Visibility, problems, techniques and applications

Course Notes for SIGGRAPH 2001
Los Angeles, California
August 2001

Organizer:

Daniel Cohen-Or	Tel-Aviv University

Speakers:

Yiorgos Chrysanthou	UCL University of London
Daniel Cohen-Or	Tel-Aviv University
Fredo Durand	Massachusetts Institute of Technology
Ned Greene	NVIDIA
Vladlen Kulton	Tel-Aviv University
Cláudio T. Silva	AT&T Labs-Research

Abstract

Visibility determination, the process of deciding what surfaces can be seen from a certain point, is one of the fundamental problems in computer graphics. Its importance has long been recognized, and in network-based graphics, virtual environments, shadow determination, global illumination, culling, and interactive walkthroughs, it has become a critical issue. This course reviews fundamental issues, current problems, and unresolved solutions, and presents an in-depth study of the visibility algorithms developed in recent years. Its goal is to provide students and graphics professionals (such as game developers) with effective techniques for visibility culling.

Schedule

8:30	–	8:40	Introduction to the course	Cohen-Or
8:40	–	9:40	Analytical Visibility	Durand
9:40	–	10:00	Hierarchical techniques for culling	Chrysanthou
10:00	–	10:15	Coffee break	
10:15	–	10:40	Hierarchical techniques for culling (cont)	Chrysanthou
10:40	–	12:00	Object Space techniques	Silva
12:00	–	1:30	Lunch break	
1:30	–	2:15	Image Space Methods	Greene
2:15	–	3:00	Viewspace part. and from-region visibility	Cohen-Or
3:00	–	3:15	Coffee break	
3:15	–	4:20	Area Visibility	Koltun
4:20	–	5:00	Hardware-Assisted Occlusion-Culling	Greene
5:00			Final Questions and Discussion	

Speaker Biographies

Daniel Cohen-Or
Computer Science Department
School of Mathematical Sciencees
Tel-Aviv University, Ramat-Aviv 69978, Israel
E-mail: daniel@math.tau.ac.il

Daniel Cohen-Or is an Associate Professor at the Department of Computer Science since 1995. He received a B.Sc. cum laude in both Mathematics and Computer Science (1985), an M.Sc. cum laude in Computer Science (1986) from Ben-Gurion University, and a Ph.D. from the Department of Computer Science (1991) at State University of New York at Stony Brook. He has been a lecturer at the Department of Mathematics and Computer Science of Ben Gurion University in 1992-1995. He is on the editorial advisory board of the international Computers and Graphics journal. He was the Program Co-chair of the symposium on volume visualization 2000. He is a member of the program committees of several international conferences on visualization and computer graphics, including IEEE Visualization, Eurographics and the Eugraphics rendering workshop. He is board member in a number of journals including IEEE TVCG, TVC, C&G, CGF. Between 1996-8 he served as the Chairman of the Central Israel SIGGRAPH Chapter. Prof. Cohen-Or has a rich record of industrial collaboration. In 1992-93 he developed a real-time flythrough with Tiltan Ltd. and IBM Israel for the Israeli Air Force. During 1994-95 he worked on the development of a new parallel architecture at Terra Ltd. In 1996-1997 he has been working with MedSim Ltd. on the development of an ultrasound simulator. He is the inventor of RichFX and Enbaya streaming technologies. His research interests are in Computer Graphics, and include rendering techniques, client/server 3D graphics applications, real-time walkthroughs and flythroughs, volume graphics, architectures and algorithms for voxel-based graphics.

Yiorgos Chrysanthou
Department of Computer Science
UCL University of London
Gower Street, London WC1E 6BT, UK
E-mail: Y.Chrysanthou@cs.ucl.ac.uk

Yiorgos Chrysanthou is a lecturer at the Computer Science Department of University College London, UK. He received his BSc in Computer Science and Statistics (1990, 1st Class Honours) and his PhD in Computer Graphics (1996) from Queen Mary and Westfield College. During the period of 1996-98 he worked as a research fellow on the COVEN (Collaborative Virtual ENvironments) ACTS project where he concentrated on real-time rendering of large scale virtual environments. Since then he is the principle investigator of a number of funded projects dealing with real-time graphics and their applications. Recently he co-edited, along with Daniel Cohen-Or, a special in the journal Computers and Graphics with theme Visibility, techniques and Applications. His research interests include real-time rendering, virtual reality and computational geometry.

Fredo Durand
NE43-255, 200 Technology Sq.
Massachusetts Institute of Technology
Cambridge, MA 02139
E-mail: fredo@graphics.lcs.mit.edu

Fredo Durand is a postdoctoral associate in the Lab for Computer Science at MIT. He received his Ph.D. in Computer Science from Grenoble University (France) in 1999 with honors. His Ph.D. work on visibility lead to 2 SIGGRAPH articles and other publications in ACM ToG and EGWR. His thesis contains a comprehensive 150 page multidisciplinary survey on visibility issues in computer graphics, computer vision, robotics and computational geometry. He has recently been invited to give the introductory talk at the Workshop on Visibility held in Corsica. His research interest include visibility, both from a theoretical and practical point of view, image-based modeling and editing, visual perception, texture synthesis, and non photorealistic rendering.

Vladlen Kulton
Computer Science Department
School of Mathematical Sciences
Tel-Aviv University, Ramat-Aviv 69978, Israel
E-mail: vladlen@tau.ac.il

Vladlen is a Ph.D. student in Tel-Aviv University, concentrating on visibility problems in computer graphics and computational geometry. He is directly involved with real-time computer graphics for over seven years. He produced prize-winning multimedia productions, and has lead a team that created a commercial real-time 3D game. As a senior software developer in Shells Interactive Ltd., he took part in the design and implementation of the ”3D Dreams” 3D engine. As a graduate student, Vladlen’s main interest is on visibility computation and its applications to computer graphics on the Internet. He is now working on harnessing results and insights from computational geometry to obtain practical solutions for visibility problems in computer graphics, and to develop new intuitive ways of perceiving and thinking about visibility.

Ned Greene
Nvidia Corporation
E-mail: ned@ngreene.com

Ned Greene is currently working as an architect on graphics hardware at the Nvidia Corporation. From 1996 to 2000, Ned developed occlusion-culling methods for graphics hardware and consulted at Hewlett-Packard Laboratories and Apple Computer. From 1989 to 1996 he was a graphics researcher in the Advanced Technology Group at Apple Computer where his work included hierarchical approaches to occlusion culling and rendering. Ned was a member of the research staff at the NYIT Computer Graphics Lab from 1980 to 1989, where he conducted research, developed animation software, and contributed to pioneering animation projects. He holds a Masters Degree in computer science from the Courant Institute at New York University and a PhD in computer science from the University of California at Santa Cruz. Over the years Ned has been a frequent contributor to the Siggraph technical program and Electronic Theater.

