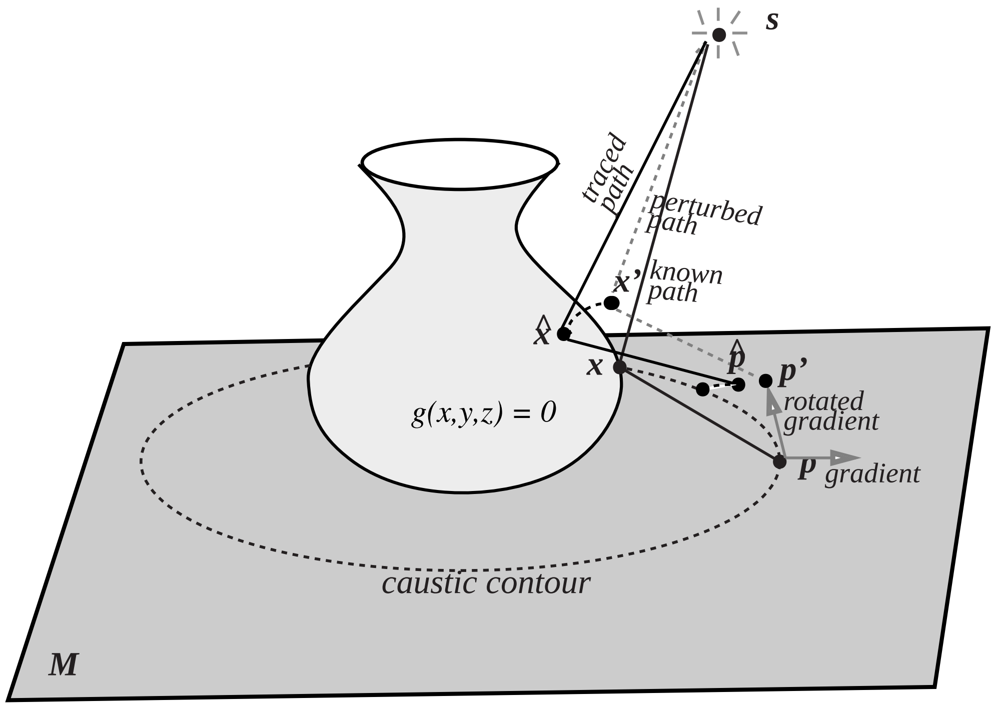


Figure 4.4: *Modified predictor-corrector method, it shows one predictor step,* $\hat{\mathbf{p}}$ *is the returned predicted point from* $\mathbf{p}$*, another similar corrector step will correct* $\hat{\mathbf{p}}$ *to the contour.*

$\mathbf{r}_k^0$, the gradient descent method generates the sequence $\mathbf{r}_k^1, \mathbf{r}_k^2, \ldots$ by

$$\mathbf{r}_k^{i+1} = \mathbf{r}_k^i + \left[\phi_0 - \phi(\mathbf{r}_k^i)\right] \frac{\nabla\phi^{\mathrm{T}}(\mathbf{r}_k^i)}{||\,\nabla\phi^{\mathrm{T}}(\mathbf{r}_k^i)\,||},$$

which converges quadratically to a point on the contour curve within the specified tolerance ϵ.

