52 COMPUTER GRAPHICS

David Dobkin and Seth Teller

INTRODUCTION

Computer graphics has often been cited 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 in both fields. At the same time, as the fields have matured, they have brought different requirements to similar problems. The body of this chapter contains an updated version of our chapter in an earlier edition of this Handbook [DT04] in which we aimed 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 formulate closed-form solutions to problems, instead employing sampling strategies and numerical methods; (3) often design into their algorithms explicit tradeoffs between running time and solution quality; (4) often analyze algorithm performance by defining as primitive operations those that have been implemented in particular hardware architectures rather than by measuring asymptotic behavior; and (5) implement most algorithms they propose.

The relationship described above has faded over the years. At the present time, the overlap between computer graphics and computational geometry has retreated from many of the synergies of the past. The availability of fast hardware driven by the introduction of graphics processing units (GPUs) that provide high speed computation at low cost has encouraged computer graphics practitioners to seek solutions that are engineering based rather than more scientifically driven. At the same time, the basic science underlying computational geometry continues to grow providing a firm basis for understanding basic processes. Adding to this divide is the unfortunate split between theoretical and practical considerations that arise when handling the very large data sets that are becoming common in the computer graphics arena. In the past, it was hoped that the asymptotic nature of many algorithms in computational geometry would bear fruit as the number of elements being processed in a computer graphics scene grew large. Unfortunately, other considerations have been more significant in the handling of large scenes. The gain in asymptotic behavior of many algorithms is overwhelmed by the additional cost in data movement required for their implementation.

Recognizing that the divide between computational geometry and computer graphics is wide at the moment, this chapter has been updated from the earlier

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

edition but new material has not been added. This decision was made in the hope that as processing power and memory sizes of more conventional CPUs become dominant in the future, there will arise a time when computer graphics practitioners will again need a firmer scientific basis on which to develop future results. When this day arrives, the ideas covered in this chapter could provide a productive landscape from which this new collaboration can develop.

52.1 RELATIONSHIP TO COMPUTATIONAL GEOMETRY

In this section we elaborate these five contacts and contrasts.

GEOMETRY VS. RADIOMETRY AND PSYCHOPHYSICS

One 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, materials, and transmission media to predict (i.e., *synthesize*) appearance. Of course, this process must account for the shapes of and spatial relationships among objects and the viewer, as must computational geometric algorithms. In graphics, however, 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, polarization, phase, etc.), and the psychophysical aspects of the human visual system. Thus rendering is a kind of radiometrically and psychophysically "weighted" counterpart to computational geometry problems involving interactions among objects.

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 many computer graphics computations as occurring on a (2D or 3D) sample 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 (see Chapter 45).

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 placement of a synthetic camera and from the locations of display elements on a two-dimensional display device, for example pixels on a computer monitor or ink

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

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 quantization into finite-precision arithmetic (Chapter 45), 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 the misweighting, or omission, of significant contributions).

TRADING SOLUTION QUALITY FOR COMPUTATION TIME

Many 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, during 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 a substantial overlap of interest in geometry, graphicists develop computational strategies that can feasibly be implemented on modern machines. These strategies change as the nature of the machines changes. For example, the rise of high speed specialized hardware has caused graphics processes to change overnight. Within computational geometry, it is rare to see such drastic changes in the set of primitive operations against which an algorithm is evaluated. 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 that arise in practice, but also by the technologies available at the time the algorithm is

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

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 1980s 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, programmable low-level graphics hardware have brought about yet another fundamental shift in recent decades.

TOWARD A MORE FRUITFUL OVERLAP

Given such substantial overlap, there is potential for fruitful collaboration between geometers and graphicists [CAA+96]. One mechanism for spurring such collaboration is the careful posing of models and open problems to both communities. To that end, these are interspersed throughout the remainder of this chapter.

52.2 GRAPHICS AS A COMPUTATIONAL PROCESS

This section gives an overview of three fundamental graphics operations: *acquisition* of some *representation* of model data, its associated attributes and illumination sources; *rendering*, or simulating the appearance of static scenes; and simulating the *behavior* of dynamic scenes either in isolation or under the influence of user *interaction*.