Cláudio T. Silva
AT&T Labs-Research
Shannon Laboratory
180 Park Avenue – room D265
Florham Park, NJ 07932
E-mail: csilva@research.att.com

Cláudio T. Silva is a Senior Member of Technical Staff in the Information Visualization Research Department at AT&T Labs-Research, and an adjunct assistant professor at the State University of New York at Stony Brook. His current research is on architectures and algorithms for building scalable displays, rendering techniques for large datasets, 3D scanning, and algorithms for graphics hardware. Before joining AT&T, Claudio was a Research Staff Member at the graphics group at IBM T. J. Watson Research Center. Claudio has a Bachelor's degree in mathematics from the Federal University of Ceara (Brazil), and MS and PhD degrees in computer science from the State University of New York at Stony Brook. While a student, and later as an NSF post-doc, he worked at Sandia National Labs, where he developed large-scale scientific visualization algorithms and tools for handling massive datasets. His main research interests are in graphics, visualization, applied computational geometry, and high-performance computing. Claudio has published over 30 papers in international conferences and journals, and presented courses at ACM SIGGRAPH, Eurographics, and IEEE Visualization conferences. He is a member of the ACM, Eurographics, and IEEE.

Contents

- Cohen-Or:

- Durand:

- Silva:

- Chrysanthou:

- Greene:

- Koltun:

Visibility Problems for Walkthrough Applications

Daniel Cohen-Or

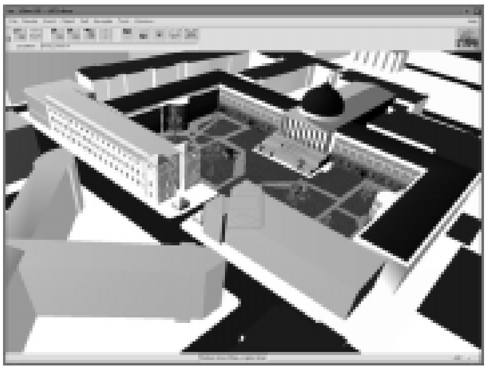
Computer Science Department

Tel-Aviv University

http://www.math.tau.ac.il/~daniel/

Virtual Reality Applications

- The user "walks" interactively in a virtual polygonal environment.
  Examples: model of a city, museum, mall, architectural design



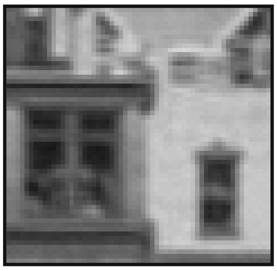
The goal: to render an updated image for each view point and for each view direction in interactive frame rate

The Model

- Composed of 3D geometric objects - Lots of simple parts
- Large and complex - hundreds thousands or even millions of polygons

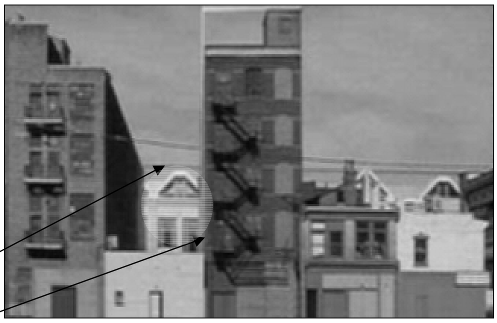
The Visibility Problem

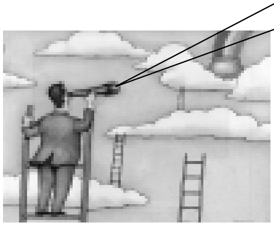
- Selecting the (exact?) set of polygons from the model which are visible from a given viewpoint



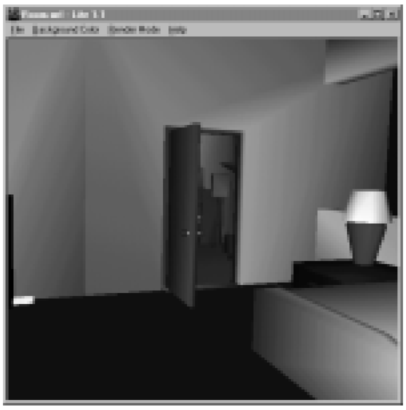
The Visibility Problem is important

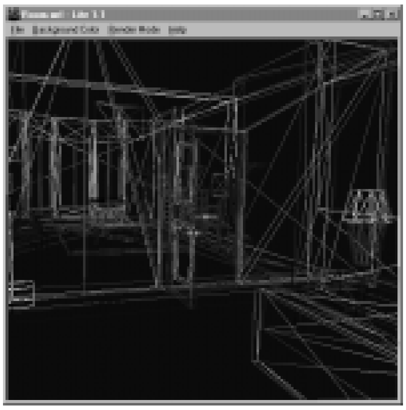
- Average number of polygons, visible from a viewpoint, is **much** smaller than the model size