The algorithm stated above is the standard curve following algorithm used by Arvo [9]. However, another postprocessing step is required for our predictor and corrector. To keep this predictor-corrector iteration going on, we should be able to compute the caustic irradiance and its gradient at each predicted or corrected point. As seen from the derivations in Section 4.2.2 and Section 4.2.3, the caustic irradiance $\phi(\mathbf{p})$ and its gradient $\nabla\phi(\mathbf{p})$ are dependent on the caustic path from $\mathbf{s}$ to $\mathbf{p}$, which will change accordingly with respect to the movement of $\mathbf{p}$. Whenever we get a new predicted point (or a corrected point) $\mathbf{p}'$, a new caustic path from $\mathbf{s}$ to $\mathbf{p}'$ must be determined. Mitchell and Hanrahan [59] employed a very expensive numerical procedure to find the caustic path for each plane point Notice that the caustic paths for the neighboring caustic contour points are highly coherent, we can estimate the new caustic path by perturbing the previous one instead of determining it from scratch. That is, the perturbation formula $\mathbf{x}' = \mathbf{x} + \mathbf{J}(\mathbf{p}' - \mathbf{p})$ using the path Jacobian provides a first order approximation of the new caustic path from $\mathbf{s}$ to $\mathbf{p}'$. However, because of linear approximation, $\mathbf{x}'$ doesn't exactly lie on the reflecting surface, the perturbed path from $\mathbf{s}$ to $\mathbf{p}'$ via $\mathbf{x}'$, illustrated with dotted lines in Figure 4.4, does not represent a true path that the ray emitted from the light source $\mathbf{s}$ will take. To adjust this perturbed virtual path to a real one, we modify the predictor and corrector routine with an inner iteration added. In this inner iteration, we first use gradient descent to perturb $\mathbf{x}'$ to the nearby surface point $\hat{\mathbf{x}}$ and then cast a ray from $\mathbf{s}$ to $\hat{\mathbf{x}}$ to get the intersection point $\hat{\mathbf{p}}$ on the plane underneath (denoted by *TraceRay* in the pseudo code), which is returned as the final predicted or corrected point during a predictor or corrector step. Figure 4.4 illustrates how our postprocessing procedure works in one step of the predictor-corrector loop. This modified curve following algorithm can be summarized in the pseudo code below, where *CausticIrradiance* and *CausticIrradianceGradient* are used to compute the caustic irradiance and the caustic irradiance gradient at a plane point. As we mentioned in Section 4.2.3, the implementation and complexity of these two routines are dependent on the particular implicit surface. To enhance the numerical stability, we progressively halves the predictor's step size until the corrector converges.

```
GenerateCausticContour( Point $\mathbf{s}$, Point $\mathbf{p}$, Surface $f$ )
    Find an initial caustic path from $\mathbf{s}$ and $\mathbf{p}$ (open question)
    $\mathbf{x} \leftarrow$ the reflection point between $\mathbf{s}$ and $\mathbf{p}$
    Scalar $\phi_0 \leftarrow$ CausticIrradiance( $\mathbf{p}$, $\mathbf{x}$ )
    Begin a new contour of irradiance $\phi_0$ from the point $\mathbf{p}$
    Point $\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$
    repeat
        Matrix $\mathbf{J} \leftarrow$ PathJacobian($\mathbf{p}$, $\mathbf{s}$, $\mathbf{x}$, $f$)
        Gradient $g \leftarrow$ CausticIrradianceGradient($\mathbf{p}$, $\mathbf{x}$, $\mathbf{J}$)
        Scalar $h \leftarrow$ User supplied step size
        repeat
            $(\hat{\mathbf{p}}, \hat{\mathbf{x}}) \leftarrow$ Predictor($\mathbf{p}$, $\mathbf{x}$, $\mathbf{J}$, $g$, $f$, $h$)
            $(\hat{\mathbf{p}}, \hat{\mathbf{x}}) \leftarrow$ Corrector($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$, $\phi_0$, $f$)
            $h \leftarrow h/2$
        until corrector converges
        $\mathbf{p} \leftarrow \hat{\mathbf{p}}$
        $\mathbf{x} \leftarrow \hat{\mathbf{x}}$
        Add $\mathbf{p}$ to the contour
    until contour closes
```

```
Predictor( **Point** $\mathbf{p}$, **Point** $\mathbf{x}$, **Matrix** $\mathbf{J}$, **Gradient** $g$, **Surface** $f$, **Scalar** $h$ )
    **Gradient** $g' \leftarrow$ Project $g$ to the plane and rotate $-\pi/2$
    **Point** $\mathbf{p}', \hat{\mathbf{p}}, \mathbf{x}', \hat{\mathbf{x}}$
    $\mathbf{p}' \leftarrow \mathbf{p} + hg'/||g'||$
    **Vector** $\Delta\mathbf{p} \leftarrow \mathbf{p}' - \mathbf{p}$
    $\mathbf{x}' \leftarrow \mathbf{x} + \mathbf{J}\Delta\mathbf{p}$
    $\hat{\mathbf{x}} \leftarrow$ ProjectToSurface($\mathbf{x}'$, $f$)
    $\hat{\mathbf{p}} \leftarrow$ TraceRay($\mathbf{s}$, $\hat{\mathbf{x}}$)
    **return** ($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$)

Corrector( **Point** $\mathbf{p}$, **Point** $\mathbf{x}$, **Scalar** $\phi$, **Surface** $f$ )
    **Point** $\mathbf{p}', \hat{\mathbf{p}}, \hat{\mathbf{x}}$
    $\hat{\mathbf{p}} \leftarrow \mathbf{p}$
    $\hat{\mathbf{x}} \leftarrow \mathbf{x}$
    **Scalar** $\hat{\phi} \leftarrow$ CausticIrradiance($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$)
    **while** $\left|\phi - In\hat{t}en\right| > \epsilon$ **do**
      **Matrix** $\mathbf{J} \leftarrow$ PathJacobian($\hat{\mathbf{p}}$, $\mathbf{s}$, $\hat{\mathbf{x}}$, $f$)
      **Gradient** $g \leftarrow$ CausticIrradianceGradient($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$, $\mathbf{J}$)
      $\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} + (\phi - \hat{\phi})g/||g||^2$
      **Vector** $\Delta\mathbf{p} \leftarrow \mathbf{p}' - \hat{\mathbf{p}}$
      $\mathbf{x}' \leftarrow \hat{\mathbf{x}} + \mathbf{J}\Delta\mathbf{p}$
      $\hat{\mathbf{x}} \leftarrow$ ProjectToSurface($\mathbf{x}'$, $f$)
      $\hat{\mathbf{p}} \leftarrow$ TraceRay($\mathbf{s}$, $\hat{\mathbf{x}}$)
      $\hat{\phi} \leftarrow$ CausticIrradiance($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$)
    **end while**
    **return** ($\hat{\mathbf{p}}$, $\hat{\mathbf{x}}$)
```