GLOSSARY

Rendering problem: Given quantitative descriptions of surfaces and their properties, light sources, and the media in which all 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: Determining which pairs of a set of objects in a scene share an unobstructed line of sight.

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

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

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

REPRESENTATION: GEOMETRY, LIGHT, AND FORCES

Every computational process requires some representation in a form amenable to simulation. In graphics, the quantities to be represented span shape, appearance,

and illumination. In simulation or interactive settings, forces must also be represented; these may arise from the environment, from interactions among objects, or from the user's actions.

The graphics practitioner's choice of representation has significant implications. For example, how is the data comprising the representation to be acquired? For efficient manipulation or increased spatial or temporal coherence, the representation might have to include, or be amenable to, spatial indexing. A number of intrinsic (winged-edge, quad-edge, facet-edge, etc.) and extrinsic (quadtree, octree, k-d 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 57), represents shape, mass, material, and connectivity properties of objects, so that, for example, complex object assemblies may be defined for use in Computer-Aided Machining environments. Some of these data structures can be adaptively subdivided, and made persistent (that is, made to exist in memory and in nonvolatile storage; see Chapter 38), so that models with wide-scale variations, or simply enormous data size, may be handled. None of these data structures is universal; each has been brought to bear in specific circumstances, depending on the nature of the data (manifold vs. nonmanifold, polyhedral vs. curved, etc.) and the problem at hand. We forego here a detailed discussion of representational issues; see Chapters 56 and 57.

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 substructure (as with layers of skin or other tissue), is another fundamental concern of graphics. For each material, 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 also must be crafted in order that the simulation process can model such effects as erosion [DPH96].

ACQUISITION

In practice, algorithms require input. Realistic scene generation can demand complex geometric and radiometric models—for example, of scene geometry and reflectance properties, respectively. Nongeometric scene generation methods can use sparse or dense collections of images of real scenes. Geometric and image 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 extraterrestrial terrain from multiple photographs), or other machine-sensing methods (e.g., [CL96]). Methods for largely automatic (i.e., minimally human-assisted) acquisition of large-scale geometric models have arisen in the past decade. For ex-

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

ample, Google StreetView [GSV16] acquisition combined with methods of image stitching (e.g., [Sze10]) developed in the computer vision community have largely automated the acquisition task.

RENDERING

We partition the simulation process of *rendering* into *visibility* and *shading* subcomponents, which are treated in separate subsections below.

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 surface intervisibility, and the computation of diffuse illumination) and their view-dependent counterparts (culling and specular illumination, respectively). View-independent tasks can be cast as precomputations, while at least some view-dependent tasks cannot occur until the instantaneous viewpoint is known. These distinctions have been blurred by the development of data structures that organize lazily-computed, view-dependent information for use in interactive settings [TBD96].

INTERACTION (SIMULATION OF DYNAMICS)

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.

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]).

We first discuss algorithmic aspects of model acquisition, a fundamental first step in graphics (Section 52.3). We next introduce rendering, with its intertwined operations of visibility determination, shading, and sampling (Section 52.4). We then pose several challenges for the future, listing problems of current or future interest in computer graphics on which computational geometry may have a substantial impact (Section 52.5). Finally, we list further references (Section 52.6).

52.3 ACQUISITION

Model acquisition is fundamental in achieving realistic, complex simulations. In some cases, the required model information may be "authored" manually, for example by a human user operating a computer-aided design application. Clearly manual authoring can produce only a limited amount of data. For more complex inputs, simulation designers have crafted "procedural" models, in which code is written to

generate model geometry and attributes. However, such models often have limited expressiveness. To achieve both complexity and expressiveness, practitioners employ sensors such as cameras and range scanners to "capture" representations of real-world objects.

GLOSSARY

Model capture: Acquiring a data representation of a real-world object's shape, appearance, or other properties.