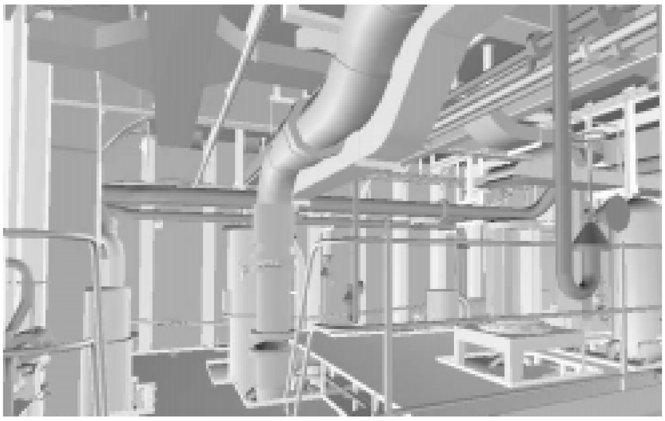


Indoor scene

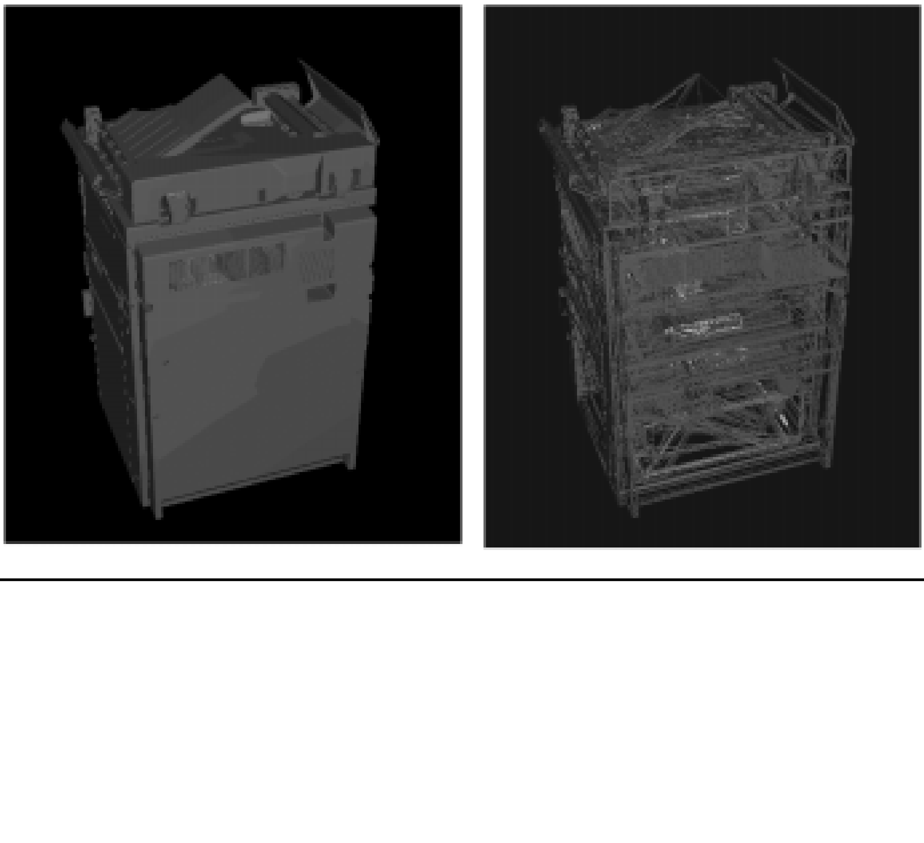




Oil-tanker ship

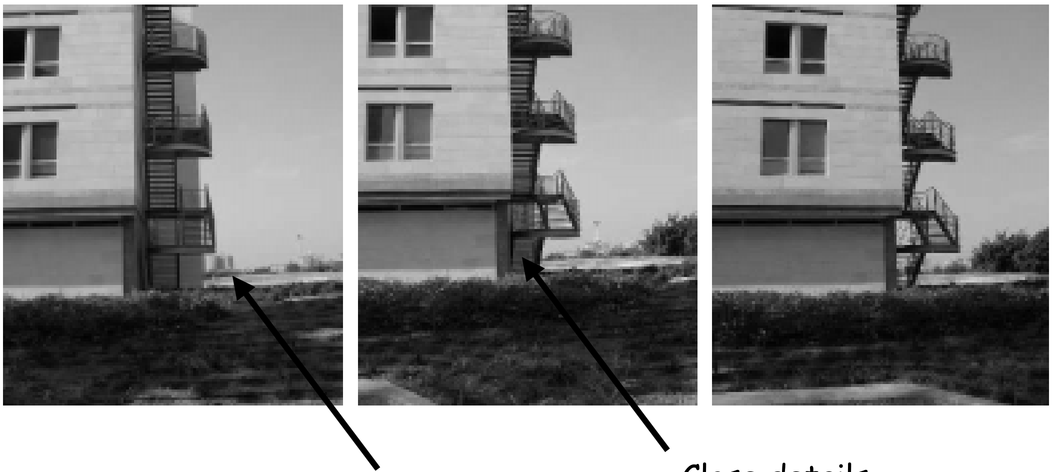


Copying Machine



The Visibility Problem is not easy...

A small change of the viewpoint might causes large changes in the visibility



Far details

Close details

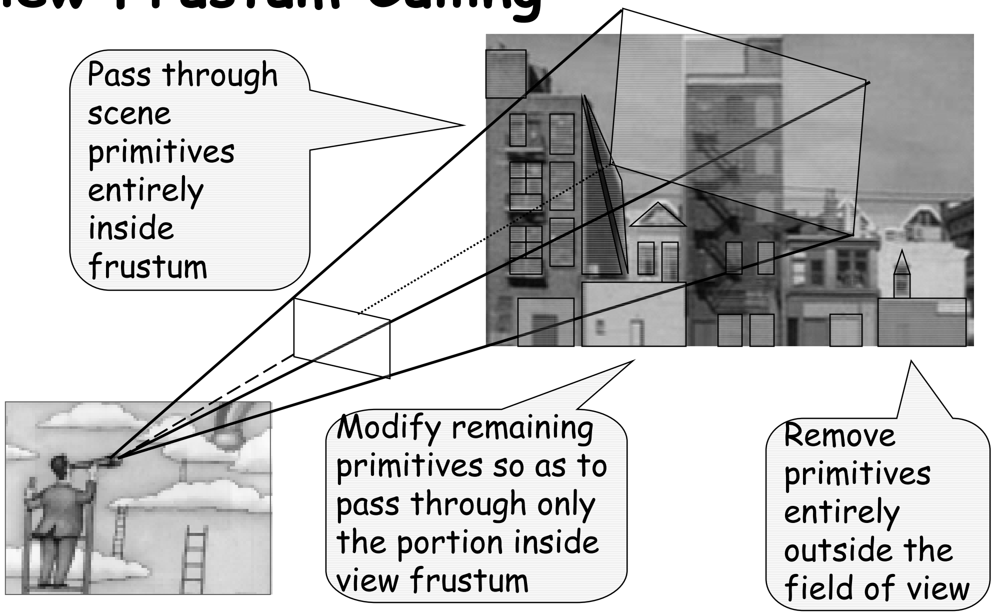
Culling

Avoid processing polygons which contribute nothing to the rendered image

A primitive can be culled by:

- View Frustum Culling
- Back Face Culling
- Occlusion Culling

View Frustum Culling



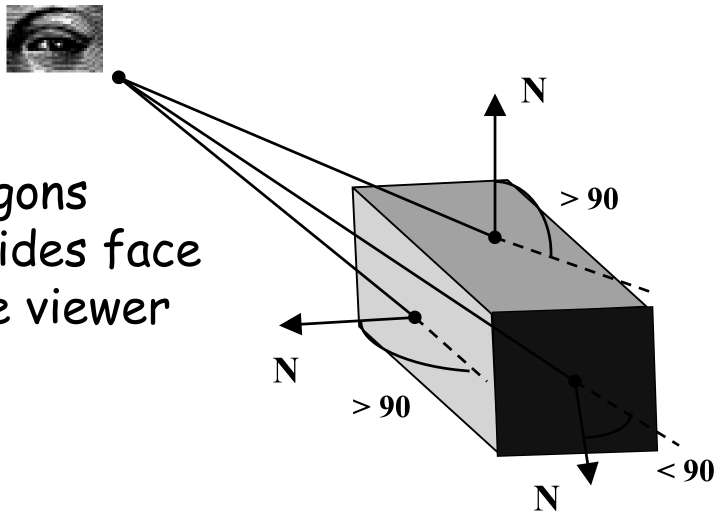
Pass through scene primitives entirely inside frustum

Modify remaining primitives so as to pass through only the portion inside view frustum

Remove primitives entirely outside the field of view

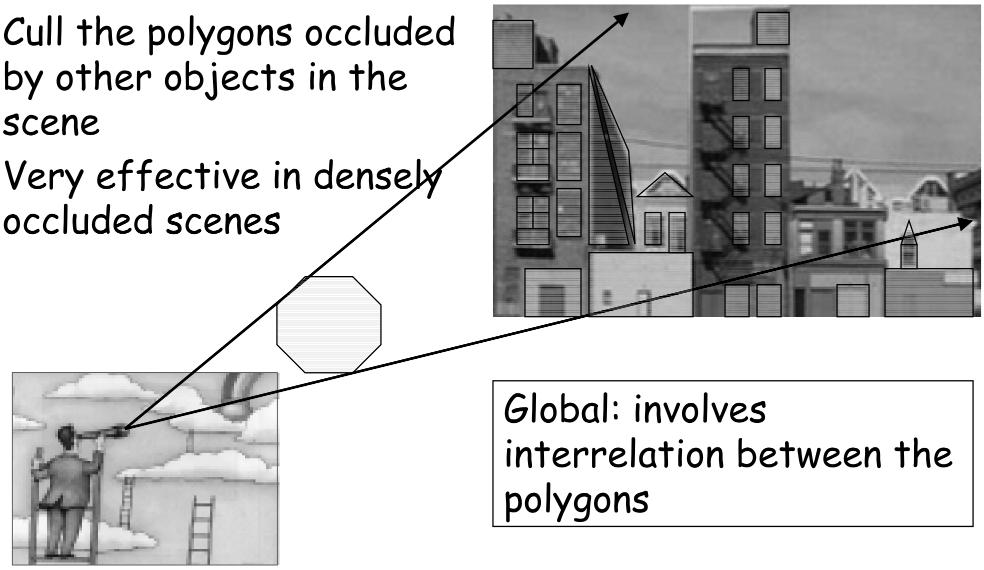
Backface Culling

cull away polygons whose front sides face away from the viewer



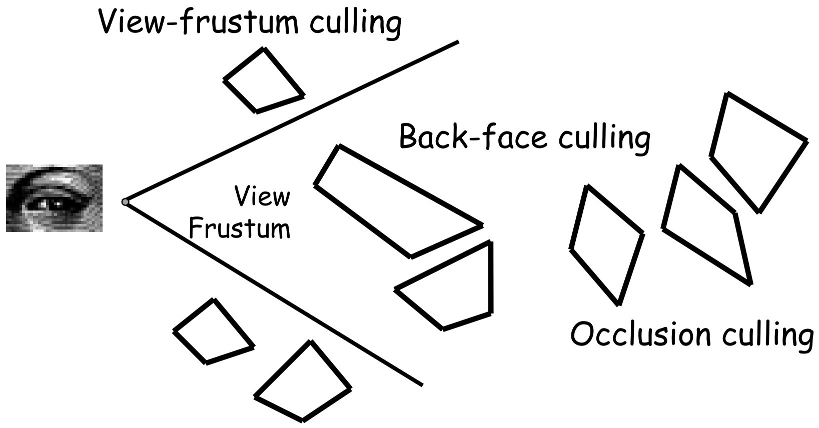
Occlusion Culling

- Cull the polygons occluded by other objects in the scene
- Very effective in densely occluded scenes



Global: involves interrelation between the polygons

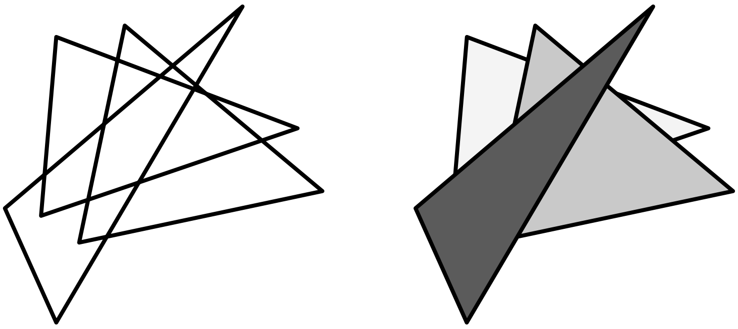
Visibility Culling



Hidden Surface Removal

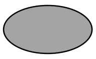
Polygons overlap, so somehow, we must determine which portion of each polygon to draw (is visible to the eye)

Output sensitive algorithms



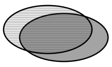
Exact Visibility

Includes all the polygons which are at least partially visible and only these polygons.



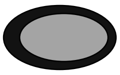
Approximate Visibility

Includes most of the visible polygons plus maybe some hidden ones.