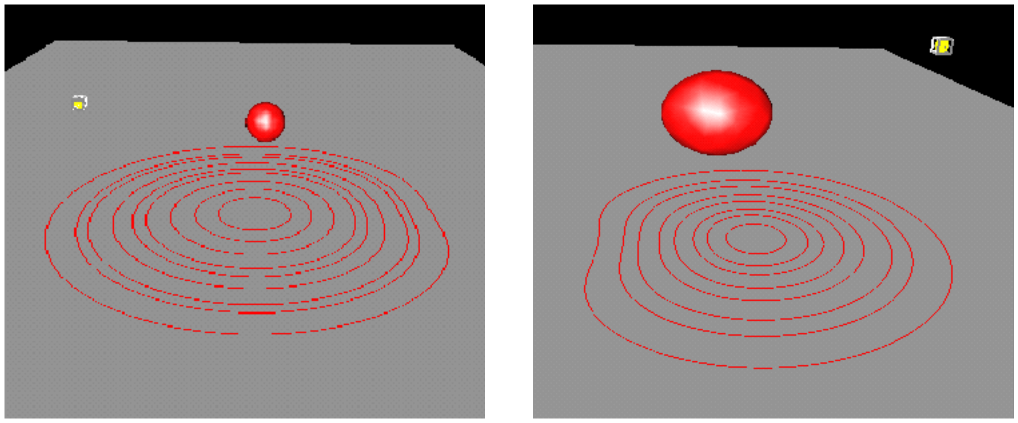


Figure 4.5: *A family of caustic contours due to a point light source. (Left) a sphere reflector, (Right) an ellipsoid reflector*

```
ProjectToSurface(**Point x**, **Surface** $f$ )
    **Point** $\hat{\mathbf{x}} \leftarrow \mathbf{x}$
    **Scalar** $e \leftarrow f(\hat{\mathbf{x}})$
    **while** $|e| > \epsilon$ **do**
        **Gradient** $g \leftarrow \nabla f(\hat{\mathbf{x}})$
        $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} - eg/||g||^2$
        $e \leftarrow f(\hat{\mathbf{x}})$
    **end while**
    **return** $\hat{\mathbf{x}}$
```

4.3 Results and Problems

The predictor-corrector method described in Section 4.2 has been tried to compute caustic contours on a plane for some simple convex reflecting surfaces with well-known implicit definitions, such as sphere and ellipsoid. The use clicks at a point in the plane to start tracing a caustic contour from that point, the step size h and the tolerance ϵ for the corrector are user-supplied parameters. Figure 4.5 shows a family of caustic contours generated for a sphere reflector and an ellipsoid reflector, respectively. Because distinct caustic contours cannot cross, there exists an inherent partial ordering among them.

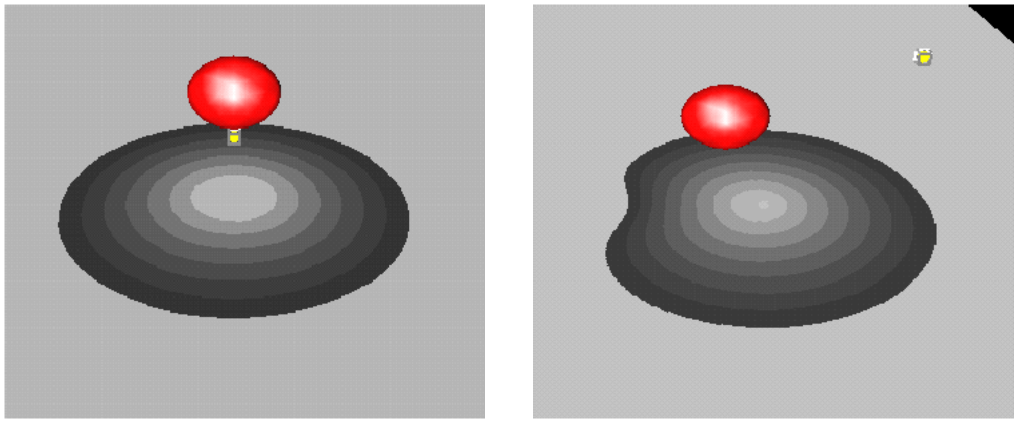


Figure 4.6: *Filled caustic contours for different light positions. Each region is shaded according to the caustic irradiance value of its contour.*

In Figure 4.6, the regions are shaded with their caustic irradiance values from outside inwards according to the partial order.

However, there are some problems associated with our caustic algorithm, which hinders its extension to more complicated surface shapes.

