To appear in E. Goodman and J. O'Rourke, eds., "CRC Handbook of Discrete and Computational Geometry," CRC Press, 1996.

42 COMPUTER GRAPHICS

David Dobkin and Seth Teller

INTRODUCTION

Computer graphics is often given as a prime application area for the techniques of computational geometry. The histories of the two fields have a great deal of overlap, with similar methods (e.g. sweep-line and area subdivision algorithms) arising independently in each. Both fields have often focused on similar problems, although with different computational models. For example, hidden surface removal (visible surface identification) is a fundamental problem of computer graphics. This problem has also motivated many researchers in computational geometry. At the same time, as the fields have matured, they have brought different requirements to similar problems. Here, we aim to highlight both similarities and differences between the fields.

Computational geometry is fundamentally concerned with the efficient quantitative representation and manipulation of ideal geometric entities to produce exact results. Computer graphics shares these goals, in part. However, graphics practitioners also model the interaction of objects with light and with each other, and the media through which these effects propagate. Moreover, graphics researchers and practitioners: 1) typically use finite precision (rather than exact) representations for geometry; 2) rarely find closed-form solutions to problems, instead employing sampling strategies and numerical methods; 3) often design explicit tradeoffs between running time and solution quality into their algorithms; and 4) implement most algorithms they propose.

GLOSSARY

Radiometry: The quantitative study of electromagnetic radiation.

Simulation: The representation of a natural process by a computation.

Psychophysics: The study of the human visual system's response to electromagnetic stimuli.

GEOMETRY VS. RADIOMETRY AND PSYCHOPHYSICS

Computer graphics can be formulated as a radiometrically "weighted" counterpart to computational geometry. A fundamental computational process in graphics is ***rendering***: the synthesis of realistic images of physical objects. This is done through the application of a simulation process to quantitative models of light and materials to predict (i.e. ***synthesize***) appearance. Of course, this process must account for the shapes of and spatial relationships between objects, as must computational geometric algorithms. However, in graphics objects are imbued further with

material properties, such as ***reflectance*** (in its simplest form, color), ***refractive index***, ***opacity***, and (for light sources) ***emissivity***. Moreover, physically justifiable graphics algorithms must model radiometry—quantitative representations of light sources and the electromagnetic radiation they emit (with associated attributes of intensity, wavelength, phase, etc.). All graphics algorithms which produce output intended for viewing must also explicitly or implicitly involve psychophysics.

CONTINUOUS IDEAL VS. DISCRETE REPRESENTATIONS

Computational geometry is largely concerned with ideal objects (points, lines, circles, spheres, hyperplanes, polyhedra), continuous representations (effectively infinite precision arithmetic), and exact combinatorial and algebraic results. Graphics algorithms (and their implementations) model such objects as well, but do so in a discrete, finite-precision computational model. For example, most graphics algorithms use a floating-point or fixed-point coordinate representation. Thus, one can think of most computer graphics computations as occurring on a (2D or 3D) grid. However, a practical difficulty is that the grid spacing is not constant, causing certain geometric predicates (e.g., sidedness) to change under simple transformations such as scaling or translation.

An analogy can be made between this distinct choice of coordinates, and the way in which geometric objects—infinite collections of points—are represented by geometers and graphics researchers. Both might represent a sphere similarly—say, by a center and radius. However, an algorithm to render the sphere must select a finite set of "sample" points on its surface. These sample points typically arise from the position of a synthetic camera, and the locations of display elements on a two-dimensional display device, for example pixels on a computer monitor or ink dots on a page in a computer printer. The colors computed at these (zero-area) sample points, through some radiometric computation, then serve as an assignment to the discrete value of each (finite-area) display element.

CLOSED-FORM VS. NUMERICAL SOLUTION METHODS

Rarely does a problem in graphics demand a closed-form solution. Instead, graphicists typically rely on numerical algorithms to estimate solution values in an iterative fashion. Numerical algorithms are chosen by reason of efficiency, or of simplicity. Often, these are antagonistic goals. Aside from the usual dangers of finite-precision arithmetic (Chapter 35), other types of error may arise from numerical algorithms. First, using a point-sampled value to represent a finite-area function's value leads to discretization errors—differences between the reconstructed (interpolated) function, which may be piecewise-constant, piecewise-linear, piecewise-polynomial etc., and the piecewise-continuous (but unknown) true function. These errors may be exacerbated by a poor choice of sampling points, by a poor piecewise function representation or basis, or by neglect of boundaries along which the true function or its derivative have strong discontinuities. Also, numerical algorithms may suffer bias and converge to incorrect solutions (e.g., due to misweighting, or omission, of significant contributions).

TRADING SOLUTION QUALITY VS. COMPUTATION TIME

The most successful graphics algorithms recognize sources of error and seek to bound them by various means. Moreover, for efficiency's sake an algorithm might deliberately introduce error. For example, while rendering, objects might be crudely approximated to speed the geometric computations involved. Alternatively, in a more general illumination computation, many instances of combinatorial interactions (e.g., reflections) between scene elements might be ignored except when they have a significant effect on the computed image or radiometric values. Graphics practitioners have long sought to exploit this intuitive "tradeoff" between solution quality and computation time.