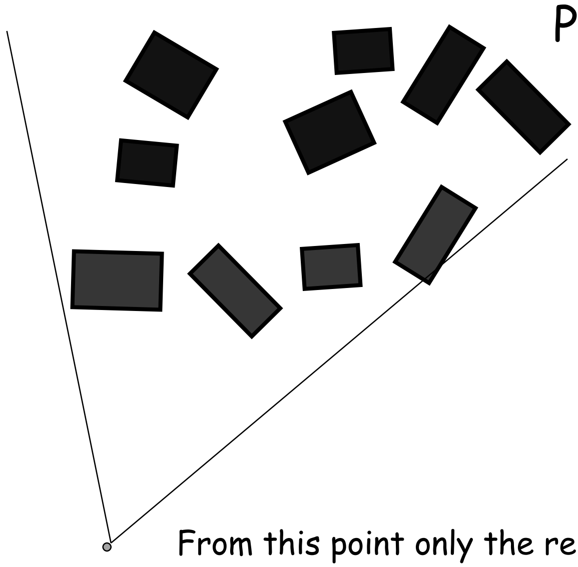
Conservative Visibility

Includes at least all the visible objects plus maybe some additional invisible objects



May classify invisible object as visible but may never classify visible object as invisible

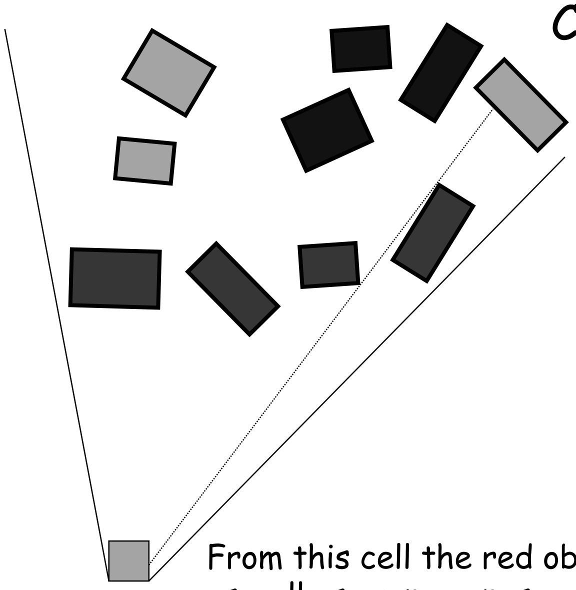
Point Visibility



From this point only the red objects are visible

Cell Visibility

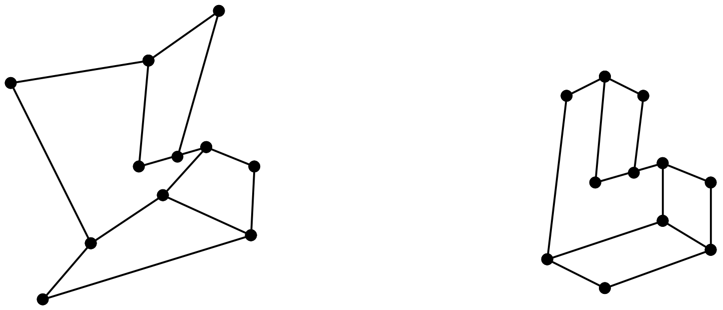
- Compute the set of all polygons visible from every possible viewpoint from a region (view-cell)



From this cell the red objects are visible as well as orange ones

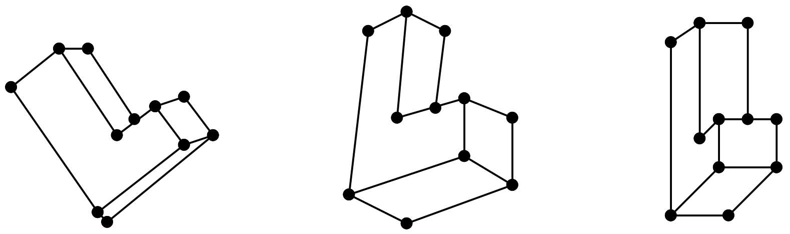
The Aspect Graph

Isomorphic graphs



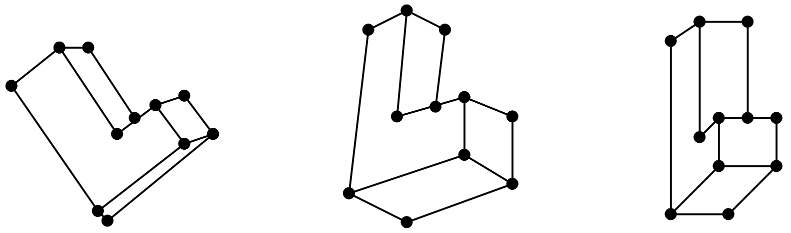
The Aspect Graph

- **ISG – Image Structure graph**
  The planner graph, defined by the outlines of an image, created by projection of a polyhedral object, in a certain view direction



The Aspect Graph (Cont.)

- **Aspect**
  Two different view directions of an object have the same <u>aspect</u> iff the corresponding Image Structure graphs are isomorphic



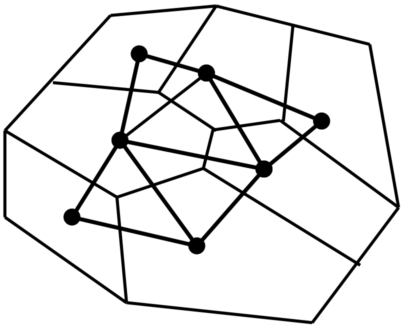
The Aspect Graph (Cont.)

- **VSP – Visibility Space Partition**
  - Partitioning the viewspace into maximal connected regions in which the viewpoints have the same view or aspect

- **Visual Event**
  A boundary of a VSP region called a VE for it marks a change in visibility

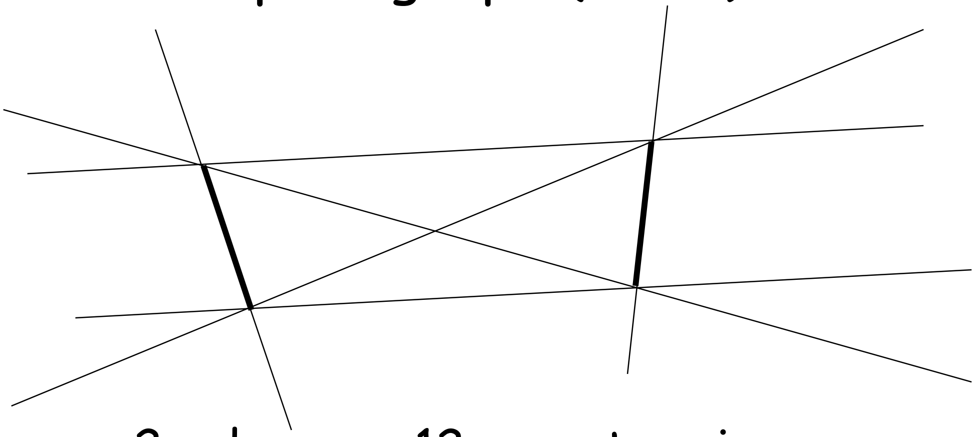
The Aspect Graph (Cont.)