First, the whole contour tracing is based on an initial real caustic path connecting the point light source and the plane point the user picks. How to find a reflection path connecting two arbitrary points is a difficult second category of reflection problem shown in Figure 3.1(b). To make things worse, there may exist more than one reflection paths in the case of a concave surface. The method proposed by Mitchell and Hanrahan [59] provides a general solution to the initial path, but solving the non-linear Lagrange system numerically is very time-consuming. In fact, some approximation methods can be employed for some particular surfaces. For example, for a sphere, it can be proved that three points $\mathbf{s}$,$\mathbf{x}$,$\mathbf{p}$ on a reflection path are coplanar with the sphere center, thus, the problem of finding a reflection point $\mathbf{x}$ on the sphere can be reduced to a 2D problem and solved by a bisection method. In our example of ellipsoid, we propose a perturbation method to find a reflection path. The idea is like this: Given a plane point $\mathbf{p}$ and a point light source $\mathbf{s}$, an initial guess $\mathbf{x}^0$ is chosen on the ellipsoid. If we cast a ray from $\mathbf{s}$ to $\mathbf{x}^0$ and get reflected to $\mathbf{p}'$ on the plane, the reflected ray will normally miss the target $\mathbf{p}$ (i.e, $\mathbf{p}'$ is off $\mathbf{p}$), but we will get a nearby path if the initial guess is reasonable. Then,

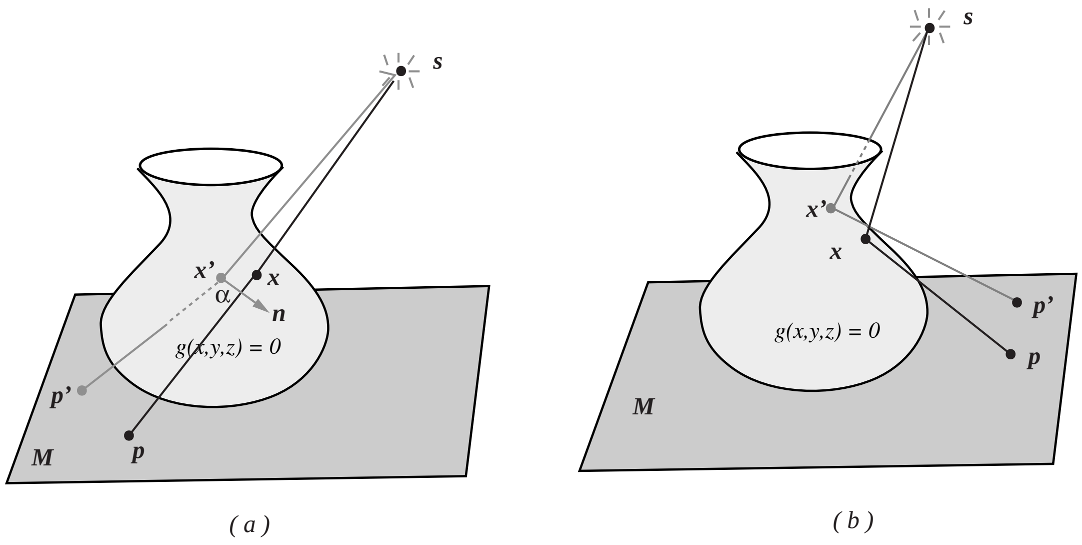


Figure 4.7: *Two cases of perturbing a valid path to an invalid path. (a) The incident angle* α *of the perturbed path is over* 90 *degree at the singular point* $\mathbf{p}$ *with a nearly straight path. (b) A concave surface occlude the perturbed path.*

the desired path from $\mathbf{s}$ to $\mathbf{p}$ can be obtained by applying the perturbation formula with path Jacobian iteratively to the nearby paths. However, whether this iterative method converges or not is determined by the initial guess, while finding a close initial position for some complicated surfaces is very difficult if not impossible.

The second problem happens in some cases where a valid caustic path is perturbed to an invalid one. Figure 4.7 shows two such cases. On the left, before perturbation, the caustic path to $\mathbf{p}$ is nearly tangent to the reflecting surface, that is, the incident angle is close to a right angle. After perturbing such a caustic path by a step h, the incident angle will be over 90 degree, the ray hitting the backface of the surface. We call such contour points *singular points.* On the right, the perturbed path passes through the surface because of the self-occlusion of a concave surface. In both cases, the perturbed path doesn't represent a valid caustic path any more. In our experimentation, whenever this occurs, the step size h is halved, which will cause the corrector diverges or converges very slowly. If the step size is decreased too small, the algorithm appears to get stuck at some point forever without any progress. Figure 4.8 shows a unclosed contour. Further

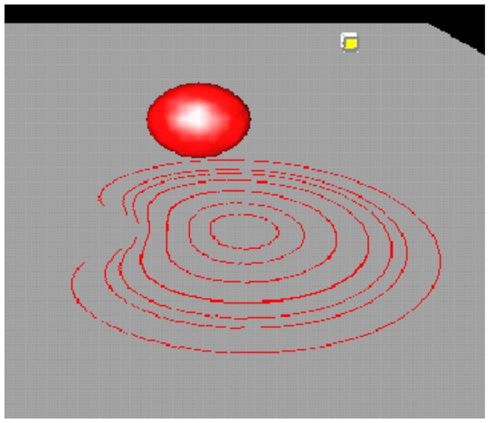


Figure 4.8: *A singular case where our algorithm fails.*

research on the optical properties of singular points will help us to develop more robust curve tracing algorithms, which is still an interesting problem for future work.