THEORY VS. PRACTICE

Graphics algorithms, while often designed with theoretical concerns in mind, are typically intended to be of practical use. Thus, while computational geometers and computer graphicists have an enormous overlap of interest in geometry, graphicists develop computational strategies which can feasibly be implemented on modern machines. Also, while computational geometric algorithms often assume "generic" inputs, in practice geometric degeneracies do occur, and inputs to graphics algorithms are at times highly degenerate (for example, comprised entirely of isothetic rectangles).

Thus, algorithmic strategies are shaped not only by challenging inputs which arise in practice, but also by the technologies available at the time the algorithm is proposed. The relative bandwidths of CPU, bus, memory, network connections, and tertiary storage have major implications for graphics algorithms involving interaction or large amounts of simulation data, or both. For example, in the 1980's the decreasing cost of memory, and the need for robust processing of general datasets, brought about a fundamental shift in most practitioners' choice of computational techniques for resolving visibility (from combinatorial, object-space algorithms to brute force, screen-space algorithms). The increasing power of general-purpose processors, the emergence of sophisticated, robust visibility algorithms, and the wide availability of dedicated low-level graphics hardware may bring about yet another fundamental shift.

TOWARD A MORE FRUITFUL OVERLAP

Given such substantial overlap, there is an enormous potential for fruitful collaboration between geometers and graphicists (see [CAA+96] for a recent call for just this kind of collaboration). One mechanism for spurring such collaboration is the careful posing of models and open problems to both communities. To that end, these are interspersed through the remainder of this chapter.

42.1 GRAPHICS AS A COMPUTATIONAL PROCESS

This section gives an overview of three fundamental graphics operations: ***acquisition*** of simulation data; ***representation*** of such data and associated attributes and energy sources; and ***simulation*** of these to predict behavior or appearance. One fundamental simulation process, ***rendering***, is partitioned into subcomponents of ***visibility*** and ***shading***, which are treated in separate sections below.

MODELING, ACQUISITION, AND SIMPLIFICATION

In practice, algorithms require input. Realistic scene generation demands extremely complex geometric and radiometric models—for example, of scene geometry and reflectance properties, respectively. These inputs must arise from some source; this "model acquisition" problem is a core problem in graphics. Models may be generated by a human designer (for example, using Computer-Aided Design packages), generated procedurally (for example, by applying recursive rules), or constructed by machine-aided manipulation of image data (for example, generating 3D topographical maps of terrestrial or extra-terrestrial terrain from multiple photographs) or other machine sensing methods (e.g., [CL96]). Methods for completely automatic (i.e., non human-assisted) generation of large-scale geometric models are still in their infancy; see Chapter 47. When datasets become extremely large, some kind of hierarchical, persistent spatial database is required for efficient storage and access to the data [FKST96], and simplification algorithms are necessary to store and display complex objects with varying fidelity (see, e.g., [CVM+96, HDD+92]).

REPRESENTATION: GEOMETRY, LIGHT, AND FORCES

Once a geometric model is acquired, it must be represented, and perhaps indexed spatially for efficient manipulation. A broad variety of intrinsic (winged-edge, quad-edge, facet-edge, etc.) and extrinsic (quadtree, octree, kd-tree, BSP tree, B-rep, CSG, etc.) data structures have been developed to represent geometric data. Continuous, implicit functions have been used to model shape, as have discretized volumetric representations, in which data types or densities are associated with spatial "voxels." A subfield of modeling, Solid Modeling (Chapter 47), represents shape, mass, material, connectivity, etc. properties of objects, in particular so that complex object assemblies may be defined, e.g. for use in Computer-Aided Machining environments (Chapter 46). Some of these data structures can be adaptively subdivided, and made persistent (that is, made to exist in memory and in non-volatile storage; see Section 42.3), so that models with wide scale variations, or simply an enormous data size, may be organized. None of the data structures above is universal; each has been brought to bear in specific circumstances, depending on the nature of the data (manifold vs. non-manifold; polyhedral vs. curved; etc.) and the problem at hand. We forego a detailed discussion of representational issues, referring the reader instead to Chapter 47.

The data structures alluded to above represent "macroscopic" properties of scene geometry—shape, gross structure, etc. Representing material properties, including reflectance over each surface, and possibly surface microstructure (such as roughness) and sub-structure (as with layers of skin or other tissue), is another fundamental concern of graphics. For each, computer graphics researchers craft and employ quantitative descriptions of the interaction of radiant energy and/or physical forces with objects having these properties. Examples include human-made objects such as machine parts, furniture, and buildings; organic objects such as flora and fauna; naturally occurring objects such as molecules, terrains, and galaxies; and wholly synthetic objects and materials. Analogously, suitable representations of radiant energy and physical forces must also be crafted in order that the simulation process can model such effects as erosion [DPH96].

SIMULATION

Graphics brings to bear a wide variety of simulation processes to predict behavior. For example, one might detect collisions to simulate a pair of tumbling dice, or simulate frictional forces in order to provide haptic (touch) feedback through a mechanical device to a researcher manipulating a virtual object [LMC94]. Increasingly, graphics researchers are incorporating spatialized sounds into simulations as well. These physically-based simulations are integral to many graphics applications. However, the generation of synthetic imagery is the most fundamental operation in graphics. The next section describes this "rendering" problem.