GEOMETRY CAPTURE

In crafting a geometry capture method, the graphics practitioner must choose a sensor, for example a (passive) camera or multi-baseline stereo camera configuration, or an (active) laser range-finder. Regardless of sensor choice, data fusion from several sensors requires intrinsic and extrinsic sensor *calibration* and *registration* of multiple sensor observations. The fundamental algorithmic challenges here include handling noisy data, and solving the *data association* problem, i.e., determining which features match or correspond across sensor observations. When the device output (e.g., a point cloud) is not immediately useful as a geometric model, an intermediate step is required to infer geometric structure from the unorganized input [HDD+92, AB99]. These problems are particularly challenging in an interactive context, for example when merging range scans acquired at video rate [RHHL02]. A decade ago, the data size became enormous in the "Digital Michelangelo" project [LPC+00] or in GIS (geographical information systems) applied over large land areas (see Chapter 59). More recently, the scanning provided by projects such as Google Earth (estimated at over 20 petabytes of information) [Mas12] has added many orders of magnitude to the complexity of problems that arise in the capture of geometry. At the same time, the availability of (relatively) high speed processing in handheld devices coupled with bandwidth that allows these devices to download from servers with little latency has facilitated the rapid display of captured images in a variety of contexts.

One thrust common to both computer graphics and computer vision includes attempts to recover 3D geometry from many cameras situated outside or within the object or scene of interest. These "volumetric stereo" algorithms must face representational issues: a voxel data structure grows in size as the cube of the scene's linear dimension, whereas a boundary representation is more efficient but requires additional a priori information.

Another class of challenges arises from hybrids of procedural and data-driven methods. For example, there exist powerful "grammars" that produce complex synthetic flora using recursive elaboration of simple shapes [MP96]. These methods have a high "amplification factor" in the sense that they can produce complex geometry from a small number of parameters. However, they are notoriously difficult to invert; that is, given a set of observations of a tree, it is apparently difficult to recover an L-system (a particular string rewriting system) that reproduces the tree.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

APPEARANCE CAPTURE

Another aspect of capture arises in the process of acquiring texture properties or other "appearance" attributes of geometric models. A number of powerful procedural methods exist for texture generation [Per85] and 3D volumetric effects such as smoke, fire, and clouds [SF95]. Researchers are challenged to make these methods data-driven, i.e., to find the procedural parameters that reproduce observations.

Recently, appearance capture approaches have emerged that attempt to avoid explicit geometry capture. These "image-based" modeling techniques [MB95] gather typically dense collections of images of the object or scene of interest, then use the acquired data to reconstruct images from novel viewpoints (i.e., viewpoints not occupied by the camera). Outstanding challenges for developers of these methods include: crafting effective sampling and reconstruction strategies; achieving effective storage and compression of the input images, which are often highly redundant; and achieving classical graphics effects such as re-illumination under novel lighting conditions when the underlying object geometry is unknown or only approximately known. Acquisition strategies are also needed when capturing materials with complex appearance due to, for example, subsurface effects (e.g., veined marble) [LPC+00].

MOTION CAPTURE

Capturing geometry and appearance of static scenes populated by rigid bodies is challenging. Yet this problem can itself be generalized in two ways. First, scenes may be dynamic, i.e., dependent on the passage of time. Second, scene objects may be articulated, i.e., composed of a number of rigid or deformable subobjects, linked through a series of geometric transformations. Although the dimensionality of the observed data may be immense, the actual number of degrees of freedom can be significantly lower; the computational challenge lies in discovering and representing the reduced dimensions efficiently and without an unacceptable loss of fidelity to the original motion. Thus motion capture yields a host of problems: segmenting objects from one another and from outlier data; inference of object substructure and degrees of freedom; and scaling up to complex articulated assemblies. Some of these problems have been addressed in Computer Vision (see also Chapter 54), although in graphics the same problems arise when processing 3D range (in contrast to 2D image) data.

OPEN PROBLEMS

Given the existence of the ultra-large data sets that have been developed in the computer graphics and computer vision communities, find the proper mathematical framework in the computational geometry community through which to evaluate the relative performance of competing algorithms for a variety of tasks. Even a partial solution to this problem will require a thorough understanding of current hardware as well as a reasonable projection of future hardwares.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

52.4 RENDERING

Rendering is the process through which a computer image of a model (acquired or otherwise) is created. To render an image that is perceived by the human visual system as being accurate is often considered to be the fundamental problem of computer graphics (*photorealistic rendering*). To do so requires visibility computations to determine which portions of objects are not obscured. Also required are shading computations to model the photometry of the situation. Because the resultant image will be sampled on a discrete grid, we must also consider techniques for minimizing sampling artifacts from the resultant image.