Finally, we lack a better way to obtain a real caustic path from a perturbed one. The way of projecting the perturbed point to the surface and then tracing a ray is obviously a very crude approximation. If the surface is very curving, the projection operation will make the new path far away from the predicted point obtained by walking along the tangent direction. Therefore, it may never make use of the gradient information computed at the previous contour point and lead to divergence of the corrector. Although we can employ a second-order perturbation to get a higher accuracy, this problem from path approximation still exists.

Chapter 5

Conclusion

5.1 Summary

This thesis presented new mathematical and computational tools for interactive rendering. By exploiting path coherence, the new tools introduce perturbation theory into incremental rendering and generate families of closely related optical paths by expanding a given path into a high dimensional Taylor series. With respect to changes in the scene or the viewer, the closed-form perturbation formula derived can be used to update a specular path efficiently and accurately, which provides an important step towards interactively viewing and rendering dynamic environments. New rendering algorithms based on perturbation methods are proposed, including rapid approximation of specular reflections and direct computation of caustic contours.

Chapter 2 introduced the concepts of path Jacobian and path Hessian to describe the linear and quadratic approximations of an optical path, presented a closed-form perturbation formula for a general reflection path based on second order Taylor expansion. This expression works for any implicitly-defined reflecting surfaces and can be evaluated by differentiating the implicit function to a certain order.

The expression for path Jacobians was derived from *Fermat's principle* and *Implicit function theorem*, and satisfies a recurrence relation for a multiple-bounce specular path. Based on path Jacobians, the expression for path Hessian were derived with tensor differentiation. In terms of path Jacobians and path Hessians, the perturbation formula was obtained by expanding a given path to a second order Taylor series. These closed-form expressions provide tools to develop efficient incremental rendering algorithms, accounting for some prominent optical effects.

Chapter 3 presented a perturbation method for interactively approximating specular reflections in arbitrary curved surfaces. This new algorithm rapidly approximates reflections of arbitrary points in 3-space by expressing them as perturbations of nearby points with known reflections. After sparse sampling in a preprocessing step, the entire process is fast enough to render specular reflections of tessellated diffuse objects at interactive rates using standard graphics hardware, and the simulated reflection is of very high accuracy.

Chapter 4 described a new approach for directly generating caustic contours on diffuse surfaces. This technique is based on the closed-form expressions for caustic irradiance and caustic irradiance gradient due to a point light source, which are derived from wavefront tracing and the closed-form formula for the path Jacobian of a specular path. Caustic contours on a diffuse surface are computed by solving an ordinary differential equation in terms of caustic irrdiance and caustic irradiance gradient. This algorithm has shown successful examples for simple convex surfaces, and further improvements are required to make it robust and extensible.

5.2 Future Work

Many extensions of the theory of specular path perturbation are possible. A natural extension is to apply the similar perturbation idea to a refraction path or a path mixing reflection and refraction. Since Fermat's principle doesn't place restriction on reflections or refractions, the tools used in deriving path Jacobians and path Hessians for a reflection path should still work for a refraction path with the exception of taking index of refraction into account. With a similar perturbation formula derived for a refraction path, nearly the same algorithm described in Chapter 3 could be applied to real-time simulation of lens effects on a moving object.

The observation that the family of reflection paths represented by a common reflection direction are closely related motivates us to explore the possibility of deriving path Jacobians and/or path Hessians in terms of a reflection direction rather than a particular position along this direction. If an analytical solution were found for the family of reflection paths with the same reflection direction, we could cache precomputed path Jacobians and/or path Hessians instead of computing them on the fly in Chapter 3, which will become an important enhancement toward interactive simulation of higher-level specular reflections.

To precisely evaluate the accuracy of path perturbation, error analysis is required to identify and quantify the bounded-error in perturbation method, which comes from such sources as truncating Taylor series, boundary exceptions, and visibility problems, etc. Another interesting topic is the extension of specular path perturbation to more general parametric surfaces without an implicit definition. For such purpose, numerical approximation can be employed to evaluate our analytical expression within certain error bound.

There are many opportunities to make the approach described in Chapter 3 faster and more robust, such as sampling rays in the object space in a highly non-uniform manner, adaptively selecting an appropriate density; balancing quality and performance by choosing proper tessellation and interpolation strategies; exploring more efficient ways to store and search the sample ray space; pre-caching path Jacobians and/or path Hessians, and so on. Finally, another practical extension would be to combine the perturbation method with environment mapping and the work of Ofek and Rappoport [64]. Reflections of distant static objects could be approximated using a conventional environment map, while accurate reflections of nearby dynamic objects could be computed by perturbation or explosion map.