RENDERING

GLOSSARY

Rendering problem: Given quantitative descriptions of surfaces and their properties, light, and the media in which these are embedded, rendering is the application of a computational model to predict appearance; that is, rendering is the synthesis of images from simulation data. Rendering typically involves for each surface a Visibility computation followed by a Shading computation.

Visibility computation: The determination of whether some set of surfaces, or sample points, is visible to a synthetic observer.

Shading computation: The determination of radiometric values on the surface (eventually interpreted as colors) as viewed by the observer.

For static scenes, and with more difficulty when conditions change with time, rendering can be factored into geometrically and radiometrically view-independent tasks (such as spatial partitioning for intervisibility, and the computation of diffuse illumination) and their view-dependent counterparts (culling and specular illumination, respectively). The view-independent tasks can be cast as precomputations, while at least some view-dependent tasks cannot occur until the instantaneous viewpoint is known. Recently, these distinctions have been somewhat blurred by

the development of data structures that organize lazily-computed, view-dependent information for use in an interactive setting [TBD96].

We discuss the two major components of Rendering: Visibility (Section 42.2) and Shading (Section 42.3), and review the formulation of Shading as a spatially distributed, recursive, radiometrically weighted Visibility computation. In the discrete domain, we discuss Sampling (Section 42.4). We then pose several challenges for the future, comprising problems of current or future interest in computer graphics in which computational geometry may have a substantial impact (Section 42.5). Finally, we list several further references (Section 42.6).

42.2 VISIBILITY

GLOSSARY

Pixel: Picture element, for example on a raster display.

Viewport: A 2D array of pixels, typically comprising a rectangular region on a computer display.

View Frustum: A truncated rectangular pyramid, representing the synthetic observer's field of view, with the synthetic eyepoint at the apex of the pyramid. The truncation is typically accomplished using "near" and "far" clipping planes, analogous to the "left, right, top, and bottom" planes that define the rectangular field of view. (If the synthetic eyepoint is placed at infinity, the frustum becomes a rectangular parallepiped.) Only those portions of the scene geometry which fall inside the view frustum are rendered.

Rasterization: The transformation of a continuous scene description, through discretization and sampling, into a discrete set of pixels on a display device.

Ray casting: A hidden-surface algorithm in which, for each pixel of an image, a ray is cast from the synthetic eyepoint through the center of the pixel [App68]. The ray is parametrized by a variable t such that $t = 0$ is the eyepoint, and $t > 0$ indexes points along the ray increasingly distant from the eye. The first intersection found with a surface in the scene (i.e., that intersection with minimum positive t) locates the visible surface along the ray. The corresponding pixel is assigned the intrinsic color of the surface, or some computed value.

Depth-buffering (also ***z-buffering***)***:*** An algorithm which resolves visibility by storing a discrete depth (initialized to some large value) at each pixel [Cat74]. Only when a rendered surface fragment's depth is less than that stored at the pixel can the fragment's color replace that currently stored at the pixel.

LOCAL VISIBILITY COMPUTATIONS

Given a scene composed of modeling primitives (e.g., polygons, or spheres), and a

viewing frustum defining an eyepoint, a view direction and field of view, the visibility operation determines which scene points or fragments are visible—connected to the eyepoint by a line segment that meets the closure of no other primitive. The visibility computation is global in nature, in the sense that the determination of visibility along a single ray may involve all primitives in the scene. Typically, however, visibility computations can be organized to involve coherent subsets of the model geometry.

In practice, algorithms for visible surface identification operate under severe constraints. First, available memory is limited. Second, the computation time allowed may be a fraction of a second—short enough to achieve interactive refresh rates under changes in viewing parameters (for example, the location or viewing direction of the observer). Third, visibility algorithms must be simple enough to be practical, but robust enough to apply to highly degenerate scenes that arise in practice.

The advent of machine rendering techniques brought about a cascade of screen-space and object-space combinatorial hidden-surface algorithms, famously surveyed and synthesized in [SSS74]. However, a memory-intensive screen-space technique—depth-buffering—soon won out due to its brutal simplicity and the decreasing cost of memory. In depth-buffering, specialized hardware performs visible surface determination independently at each pixel. Each polygon to be rendered is rasterized, producing a collection of pixel coordinates and an associated depth for each. A polygon fragment is allowed to "write" its color into a pixel only if the depth of the fragment at hand is less than the depth stored at the pixel (all pixel depths are initialized to some large value). Thus, in a complex scene each pixel might be written many times to produce the final image, wasting compute and memory bandwidth. This is known as the ***overdraw*** problem.