GLOSSARY

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.

Pixel: A 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 parallelepiped.) Only those portions of the scene geometry that 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 parameterized 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., the 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 that 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.

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

BRDF: The Bidirectional Reflectance Distribution Function, which maps incident radiation (at general position and angle of incidence) to reflected exiting radiation (at general position and angle of exiting). Unitless, in $[0, 1]$.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

BTDF: The Bidirectional Transmission Distribution Function, which maps incident radiation (at general position and angle of incidence) to transmitted exiting radiation (at general position and angle of exiting). Analogous to the BRDF.

Radiance: The fundamental quantity in image synthesis, which is conserved along a ray traveling through a nondispersive 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 that depend only on the surface normal and the size and position of all other surfaces and light sources, and that 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 spherical/hemispherical integral of irradiance at each surface point. Ray tracing algorithms are best suited to scenes with small light sources and specular surfaces.

Hybrid algorithm: A global illumination algorithm that models both diffuse and specular interactions (e.g., [SP89]).

VISIBILITY

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 may be 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 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

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

written many times to produce the final image, wasting computation and memory bandwidth. This is known as the ***overdraw*** problem.

Four decades of spectacular improvement in graphics hardware have ensued, and high-end graphics workstations now contain hundreds of increasingly complex processors that clip, illuminate, rasterize, and texture billions 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 (Section 33.8.1).

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 at 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 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 that 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 the task is to render a scene of n polygons. If visible fragments must be rendered *exactly*, any correct algorithm must expend at least kn^2 time, 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.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

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 (Section 33.8.1). In computer graphics, hardware considerations motivate a fundamentally different approach: rendering a (judiciously chosen) superset of those polygons that 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

In previous decades, several hybrid object-space/screen-space visibility algorithms emerged (e.g., [GKM93]). As general-purpose processors became faster, such hybrid algorithms became more widely used. In certain situations, these algorithms operated entirely in object space, without relying on special-purpose graphics hardware [CT96]. In the past decade, the arrival of high powered graphics processing units (GPUs) from companies such as nVidia have allowed programming to occur at the level of programmable shaders at the vertex, geometry and pixel level. These shaders tend to be largely driven by hardware considerations but give hybrid solutions. Vertex shaders operate at the level of the vertex and can do the calculations necessary to properly light the vertex and, as necessary transform the neighborhood of the vertex. Geometry shaders operate at the level of the geometric primitive, typically the triangle, and can perform calculations at this level. Vertices and geometries are then scan-converted to provide information for the pixel shader which operates solely in image space and provides a rendering of the scene.

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 the color and intensity of the incident and exiting 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.

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 on 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) that 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, and 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, under some conditions, cause important energy-bearing paths to be overlooked. For example, a bright light source (such as the sun) filtering through many parts of a house to reach an interior space may be incorrectly discounted by this condition.

In recent years, a hardware trend has developed in support of "programmable shading," in which a (typically short, straight-line) program can be downloaded into graphics hardware for application to every vertex, polygon or pixel processed.[1] This trend has spurred research into, for example, ways to "factor" complex shading calculations into suitable components for mapping to hardware.

ALIASING

From a purely physical standpoint, the amount of energy leaving a surface in a particular direction is the product of the spherical integral of incoming energy and the bidirectional reflectance (and transmittance, as appropriate) in the exiting 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 explore 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 function's 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 that 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***.

[1] Current manufacturers include NVIDIA `http://nvidia.com/` and AMD `http://amd.com/`.

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 that 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.

SAMPLING

Sampling patterns can arise from a regular grid (e.g., pixels in a viewport) but these lead to a stair-stepping 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; otherwise aliasing (including Moiré patterns) will result.

DISCREPANCY

The quality of sampling patterns can be evaluated with a measure known as ***discrepancy*** (Chapter 47). 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. See also Chapter 13.

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 discrepancies growing as $O(N^{-1}\log^{k-1} N)$. 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

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

the unit square, modeling 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 in 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

$$|\sharp(S \cap S^+)/\sharp(S) - \text{Area}(S^+)|\,,$$

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.