As mentioned in Section 4.3, our current caustic contour algorithm proposed in Chapter 4 possesses some problems and limitations. Further research on wavefront tracing is required to have a solid understanding of underlying reasons for singular point problem. On the other hand, the self-occlusion problem (that is, the perturbed path is occluded by the reflector itself) poses an interesting topic for perturbation theory. That is, how is a local approximation theory such as perturbation applied to handle a global effect such as occlusion? More sophisticated theory about compromising local optimization and global constraints should be further investigated.

Perturbation methods of this nature can also find many other potential applications in image synthesis. For example, by projecting path Jacobian into the image plane, we can introduce and compute *image Jacobian*, which approximates how pixels moves with respect to changes in the viewer. Then, image differentiation approach could be applied to image warping where specular effects are prominent, as in the work of Lischinski and Rappoport [53]. Furthermore, the benefit of analytical path perturbation over random perturbation may provide a powerful mutation strategy for metropolis light transport [79] to reduce variance and improve convergence rate.

Appendix A

Additional Derivations and Background

A.1 Caustic Irradiance for a Sphere Reflector

For a reflecting sphere of radius r with its center located at point $\mathbf{c}$, its normal curvatures keep constant($1/r$) along any directions and there is no principal direction($\kappa_{uv}^{(s)} = 0$), thus we will have the following equations from wavefront tracing:

Incident Wavefront:

$$\begin{aligned} \kappa_u^{(i)} &= -\frac{1}{d_1} \\ \kappa_v^{(i)} &= -\frac{1}{d_1} \\ \kappa_{uv}^{(i)} &= 0. \end{aligned}$$

Surface Curvature:

$$\begin{aligned} \kappa_u^{(s)} &= \frac{1}{r} \\ \kappa_v^{(s)} &= \frac{1}{r} \\ \kappa_{uv}^{(s)} &= 0. \end{aligned}$$

Reflected Wavefront:

$$\begin{aligned}\kappa_u^{(r)} &= -\frac{1}{d_1} - \frac{2}{r}\cos\alpha \\ \kappa_v^{(r)} &= -\frac{1}{d_1} - \frac{2}{r\cos\alpha} \\ \kappa_{uv}^{(r)} &= 0.\end{aligned}$$

and its principal directions are coincident with direction **u** and **v**,

$$\begin{aligned}\kappa_\xi^{(r)} &= -\frac{1}{d_1} - \frac{2}{r}\cos\alpha \\ \kappa_\eta^{(r)} &= -\frac{1}{d_1} - \frac{2}{r\cos\alpha} \\ \theta &= 0.\end{aligned}$$

Wavefront at p:

$$\begin{aligned}\kappa_\xi &= \frac{\kappa_\xi^{(r)}}{1-d_2\kappa_\xi^{(r)}} \\ \kappa_\eta &= \frac{\kappa_\eta^{(r)}}{1-d_2\kappa_\eta^{(r)}}\end{aligned}$$

Finally, the irradiance function is

$$\phi(\mathbf{p}) = |\kappa_\xi\kappa_\eta|. \tag{A.1}$$

A.2 Caustic Irradiance Gradient for a Sphere Reflector

Assuming a positive Gaussian curvature, we can differentiate the caustic irradiance function $\phi\mathbf{p}$ restricted along a caustic contour with respect to **p**, as described in Section 4.2.3. Here, we only list the major results, showing the simplification from the speciality of a sphere.

$$\nabla\phi = \frac{\kappa_\eta}{(1-d_2\kappa_\xi^{(r)})^2}\nabla\kappa_\xi^{(r)} + \frac{\kappa_\xi}{(1-d_2\kappa_\eta^{(r)})^2}\nabla\kappa_\eta^{(r)} + (\kappa_\eta\kappa_\xi^2 + \kappa_\xi\kappa_\eta^2)\nabla d_2 \tag{A.2}$$

where

$$\begin{aligned}
\nabla \kappa_\xi^{(r)} &= \frac{\nabla d_1}{d_1^2} - \frac{2}{r}\nabla \cos\alpha \\
\nabla \kappa_\eta^{(r)} &= \frac{\nabla d_1}{d_1^2} + \frac{2}{r\cos^2\alpha}\nabla \cos\alpha \\
\nabla d_1 &= \frac{(\mathbf{x}-\mathbf{s})^{\mathrm{T}}}{||\,\mathbf{x}-\mathbf{s}\,||}\mathbf{J} = \mathbf{n}^{(i)\mathrm{T}}\mathbf{J} \\
\nabla d_2 &= \frac{(\mathbf{x}-\mathbf{p})^{\mathrm{T}}}{||\,\mathbf{x}-\mathbf{p}\,||}(\mathbf{J}-\mathbf{I}) = \mathbf{n}^{(r)\mathrm{T}}(\mathbf{I}-\mathbf{J}) \\
\nabla \cos\alpha &= (\mathbf{n}^{(s)})^{\mathrm{T}}\nabla\mathbf{n}^{(i)} + (\mathbf{n}^{(i)})^{\mathrm{T}}\nabla\mathbf{n}^{(s)} \\
\nabla \mathbf{n}^{(i)} &= \left[\frac{\mathbf{P}(\mathbf{x}-\mathbf{s})}{||\,\mathbf{x}-\mathbf{s}\,||}\right]\mathbf{J} \\
\nabla \mathbf{n}^{(s)} &= -\mathbf{S}.
\end{aligned}$$