Two decades of spectacular improvement in graphics hardware have ensued, and high-end graphics workstations now contain hundreds of increasingly complex processors which clip, illuminate, rasterize, and texture millions of polygons per second. This capability increase has naturally led users to produce ever more complex geometric models, which suffer from increasing overdraw. Object simplification algorithms, which represent complex geometric assemblages with simpler shapes, do little to reduce overdraw. Thus, visible-surface identification (hidden-surface elimination) algorithms have again come to the fore. In recent years, several hybrid object-space/screen-space visibility algorithms have emerged (e.g., [GKM93]). As general purpose processors continue to become faster, such hybrid algorithms will become more widely used. In certain situations, these algorithms will operate entirely in object space, without relying on special-purpose graphics hardware [CT96]. That is, sufficiently fast processors and efficient algorithms will augment, and may supplant, the depth-buffer in graphics architectures of the coming decade.

GLOBAL VISIBILITY COMPUTATIONS

Real-time systems perform visibility computations from an instantaneous synthetic viewpoint along rays associated with one or more samples at each pixel of some viewport. However, visibility computations also arise in the context of global illumination algorithms, which attempt to identify *all* significant light transport among

point and area emitters and reflectors, in order to simulate realistic visual effects such as diffuse and specular interreflection and refraction. A class of "global" visibility algorithms has arisen for these problems. For example, in radiosity computations, a fundamental operation is determining ***area-area*** visibility in the presence of ***blockers***; that is, the identification of those (area) surface elements visible to a given element, and for those partially visible, all tertiary elements causing (or potentially causing) occlusion [HW91, HSA91].

CONSERVATIVE ALGORITHMS

Graphics algorithms often employ *quadrature* techniques in their innermost loops—for example, estimating the energy arriving to one surface from another by casting multiple rays and determining an energy contribution along each. Thus, any efficiency gains in this frequent process (e.g., omission of energy sources known not to contribute any energy at the receiver, or omission of objects known not to be blockers) will significantly improve overall system performance. Similarly, occlusion culling algorithms (omission of objects known not to contribute pixels to the rendered image) can significantly reduce overdraw. Both techniques are examples of ***conservative*** algorithms, which overestimate some geometric set by combinatorial means, then perform a final sampling-based operation which produces a (discrete) solution or quadrature. Of course, the success of conservative algorithms in practice depends on two assumptions: first, that through a relatively simple computation, a usefully tight bound can be attained on whatever set would have been computed by a more sophisticated (e.g., exact) algorithm; and second, that the aggregate time of the conservative algorithm and the sampling pass is less than that of an exact algorithm for input sizes encountered in practice.

This idea can be illustrated as follows. Suppose one is to render an n-polygon scene. Should one choose to render *exactly* the visible fragments, one's algorithm would expend at least kn^2 time, since since n polygons (e.g., two slightly misaligned combs, each with $n/2$ teeth) can cause $O(n^2)$ visible fragments to arise. But a conservative algorithm might simply render all n polygons, incurring some overdraw (to be resolved by a depth-buffer) at each pixel, but expending only time linear in the size of the input.

This highlights an important difference between computational geometry and computer graphics. Standard computational geometry cost measures would show that the $O(n^2)$ algorithm is optimal in an output-sensitive model (see Chapter 25). In computer graphics, hardware considerations motivate a fundamentally different approach: rendering a (judiciously chosen) superset of those polygons which will contribute to the final image. A major open problem is to unify these approaches by finding a cost function that effectively models such considerations (see below).

HARDWARE TRENDS

Special-purpose graphics hardware may go away, except for the *crossbar switch* [MCEF94], or geometric transformation operation, that takes rendered entities from their initial locations (fragments of object-space primitives) to their final location

in screen space (rendered pixels). Likewise, modern graphics architectures must perform visibility determination, and all contain some form of depth buffer. However, given sufficiently fast ray casting capability, the depth buffer might no longer be necessary. One could perform "analytic visibility" at each pixel, obviating the standard graphics technique of fragmenting all objects into large collections of polygons, then rasterizing them. Another alternative is "image-based" rendering (see below).

OPEN PROBLEMS

Each of the problems below assumes a geometric model consisting of n polygons.

1. Solved Problem: The set of visible fragments can have complexity $\omega(n^2)$ in the worst case. However, the complexity is lower for many scenes. If k is the number of edge incidences (vertices) in the projected visible scene, the set of visible fragments can be computed in optimal output sensitive $O(nk^{1/2} \lg\ n)$ time [SO92]. See Chapter 25.

2. Give a spatial partitioning and ray casting algorithm that runs in amortized nearly constant time (that is, has only a weak asymptotic dependence on total scene complexity). Identify a useful "density" parameter of the scene (e.g., the largest number of simultaneously visible polygons), and express the amortized cost of a ray cast in terms of this parameter.

3. Give an output-sensitive algorithm which, for specified viewing parameters, determines the set of "contributing" polygons—i.e., those which contribute their color to at least one viewport pixel.

4. Give an output-sensitive algorithm which, for specified viewing parameters, approximates the visible set to within ϵ. That is, produce a superset of the visible polygons of size (alternatively, total projected area) at most $(1 + \epsilon)$ times the size (projected area) of the true set. Is the lower bound for this problem asymptotically smaller than that for the exact visibility problem?

5. For machine-dependent parameters A and B describing the transform (per-vertex) and fill (per-pixel) costs of some rendering architecture, give an algorithm to compute a superset S of the visible polygon set minimizing the rendering cost on the specified architecture.