- **Aspect Graph**
  - A vertex for each region of the VSP
  - An edge connecting adjacent regions



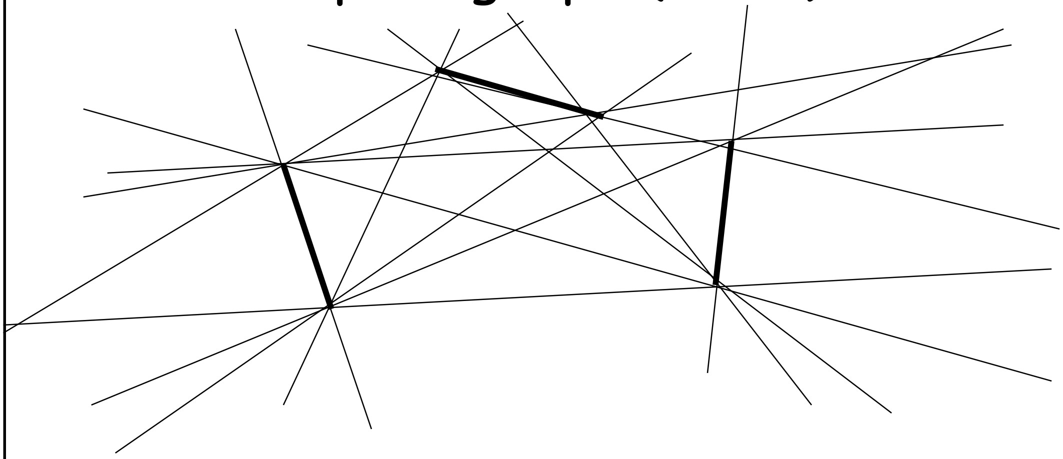
Regions of the VSP are not maximal but maximal connected regions.

Aspect graph (Cont.)



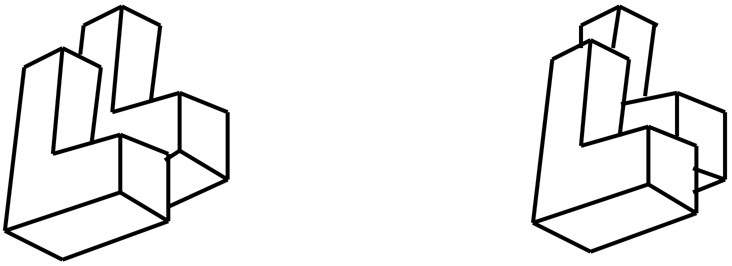
2 polygons - 12 aspect regions

Aspect graph (Cont.)

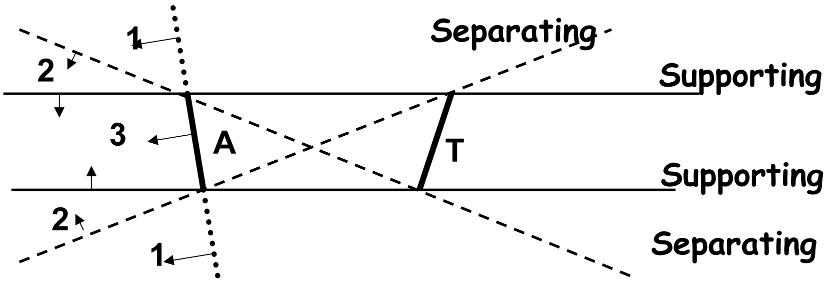


3 polygons - “many” aspect regions

Different aspect regions can have equal sets of visible polygons



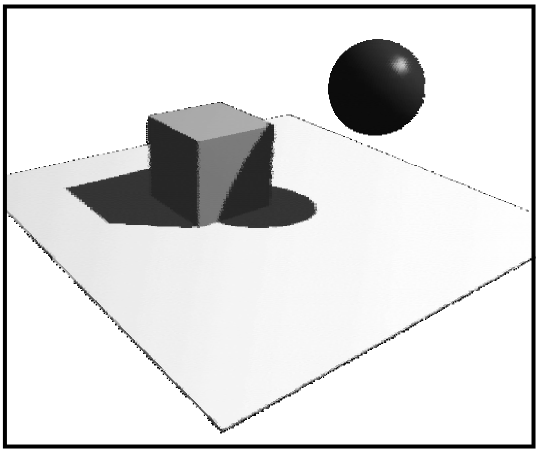
Supporting & Separating Planes



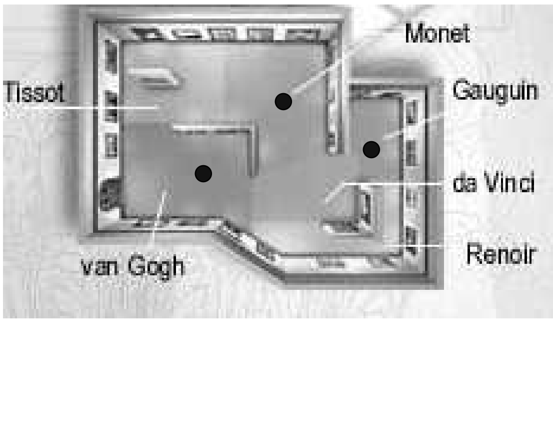
T is not occluded in region 1
T is partially occluded in region 2
T is completely occluded in region 3

A - occluder
T - occludee

Visibility from the light source



The Art Gallery Problem



See: ftp://ftp.math.tau.ac.il/pub/~daniel/pg99.pdf

Classification of visibility algorithms

- Exact vs. Approximated
- Conservative vs. Exact
- Precomputed vs. Online
- Point vs. Region
- Image space vs. Object space
- Software vs. Hardware
- Dynamic vs. Static scenes

Visibility is important and interesting

- Only a small fraction of the scene is visible from a given point.