The path Jacobian **J** has been computed from implicit function theorem in Section 2.3, and the shape operator for a sphere

$$f(x,y,z) = (x-c_1)^2 + (y-c_2)^2 + (z-c_3)^2 = r^2$$

can be derived from equation (4.4). Obviously, the gradient of the shape operator is not necessary in the sphere case.

Bibliography

[1] S. J. Adelson and L. F.Hodges. Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications*, 15:43–52, 1995.

[2] S. J. Adelson and L.F. Hodges. Stereoscopic ray tracing. *The Visual Computer*, 10(3):127–144, 1993.

[3] John Amanatides. Ray tracing with cones. *Computer Graphics*, 18(3):129–135, July 1984.

[4] James Arvo. Backward ray tracing. In *Developments in Ray Tracing, SIGGRAPH '86 Course Notes*, volume 12, August 1986.

[5] James Arvo. The irradiance Jacobian for partially occluded polyhedral sources. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 343–350, July 1994.

[6] James Arvo and David Kirk. Fast ray tracing by ray classification. *Computer Graphics*, 21(4):55–64, July 1987.

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

[8] James Arvo and Kevin Novins. Iso-contour volume rendering. In *1994 ACM/IEEE Symposium on Volume Visualization*, Washington DC, October 1994.

[9] James Arvo, Kenneth Torrance, and Brian Smits. A framework for the analysis of error in global illumination algorithms. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 75–84, July 1994.

[10] A. Aziz and T. Y. Na. *Perturbation Methods in Heat Transfer*. Hemisphere Publishing Corporation, New York, 1984.

[11] Kavita Bala, Julie Dorsey, and Seth Teller. Bounded-error interactive ray tracing. Technical Report LCS TR-748, MIT, 1998.

[12] Rui M. Bastos, António A. de Sousa, and Fernando N. Ferreira. Reconstruction of illumination functions using hermite bicubic interpolation. In *Proceedings of the Fourth Eurographics Workshop on Rendering*, Paris, France, June 1993.

[13] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. *Computer Graphics*, 23(3):325–334, July 1989.

[14] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2):35–42, July 1992.

[15] C. M. Bender and S. A. Orszag. *Advanced Mathematical Methods for Scientists and Engineers*. McGraw-Hill, New York, 1978.

[16] James F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.

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

[18] N. Briere and P. Poulin. Hierarchical view-dependent structures for interactive scene manipulation. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 83–90, August 1996.

[19] J. Chapman, T. W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. In *Proceedings of Graphics Interface'90*, pages 196–204, May 1990.

[20] J. Chapman, T. W. Calvert, and J. Dill. Spatio-temporal coherence in ray tracing. In *Proceedings of Graphics Interface'91*, pages 101–108, June 1991.

[21] S. E. Chen and L. William. View interpolation for image synthesis. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 279–288, August 1993.

[22] S.E. Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. *Computer Graphics*, 24(4):135–144, August 1990.

[23] S.E. Chen, H. Rushmeier, G. Miller, and D. Turner. A progressive multi-pass method for global illumination. *Computer Graphics*, 25(4):165–174, July 1991.

[24] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84, August 1988.

[25] J. D. Cole. *Perturbation Methods in Applied Mathmatics.* Blaisdell, Waltham, Mass., 1968.

[26] Robert L. Cook. Shade trees. *Computer Graphics*, 18(3):137–145, July 1984.

[27] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, July 1984.

[28] George Drettakis and Eugene L. Fiume. Concrete computation of global illumination using structured sampling. In *Proceedings of the Third Eurographics Workshop on Rendering,* Bristol, United Kingdom, 1992.

[29] George Drettakis and François X. Sillion. Interactive update of global illumination using a line-space hierarchy. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 57–64, August 1997.

[30] M. Van Dyke. *Perturbation Methods in Fluid Mechanics.* Academic Press, New York, 1964.

[31] D. Forsyth, C. Yang, and K. Teo. Efficient radiosity in dynamic environments. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, June 1994.

[32] Akira Fujimoto and Takayuki Tanaka. ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[33] David George, F. Sillion, and D. Greenberg. Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4):26–34, July 1990.

[34] Andrew Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[35] Andrew Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, March 1988.

[36] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, July 1984.

[37] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 43–54, August 1996.

[38] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, July 1991.

[39] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(4):145–154, July 1990.