6. In a local illumination computation, identify those polygons (or a superset) visible from the synthetic observer, and construct, for each visible polygon P, an efficient function $V(p)$ which returns 1 iff point p on P is visible from the viewpoint.

7. In a global illumination computation, identify all pairs (or a superset) of intervisible polygons, and for each such pair P, Q, construct an efficient function $V(p, q)$ which returns 1 iff point p on P is visible from point q on Q.

8. ***Image-based rendering*** [MB95]: Given a 3D model, generate a minimal set of images of the model such that for all subsequent query viewpoints, the correct image can be recovered by combination of the sample images.

42.3 SHADING

Through sampling and visibility operations, a visible surface point or fragment is identified. This point or fragment is then ***shaded*** according to a ***local*** or ***global*** illumination algorithm. Given scene light sources and material reflection and transmission properties, and the propagative media comprising and surrounding the scene objects, the shading operation determines color and intensity of the incident and exitant radiation at the point to be shaded. Shading computations can be characterized further as ***view-independent*** (modeling only purely diffuse interactions, or directional interactions with no dependence on the instantaneous eye position) or ***view-dependent***.

Most graphics workstations perform a "local" shading operation in hardware, which, given a point light source, a surface point, and an eye position, evaluates the energy reaching the eye via a single reflection from the surface. This local operation is implemented in the software and hardware offered by most workstations. However, this simple model cannot produce realistic lighting cues such as shadows, reflection, and refraction. These require more extensive, global computations as described below.

GLOSSARY

Irradiance: Total power per unit area impinging on a surface element. Units: POWER PER RECEIVER AREA.

BRDF: The Bi-directional Reflectance Distribution Function, which maps incident radiation (at general position and angle of incidence) to reflected exitant radiation (at general position and angle of exitance). Unitless, in [0..1].

BTDF: The Bi-directional Transmission Distribution Function, which maps incident radiation (at general position and angle of incidence) to transmitted exitant radiation (at general position and angle of exitance). Analogous to the BRDF.

Radiance: The fundamental quantity in image synthesis, which is conserved along a ray traveling through a non-dispersive medium, and is therefore "the quantity that should be associated with a ray in ray tracing" [CW93]. Units: POWER PER SOURCE AREA PER RECEIVER STERADIAN.

Radiosity: A global illumination algorithm for ideal diffuse environments. Radiosity algorithms compute shading estimates which depend only on the surface normal and the size and position of all other surfaces and light sources, and are independent of view direction. Also a physical quantity with units POWER PER SOURCE AREA.

Ray tracing: An image synthesis algorithm in which ray casting is followed, at each surface, by a recursive shading operation involving a hemispherical integral of irradiance at each surface point. Ray tracing algorithms are best suited to scenes with small light sources and specular surfaces.

Hybrid: Global illumination algorithms which model both diffuse and directional interactions (e.g., [SP89]).

SHADING AS RECURSIVE WEIGHTED INTEGRATION

Most generally, the shading operation computes the energy leaving a differential surface element in a specified differential direction. This energy depends on the surface's emittance, and the product of the surface's reflectance with the total energy incident from all other surfaces. This relation is known as the ***Rendering Equation*** [Kaj86], which states intuitively that each surface fragment's appearance, as viewed from a given direction, depends on any light it emits, plus any light (gathered from other objects in the scene) which it reflects in the direction of the observer. Thus shading can be cast as a recursive integration; to shade a surface fragment F, shade all fragments visible to F, then sum those fragments' illumination upon F (appropriately weighted by the BRDF or BTDF) with any direct illumination of F. Effects such as diffuse illumination, motion blur, Fresnel effects, etc., can be simulated by supersampling in space, time, and wavelength, respectively, then averaging [CPC84].

Of course, a base case for the recursion must be defined. Classical ray tracers truncate the integration when a certain recursion depth is reached. If this maximum depth is set to 1, ray casting (the determination of visibility for eye rays only) results. More common is to use a small constant greater than one, which leads to "Whitted" or "classical" ray tracing [Whi80]. For efficiency, practitioners also employ a thresholding technique: when multiple reflections cause the weight with which a particular contribution will contribute to the shading at the root to drop below a specified threshold, the recursion ceases. These termination conditions can, in some conditions, cause important energy-bearing paths to be overlooked. For example, an extremely bright light source (such as the sun) filtering through many parts of a house to reach an interior space may be incorrectly discarded by this condition.

ALIASING

From a purely physical standpoint, the amount of energy leaving a surface in a particular direction is the spherical integral of incoming energy, times the bidirectional reflectance (and transmittance, as appropriate) in the exitant direction. From a psychophysical standpoint, the perceived color is an inner product of the energy distribution incident on the retina with the retina's spectral response function. We do not consider psychophysical considerations further here.

Global illumination algorithms perform an integration of irradiance at each point to be shaded. Ray tracing and Radiosity are examples of global illumination algorithms. Since no closed-form solutions for global illumination are known for general scenes, practitioners employ sampling strategies. Graphics algorithms typically attempt "reconstruction" of some illumination function (e.g., irradiance, or radiance) given some set of samples of the functions' values, and possibly other information, for example about light source positions, etc. However, such reconstruction is subject to error for two reasons.