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 that pass through two points of S, 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 $O(n^2 \log n)$ 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 28 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 $O(n^2 \log n)$, 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 $O(n \log^2 n)$.

The model given above can be generalized to compute ***bichromatic discrepancy***. In this case, we have sample points that 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 VC-dimension (see Section 47.1).

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

OPEN PROBLEMS

An enormous literature of adaptive, backward, forward, distribution, etc. ray tracers has evolved to address sampling and bias errors. However, the fundamental issues can be stated simply. (Each of the problems below assumes a geometric model consisting of n polygons.)

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.

The problems below are open for both the unit cube and unit ball in all dimensions.

1. 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 $O(nk^{1/2} \log\ n)$ time [SO92]. Although specialized results have been obtained, optimality has not been reached in many cases. See Table 33.8.1.

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 (resp., 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 $V(p)$ that returns 1 iff point $p \in P$ 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 $V(p, q)$ that returns 1 iff point $p \in P$ is visible from point $q \in Q$.

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.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

9. 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.

10. Given a geometric model M, a collection of light sources L, and a threshold ϵ, identify *all* optical paths bearing radiance greater than ϵ.

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

12. Given N, generate a minimum-discrepancy pattern of N samples.

13. Given a low-discrepancy pattern of K points, generate a low (or lower) discrepancy pattern of $K + 1$ points.

52.5 FURTHER CHALLENGES

We have described several core problems of computer graphics and illustrated 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 combinatorial aspects are perhaps less obvious, but for which interaction may be equally fruitful.

INDEX AND SEARCH

The proliferation of geometric models leads to a problem analogous to that in document storage: how to index models so that they can be efficiently found later. In particular, we might wish to define the Google of 3D models. Searching by name is of limited utility, since in many cases a model's author may not have named it suggestively, or as expected by the seeker. Searching by attributes or appearance is likely to be more fruitful or at the least, a necessary adjunct to searching by name. Perhaps the most successful search mechanisms to date are those relying on geometric "shape signatures" of objects, along with name and attribute metadata where available [FMK+03]. One promising class of signatures related to the medial axis transform is the "shock graph" [LK01]. A first step toward building such a system appears in [OFCD02].

TRANSMISSION AND LEVEL OF DETAIL

Fast network connectivity is not yet universally deployed, and the number and size of available models is growing inexorably with time. Thus in many contexts it is important to store, transmit, and display geometric information efficiently. A variety of techniques have been developed for "progressive" [Hop97] or "multiresolution" geometry representation [GSS99], as well as for automated level-of-detail generation from source objects [GH97]. For specific model classes, e.g., terrain,

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

efficient algorithms have been developed for varying the fidelity of the display across the field of view [BD98]. Finally, some practitioners have proposed techniques to choose levels of detail, within some time rendering budget, to optimize some image quality criterion [FKST96].

OPEN PROBLEM

Robust simplification. Cheng et al. [CDP04] recently gave a method for computing levels of details that preserve the genus of the original surface. Combine their techniques with techniques for robust computation to derive a robust and efficient scheme for simplification that can be easily implemented. See Chapter 45.

INTERACTION

In addition to off-line or batch computations, graphics practitioners develop on-line computations which involve a user in an interactive loop with input and output (display) stages, such as scientific visualization. For responsiveness, such applications may have to produce many outputs per second: rendering applications typically must maintain 30Hz or faster, whereas haptic or force-feedback applications may operate at 10KHz. Modern applications must also cope with large datasets, only parts of which may be memory-resident at any moment. Thus effective techniques for indexing, searching, and transmitting model data are required. For out-of-core data, predictive fetching strategies are required to avoid high-latency "hiccups" in the user's display.

Beyond seeing and feeling virtual representations of an object, new "3D printing" techniques have emerged for rapid prototyping applications that create real, physical models of objects. Computational geometry algorithms are required to plan the slicing or deposition steps needed. Also, "augmented reality" (AR) methods attempt to provide synthetically generated image overlays onto real scenes, for example using head-mounted displays or hand-held projectors. AR methods require good, low-latency 6-DOF tracking of the user's head or device position and orientation in extended environments.