- Small changes in the view point can cause large changes in the visibility

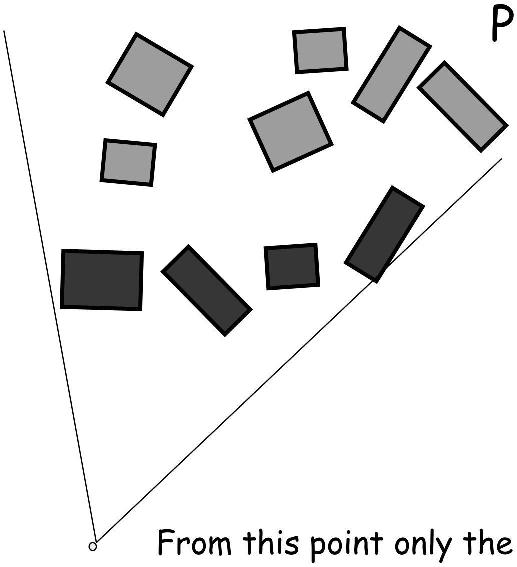
Thanks for Listening

From-region Visibility

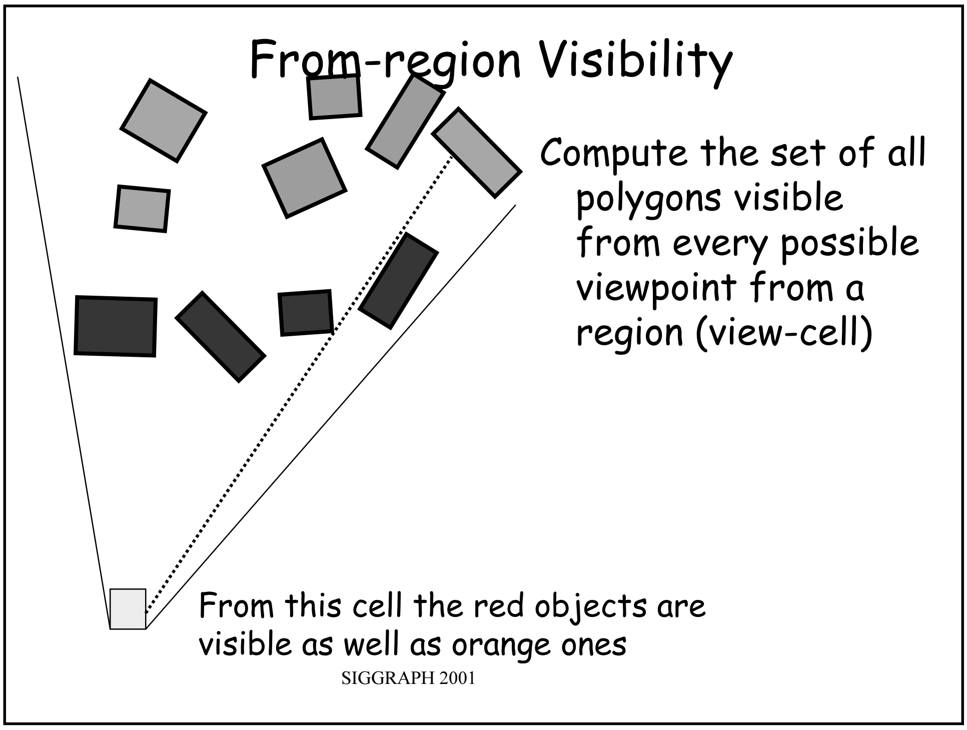
Daniel Cohen-Or
Tel-Aviv University

GTEC20001, Hong Kong

Point Visibility



From this point only the red objects are visible

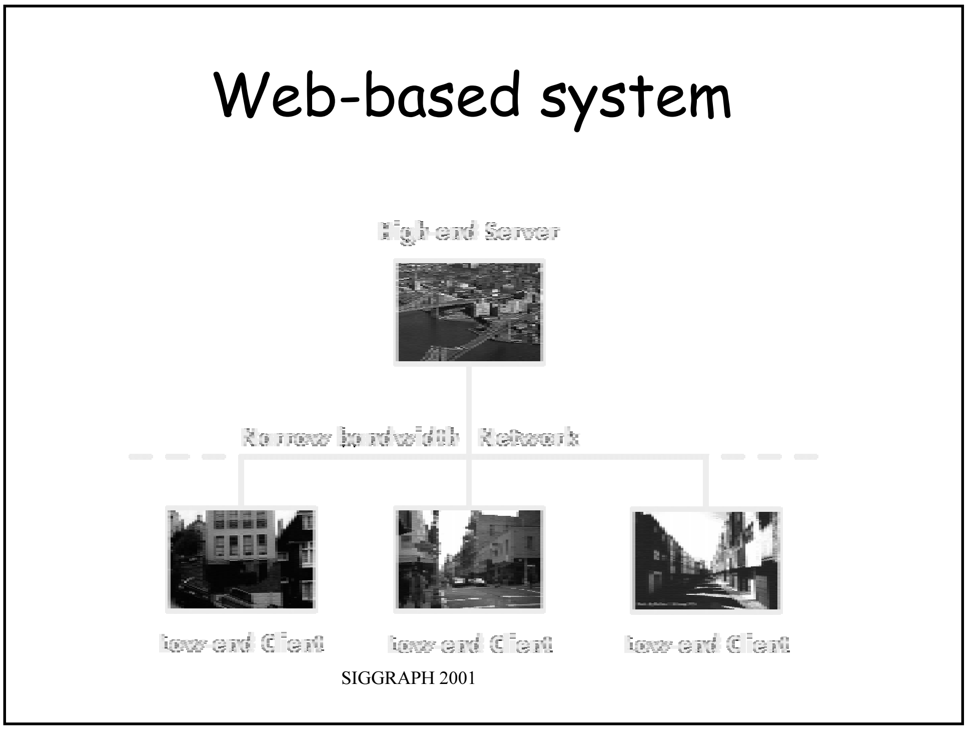


From-region Visibility

Compute the set of all polygons visible from every possible viewpoint from a region (view-cell)