First, the well-known phenomenon of ***aliasing*** occurs when insufficient samples are taken to find all high-frequency terms in a sampled signal. In image processing,

samples arise from measurements, and reconstruction error arises from samples which are too widely spaced. However, in graphics, the sample values arise from a simulation process; for example, the evaluation of a local illumination equation, or the numerical integration of irradiance. Thus, reconstruction error can arise from simulation errors in generating the samples. This second type of error is called ***biasing***.

For example, classical ray tracers [Whi80] may suffer from biasing in three ways. First, at each shaded point, they compute irradiance only: from direct illumination by point lights; along the reflected direction; and along the refracted direction. Significant "indirect" illumination which occurs along any direction other than these is not accounted for. Thus indirect reflection and focusing effects are missed. Classical ray tracers also suffer biasing by truncating the depth of the recursive ray tree at some finite depth d; thus, they cannot find significant paths of energy from light source to eye of length greater than d. Third, classical ray tracers truncate ray trees when their weight falls below some threshold. This can fail to account for large radiance contributions due to bright sources illuminating surfaces of low reflectance.

OPEN PROBLEMS

An enormous literature of adaptive, backward, forward, distribution, etc., ray tracers has grown up to address sampling and bias errors. However, the fundamental issue can be stated simply as:

1. Given a geometric model M, a collection of light sources L, a synthetic viewpoint E, and a threshold ϵ, identify all optical paths to E bearing radiance greater than ϵ.
2. Given a geometric model M, a collection of light sources L, and a threshold ϵ, identify *all* optical paths bearing radiance greater than ϵ.

A related *inverse* problem arises in machine vision, now being adopted by computer graphics practitioners as a method for acquiring large-scale geometric models from imagery:

3. An observation of a real object comprises the *product* of irradiance and reflection (BRDF). How can one deduce the BRDF from such observations?

42.4 SAMPLING

Sampling patterns can arise from a regular grid (e.g., pixels in a viewport) but these lead to a stairstepping kind of aliasing. One solution is to ***supersample*** (i.e. take multiple samples per pixel) and average the results. However, one must take care to supersample in a way that does not align with the scene geometry or some underlying attribute (e.g., texture) in a periodic, spatially varying fashion, else aliasing (including Moiré patterns) will result.

DISCREPANCY

The quality of sampling patterns can be evaluated with a measure known as ***discrepancy***. For example, if we are sampling in a pixel, features interacting with the pixel can be modeled by line segments (representing parts of edges of features) crossing the pixel. These segments divide the pixel into two regions. A good sampling strategy will ensure that the proportion of sample points in each region approximates the proportion of pixel area in that region. The difference between these quantities is the discrepancy of the point set with respect to the line segment. We define the discrepancy of a set of samples (in this case) as the maximum discrepancy with respect to all line segments. Other measures of discrepancy are possible, as described below.

Sampling patterns are used to solve integral equations. The advantage of using a low discrepancy set is that the solution will be more accurately approximated, resulting in a better image. These differences are expressed in solution convergence rates as a function of the number of samples. For example, truly random sampling has a discrepancy that grows as $O(N^{-\frac{1}{2}})$ where N is the number of samples. There are other sampling patterns (e.g. the *Hammersley points*) that have discrepancy growing as $O(N^{-1}\lg^{k-1} N)$. Sometimes one wishes to combine values obtained by different sampling methods [VG95]. The search for good sampling patterns, given a fixed number of samples, is often done by running an optimization process which aims to find sets of ever-decreasing discrepancy. A crucial part of any such process is the ability to quickly compute the discrepancy of a set of samples.

COMPUTING THE DISCREPANCY

There are two common questions that arise in the study of discrepancy. First, given fixed N, how to construct a good sampling pattern in the model described above. Second, how to construct a good sampling pattern in an alternative model.

For concreteness, consider the problem of finding low discrepancy patterns in the unit square. The unit square models an individual pixel. As stated above, the geometry of objects is modeled by edges that intersect the pixel dividing it into two regions, one where the object exists and one where it does not. An ideal sampling method would sample the regions proportion to their relative areas.

We model this as a discrepancy problem as follows. Let S be a sample set of points in the unit square. For a line l (actually, a segment arising from a polygon boundary in the scene being rendered), define the two regions S^+ and S^- into which l divides S. Ideally, we want a sampling pattern that has the same fraction of samples in the region S^+ as the area of S^+. Thus, in the region S^+, the discrepancy with respect to l is $|\sharp(S \cap S^+)/\sharp(S) - Area(S^+)|$, where $\sharp(\cdot)$ denotes the cardinality operator. The discrepancy of the sample set S with respect to a line l is defined as the larger of the discrepancies in the two regions. The discrepancy of set S is then the maximum, over *all* lines l, of the discrepancy of S with respect to l. The intuition behind this definition is fairly clear.