An exciting new class of "pervasive computing" and "mobile computing" applications attempts to move computation away from the desktop and out into the extended work, home, or outdoor environment. These applications are by nature integrative, encompassing geometric and functional models, position and orientation tracking, proximity data structures, ad hoc networks, and distributed self-calibration algorithms [PMBT01].

OPEN PROBLEM

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])?

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

DYNAMICS

When simulations include objects that affect each other through force exchange or collision, they must efficiently identify the actual interactions. Usually there is significant temporal coherence, i.e., the set of objects near a given object changes slowly over time. A number of techniques have been proposed to track moving objects in a spatial index or closest-pair geometric data structure in order to detect collisions efficiently [MC95, LMC94, BGH99]. The "object" of interest may be the geometric representation of a user, for example of a finger or hand probing a virtual scene. Recently, some authors have proposed synthesizing sound information to accompany the visual simulation outputs [OSG02].

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. It is a challenge to determine how 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.

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 geometric representations and algorithms are necessary to capture and model such phenomena [DPH96]. See Chapter 53.

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 scores or 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 Section 50.1). 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.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

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 per word). 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 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.

52.6 SOURCES AND RELATED MATERIAL

SURVEYS

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

[Dob92]: A survey article on computational geometry and computer graphics.

[Dor94]: Survey of object-space hidden-surface removal algorithms.

[Yao92, LP84]: Surveys of computational geometry.

[CCSD03]: Survey of visibility for walkthroughs.

RELATED CHAPTERS

Chapter 13: Geometric discrepancy theory and uniform distribution
Chapter 16: Subdivisions and triangulations of polytopes
Chapter 30: Polygons
Chapter 33: Visibility
Chapter 39: Collision and proximity queries
Chapter 41: Ray shooting and lines in space
Chapter 50: Algorithmic motion planning
Chapter 56: Splines and geometric modeling
Chapter 45: Robust geometric computation
Chapter 57: Solid modeling

REFERENCES

[AB99] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete Comput. Geom.*, 22:481–504, 1999.

[App68] A. Appel. Some techniques for shading machine renderings of solids. In *Proc. Spring Joint Computer Conference*, pages 37–45, ACM Press, 1968.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

[BD98] M. de Berg and K. Dobrindt. On levels of detail in terrains. *Graphical Models Image Proc.*, 60:1–12, 1998.

[BGH99] J. Basch, L.J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31:1–28, 1999.

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

[Cat74] E.E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph.D. thesis, Univ. Utah, TR UTEC-CSc-74–133, 1974.

[CCSD03] D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE. Trans. Visualization Comput. Graphics*, 9:412–431, 2003.

[CDP04] S.-W. Cheng, T.K. Dey, and S.-H. Poon. Hierarchy of surface models and irreducible triangulation. *Comput. Geom.*, 27:135–150, 2004.

[CL96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. 23rd ACM Conf. SIGGRAPH*, pages 303–312, 1996.

[CPC84] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18:137–145, 1984.

[CT96] S. Coorg and S. Teller. Temporally coherent conservative visibility. *Comput. Geom.* 12,105–124, 1999.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P.K. Agarwal, F.P. Brooks, Jr., and W.V. Wright. Simplification envelopes. In *Proc. 23rd ACM Conf. SIGGRAPH*, pages 119–128, 1996.

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

[DEM93] D.P. Dobkin, D. Eppstein, and D.P. Mitchell. Computing the discrepancy with applications to supersampling patterns. In *Proc. 9th Sympos. Comput. Geom.*, pages 47–52, ACM Press, 1993.

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

[Dor94] S.E. 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 *Proc. 23rd ACM Conf. SIGGRAPH*, pages 411–420, 1996.

[DT04] D.P. Dobkin and S. Teller. Computer graphics. In J.E. Goodman and J. O'Rourke, editors, *CRC Handbook of Discrete and Computational Geometry*, 2nd edition, CRC Press, Boca Raton, 2004.

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

[FMK+03] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D.P. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Trans. Graph.*, 22:83–105, 2003.

[GH97] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Proc. 24th ACM Conf. SIGGRAPH*, pages 209–216, 1997.

[GKM93] N. Greene, M. Kass, and G.L. Miller. Hierarchical Z-buffer visibility. In *Proc. 20th ACM Conf. SIGGRAPH*, pages 231–238, 1993.