[40] Paul S. Heckbert. *Simulating Global Illumination Using Adaptive Meshing*. Ph.D. thesis, University of California, Berkeley, June 1991.

[41] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. *Computer Graphics*, 18(3):119–127, July 1984.

[42] E. J. Hinch. *Perturbation Methods*. Cambridge University Press, New York, 1991.

[43] John A. Kneisly II. Local curvature of wavefronts in an optical system. *Journal of the Optical Society of America*, 54(2):229–235, February 1964.

[44] Henrik Wann Jensen. Rendering caustics on non-lambertian surfaces. In *Proceedings of Graphics Interface '96*, pages 116–121, Toronto, 1996.

[45] Henrik Wann Jensen and Niel Jørgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computer & Graphics*, 19(2):215–224, March 1995.

[46] David A. Jevans. Object space temporal coherence for ray tracing. In *Proceedings of Graphics Interface '92*, pages 176–183, May 1992.

[47] S. Badt Jr. Two algorithms taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132, 1988.

[48] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986.

[49] Sing Bing Kang. A survey of image-based rendering techniques. Technical Report CRL 97/4, Cambridge Research Laboratory, August 1997.

[50] Erwin Kreyszig. *Differential Geometry*. Dover Publications, New York, 1991.

[51] Marc Levoy and Pat Hanrahan. Light field rendering. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 31–42, August 1996.

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

[53] 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.

[54] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.

[55] W. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, 1995.

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

[57] Leonard McMillan. *An Image-based Approach to Three-Dimensional Computer Graphics*. Ph.D. thesis, UNC, 1997.

[58] Leonard McMillan and Gary Bishop. Head-tracked stero display using image warping. In *IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, SPIE Proceedings #2409A, San Jose, February 1995.

[59] Don Mitchell and Pat Hanrahan. Illumination from curved reflectors. *Computer Graphics*, 26(2):283–291, July 1992.

[60] Stefan Müller and Frank Schöffel. Fast radiosity repropagation for interactive virtual environments using a shadow-form-factor list. In *Proceedings of the Fifth Eurographics Workshop on Rendering*, June 1994.

[61] K. Murakami and K. Hirota. Incremental ray tracing. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics, Conference Proceedings*, pages 15–29, Rennes, France, June 1990.

[62] Ail Hasan Nayfeh. *Perturbation Methods*. John Wiley & Sons, New York, 1973.

[63] Jeffry S. Nimeroff, Julie Dorsey, and H. Rushmeier. A framework for global illumination in animated environments. In *The Sixth Annual Eurographics Workshop on Rendering*, pages 223–236, June 1995.

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

[65] Barrett O'Neill. *Elementary Differential Geometry*. Academic Press, New York, 1966.

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

[67] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Transactions on Graphics*, 9(1):1–27, January 1990.

[68] David Salesin, Dani Lischninski, and Tony DeRose. Reconstructing illumination functions with selected discontinuities. In *Proceedings of the Third Eurographics Workshop on Rendering,* Bristol, United Kingdom, pages 99–112, May 1992.

[69] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers and Graphics*, 13(4):445–460, 1989.

[70] Lee A. Segel and G. H. Handelman. *Mathematics Applied to Continuum Mechanics.* Macmillan Publishing Company, New York, 1977.

[71] Carlo H. Séquin and Eliot K. Smyrl. Parameterized ray tracing. *Computer Graphics*, 23(3):307–314, July 1989.

[72] Erin S. Shaw. Hierarchical radiosity for dynamic environments. Master's thesis, Cornell University, August 1994.

[73] F. Sillion and C. Puech, editors. *Radiosity and Global Illumination.* Morgan Kaufmann, San Francisco, 1994.

[74] François Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(4), August 1989.

[75] O. N. Stavroudis. *The Optics of Rays, Wavefronts, and Caustics.* Academic Press, New York, 1972.

[76] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 251–258, August 1997.

[77] Richard Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, 1996.

[78] Seth Teller, Kavita Bala, and Julie Dorsey. Conservative radiance interpolants for ray tracing. In *Seventh Eurographics Workshop on Rendering*, Porto, Portugal, June 1996.

[79] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, pages 65–76, August 1997.

[80] Christophe Vedel. Improved storage and reconstruction of light intensities on surfaces. In *Proceedings of the Third Eurographics Workshop on Rendering,* Bristol, United Kingdom, pages 113–121, May 1992.

[81] Christophe Vedel. Computing illumination from area light sources by approximate contour integration. In *Proceedings of Graphics Interface '93*, pages 237–244, 1993.

[82] T. Vettor and T. Poggio. Linear object classes and image synthesis from a single example image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):733–742, July 1997.

[83] John Wallace, M. F. Cohen, and D. P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(3):311–320, July 1987.

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

[85] Gregory J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse environments. *Computer Graphics*, 22(3):85–92, 1988.

[86] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 32(6):343–349, June 1980.

[87] G. Wolberg, editor. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1990.