Finding the discrepancy in this setting is an interesting computational geometry problem. First, we observe that we do not need to consider all lines. Rather, we

need consider only those lines which pass through two of the points in our set plus a few lines derived from boundary conditions. This suggests the $O(n^3)$ algorithm of computing the discrepancy of each of the $O(n^2)$ lines separately. This can be improved to $O(n^2 \lg n)$ by considering the fan of lines with a common vertex (i.e. one of the sample points) together. This can be further improved by appealing to duality. The traversal of this fan of lines is merely a walk in the arrangement of lines in dual space that are the duals of the sample points. This observation allows us to use techniques similar to those in Chapter 21 to derive an algorithm that runs asymptotically as $O(n^2)$. Full details are given in [DEM93].

There are other discrepancy models that arise naturally. A second obvious candidate is to measure the discrepancy of sample sets in the unit square with respect to axis-oriented rectangles. Here we can achieve a discrepancy of $O(n^2 \lg n)$, again using geometric methods. We use a combination of techniques, appealing to the incremental construction of 2D convex hulls to solve a basic problem, then using the sweep paradigm to extend this incrementally to a solution of the more general problem. The sweep is easier in the case in which the rectangle is anchored with one vertex at the origin, yielding an algorithm with running time $O(n \lg^2 n)$.

The model given above can be generalized to compute ***bichromatic discrepancy***. In this case, we have sample points which are colored either black or red. We can now define the discrepancy of a region as the difference between its number of red and black points. Alternatively, we can look for regions (of the allowable type) that are most nearly monochromatic in red while their complements are nearly monochromatic in black. This latter model has application in computational learning theory. For example, red points may represent situations in which a concept is true, black situations where it is false. The minimum discrepancy rectangle is now a classifier of the concept. This is a popular technique for computer-assisted medical diagnosis.

The relevance of these algorithms to computational geometry is that they will lead to faster algorithms for testing the "goodness" of sampling patterns, and thus eventually more efficient algorithms with bounded sampling error. Also, algorithms for computing the discrepancy relative to a particular set system are directly related to the system's V-C dimension (see Chapter 31).

OPEN PROBLEMS

Both problems below are posed for the unit cube and unit ball in all dimensions.

1. Given N, generate a minimum-discrepancy pattern of N samples.
2. Given a low-discrepancy pattern of K points, generate a low (or lower) discrepancy pattern of $K + 1$ points.

42.5 FURTHER CHALLENGES

We have described several core problems of computer graphics and shown the impact of computational geometry. We have only scratched the surface of a highly fruitful interaction; the possibilities are expanding, as we describe below. These computer graphics problems all build on the combinatorial framework of computational geometry and so have been, and continue to be, ripe candidates for application of computational geometry techniques. Numerous other problems remain whose combinatoric aspects are perhaps less obvious, but for which interaction may be equally fruitful. We describe some of these here.

We have focused this chapter on problems in which the parameters are static; that is, the geometry is unchanging, and nothing is moving (except perhaps the synthetic viewpoint). Now, we briefly describe situations where this is not the case and deeper analysis is required. In these situations it is likely that computational geometry can have a tremendous impact; we sketch some possibilities here.

Each of the static assumptions above may be relaxed, either alone or in combination. For example, objects may evolve with time; we may be interested in transient rather than steady state solutions; material properties may change over time; object motions may have to be computed and resolved, etc. There is a challenge in determining how the techniques of computational geometry can be modified to address state of the art and future computer graphics tasks in dynamic environments.

Among the issues we have not addressed where these considerations are important are the following:

Collision detection and force feedback. Imagine that every object has an associated motion, and that some objects (e.g., virtual probes) are interactively controlled. Suppose further that when pairs of objects intersect, there is a reaction (due, e.g. to conservation of momentum). Here we wish to render frames, and generate haptic feedback, while accounting for such physical considerations. Are there suitable data structures and algorithms within computational geometry to model and solve this problem (e.g., [LMC94, MC95])?

Model changes over time. In a realistic model, even unmoving objects change over time, for example becoming dirty or scratched. In some environments, objects rust or suffer other corrosive effects. Sophisticated geometry representations and algorithms are necessary to capture and model such phenomena [DPH96].

Inverse processes. Much of what we have described is a feed forward process in which one specifies a model and a simulation process and computes a result. Of equal importance in design contexts is to specify a result and a simulation process, and compute a set of initial conditions that would produce the desired result. For example, one might wish to specify the appearance of a stage, and deduce the intensities of hundreds of illuminating light sources that would result in this appearance [SDSA93]. Or, one might wish to solve an inverse kinematics problem in which an object with multiple parts and numerous degrees of freedom is specified. Given initial and final states, one must compute a smooth, minimal energy path between the states, typically in an underconstrained framework. This is a common

problem in robotics (see Chapter 41). However, the configurations encountered in graphics tend to have very high complexity. For example, convincingly simulating the motion of a human figure requires processing kinematic models with hundreds of degrees of freedom.

External memory algorithms. Computational geometry assumes a realm in which all data can be stored in RAM and accessed at no cost (or unit cost). Increasingly often, this is not the case in practice. For example, many large databases cannot be stored in main memory. Only a small subset of the model contributes to each generated image, and efficient algorithms for efficiently identifying this subset, and maintaining it under small changes of the viewpoint or model, form an active research area in computer graphics. Given that motion in virtual environments is usually smooth, and that hard real-time constraints preclude the use of purely reactive, synchronous techniques, such algorithms must be *predictive* and *asynchronous* in nature [FKST96]. Achieving efficient algorithms for appropriately shuttling data between secondary (and tertiary) storage and main memory is an interesting challenge for computational geometry.