[GSS99] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proc. 26th ACM Conf. SIGGRAPH*, pages 325–334, 1999.

[GSV16] Understand Google Street View, `https://www.google.com/maps/streetview/understand/`, retrieved February 7, 2016.

[HDD+92] H. Hoppe, T.D. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. 19th ACM Conf. SIGGRAPH*, pages 71–78, 1992.

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *Proc. 24th ACM Conf. SIGGRAPH*, pages 189–198, 1997.

[HSA91] P. Hanrahan, D. Salzman, and L.J. Aupperle. A rapid hierarchical radiosity algorithm. In *Proc. 18th ACM Conf. SIGGRAPH*, pages 197–206, 1991.

[HW91] E. Haines and J.R. Wallace. Shaft culling for efficient ray-traced radiosity. In *Proc. 2nd Eurographics Workshop Rendering*, pages 122–138, 1991.

[Kaj86] J.T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20:143–150, 1986.

[LK01] F.F. Leymarie and B.B. Kimia. The shock scaffold for representing 3D shape. In *Proc. 4th Internat. Workshop Visual Form*, pages 216–229. Springer-Verlag, Berlin, 2001.

[LMC94] M.C. Lin, D. Manocha, and J.F. Canny. Fast contact determination in dynamic environments. In *Proc. Internat. Conf. Robot. Autom.*, pages 602–609, 1994.

[LP84] D.T. Lee and F.P. Preparata. Computational geometry: A survey. *IEEE Trans. Comput.*, 33:1072–1101, 1984.

[LPC+00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proc. 27th ACM Conf. SIGGRAPH*, pages 131–144, 2000.

[Mas12] Machable. 11 Fascinating Facts About Google Maps, `http://mashable.com/2012/08/22/google-maps-facts/#OCzys4bUBkqO`, retrieved February 3, 2016.

[MB95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proc. 22nd ACM Conf. SIGGRAPH 95*, pages 39–46, 1995.

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

[MP96] R. Mech and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *Proc. 23rd ACM Conf. SIGGRAPH*, pages 397–410, 1996.

[OFCD02] R. Osada, T. Funkhouser, B. Chazelle, and D.P. Dobkin. Shape distributions. *ACM Trans. Graph.*, 21:807–832, 2002.

[OSG02] J. O'Brien, C. Shen, and C. Gatchalian. Natural phenomena: Synthesizing sounds from rigid-body simulations. *Proc. ACM SIGGRAPH Sympos. Computer Animation*, 2002.

[Per85] K. Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19:287–296, 1985.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

[PMBT01] N. Priyantha, A. Miu, H. Balakrishnan, and S. Teller. The cricket compass for context-aware mobile applications. In *Proc. 7th ACM Conf. Mobile Comput. Network*, pages 1–14, 2001.

[RHHL02] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3D model acquisition. In *Proc. 29th ACM Conf. SIGGRAPH*, pages 438–446, 2002.

[SDSA93] C. Schoeneman, J. Dorsey, B. Smits, and J. Arvo. Painting with light. *Proc. 20th ACM Conf. SIGGRAPH*, pages 143–146, 1993.

[SF95] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proc. 22nd ACM Conf. SIGGRAPH*, pages 129–136, 1995.

[SO92] M. Sharir and M.H. Overmars. A simple output-sensitive algorithm for hidden surface removal. *ACM Trans. Graph.*, 11:1–11, 1992.

[SP89] F.X. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. *SIGGRAPH Comput. Graph.*, 23:335–344, 1989.

[SSS74] I.E. Sutherland, R.F. Sproull, and R.A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Comput. Surv.*, 6:1–55, 1974.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, Berlin, 2010.

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

[VG95] E. Veach and L.J. Guibas. Optimally combining sampling techniques for Monte Carlo rendering. In *Proc. 22nd ACM Conf. SIGGRAPH*, pages 419–428, 1995.

[Whi80] T. Whitted. An improved illumination model for shading display. *Commun. ACM*, 23:343–349, 1980.

[Yao92] F.F. Yao. Computational geometry. In *Algorithms and Complexity, Handbook of Theoretical Computer Science*, volume A, Elsevier, Amsterdam, pages 343–389, 1992.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.