42.6 SOURCES AND RELATED MATERIAL

SURVEYS

All results not given an explicit reference above may be traced in these surveys.

- [Dob92]: A survey article on computation geometry and computer graphics.
- [Dor94]: A survey of object-space hidden-surface removal algorithms.
- [Yao90, LP84]: Surveys of computational geometry.

RELATED CHAPTERS

Chapter 22: Triangulations
Chapter 23: Polygons
Chapter 32: Ray shooting and lines in space
Chapter 36: Parallel algorithms in geometry
Chapter 46: Computer-aided machining
Chapter 47: Solid modeling

REFERENCES

References

[App68] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of SJCC*, pages 37–45. Thompson Books, Washington, D.C., 1968.

[CAA+96] B. Chazelle, N. Amenta, T. Asano, G. Barequet, M. Bern, J.-D. Boissonnat, J. Canny, K. Clarkson, D. Dobkin, B. Donald, S. Drysdale, H. Edelsbrunner, D. Eppstein, A. R. Forrest, S. Fortune, K. Goldberg, M. Goodrich, L. Guibas, P. Hanrahan, C. Hoffmann, D. Huttenlocher, H. Imai, D. Kirkpatrick, D. Lee, K. Mehlhorn, V. Milenkovic, J. Mitchell, M. Overmars, R. Pollack, R. Seidel, M. Sharir, J. Snoeyink, G. Toussaint, S. Teller, H. Voelcker, E. Welzl, and C. Yap. Application Challenges to Computational Geometry: CG Impact Task Force Report. Technical Report TR-521-96, Princeton CS Dept., April 1996.

[Cat74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, December 1974. Also TR UTEC-CSc-74–133, CS Dept., University of Utah.

[CL96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH '96*, pages 303–312, 1996.

[CPC84] R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 137–45, July 1984.

[CT96] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc.* 12^{th} *Annual ACM Symposium on Computational Geometry*, 1996.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Proceedings of SIGGRAPH '96*, pages 119–128, August 1996.

[CW93] M. Cohen and J. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press, Cambridge, MA, 1993.

[DEM93] D. Dobkin, D. Eppstein, and D. Mitchell. Computing the discrepancy with applications to supersampling patterns. In *Proc.* 9^{th} *Annual ACM Symposium on Computational Geometry*, pages 47–52, 1993.

[Dob92] D. Dobkin. Computational geometry and computer graphics. *Proceedings of the IEEE*, 80:1400–1411, 1992.

[Dor94] S. Dorward. A survey of object-space hidden surface removal. *Internat. J. Comput. Geom. Appl.*, 4:325–362, 1994.

[DPH96] J. Dorsey, H. Pedersen, and P. Hanrahan. Flow and changes in appearance. In *Proceedings of SIGGRAPH '96*, pages 411–420, 1996.

[FKST96] T. Funkhouser, D. Khorramabadi, C. Séquin, and S. Teller. The UCB system for interactive visualization of large architectural models. *Presence*, 5(1):13–44, Winter 1996.

[GKM93] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Proceedings of Siggraph '93*, pages 231–238, August 1993.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78, July 1992.

[HSA91] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics (Proc. Siggraph '91)*, 25(4):197–206, 1991.

[HW91] E. Haines and J. Wallace. Shaft culling for efficient ray-traced radiosity. In *Proc. 2^{nd} Eurographics Workshop on Rendering*, May 1991.

[Kaj86] J. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[LMC94] M. Lin, D. Manocha, and J. Canny. Fast contact determination in dynamic environments. In Edna Straub and Regina Spencer Sipple, editors, *Proceedings of the International Conference on Robotics and Automation. Volume 1*, pages 602–609, May 1994.

[LP84] D. Lee and F. Preparata. Computational geometry: a survey. *IEEE Transactions on Computing*, 33:1072–1101, 1984.

[MB95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. Addison Wesley, August 1995.

[MC95] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *1995 Symposium on Interactive 3D Graphics*, pages 181–188, April 1995.

[MCEF94] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[SDSA93] C. Schoeneman, J. Dorsey, B. Smits, and J. Arvo. Painting with light. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 143–146, 1993.

[SO92] M. Sharir and M. Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11(1):1–11, 1992.

[SP89] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 335–344, July 1989.

[SSS74] I. Sutherland, R. Sproull, and R. Schumacker. A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1):1–55, 1974.

[TBD96] S. Teller, K. Bala, and J. Dorsey. Conservative radiance envelopes for ray tracing. In *Proc. 7^{th} Eurographics Workshop on Rendering*, pages 105–114, June 1996.

[VG95] E. Veach and L. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *SIGGRAPH 95 Conference Proceedings*, pages 419–428, August 1995.

[Whi80] T. Whitted. An improved illumination model for shading display. *CACM*, 23(6):343–349, 1980.

[Yao90] F. Yao. *Computational geometry*, pages 345–490. Elsevier, 1990.