From this cell the red objects are visible as well as orange ones

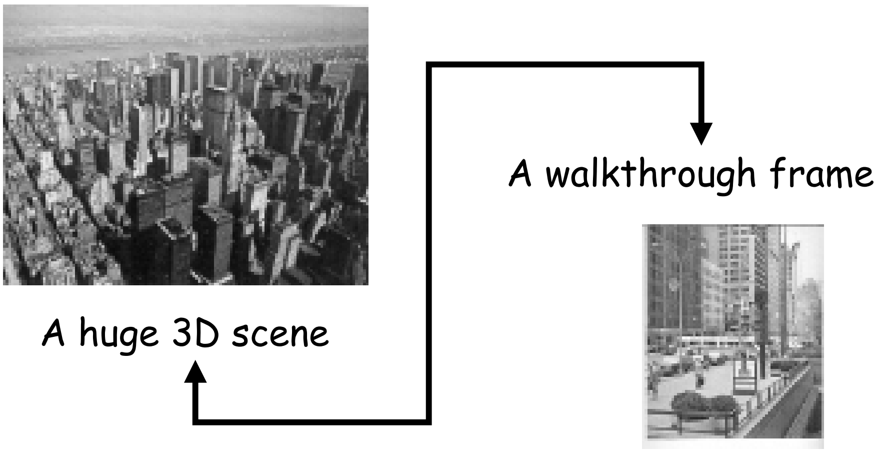
SIGGRAPH 2001



Web-based system

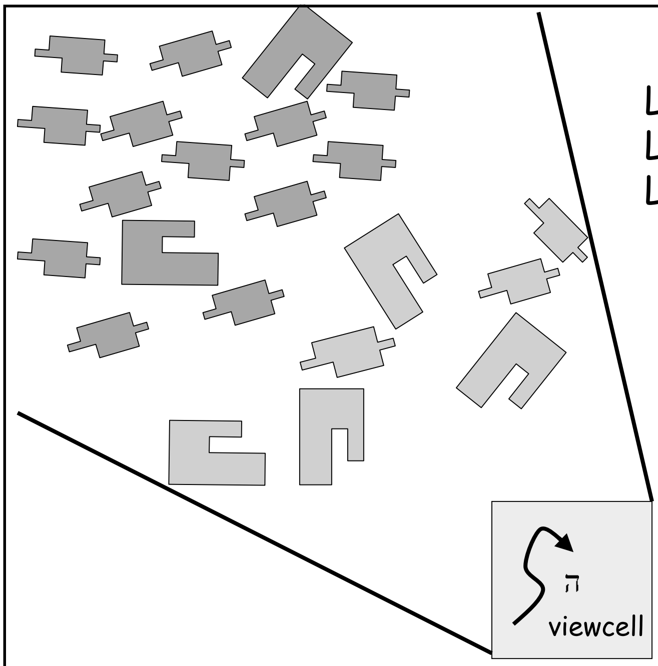
SIGGRAPH 2001

Web-based client-server system

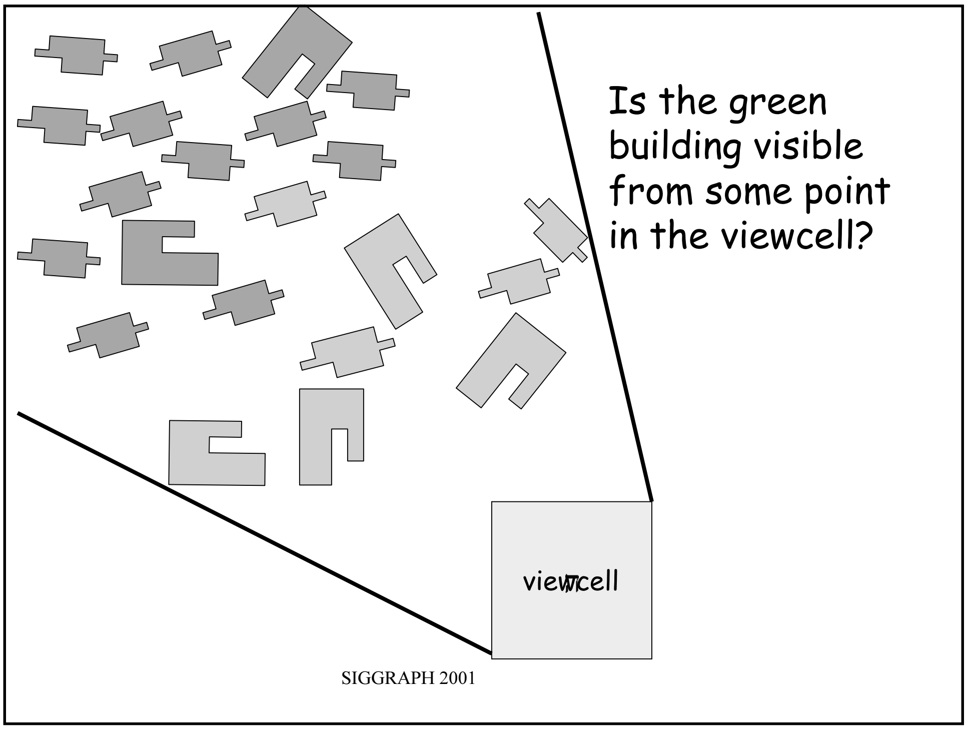


A walkthrough frame

A huge 3D scene

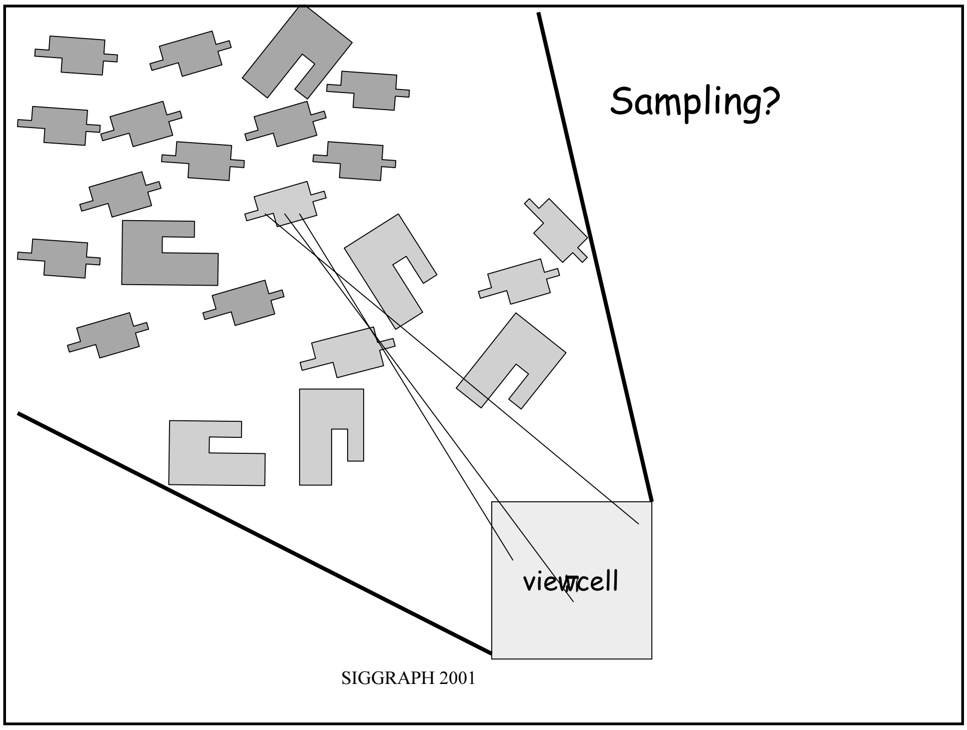


Latency!
Latency!
Latency!



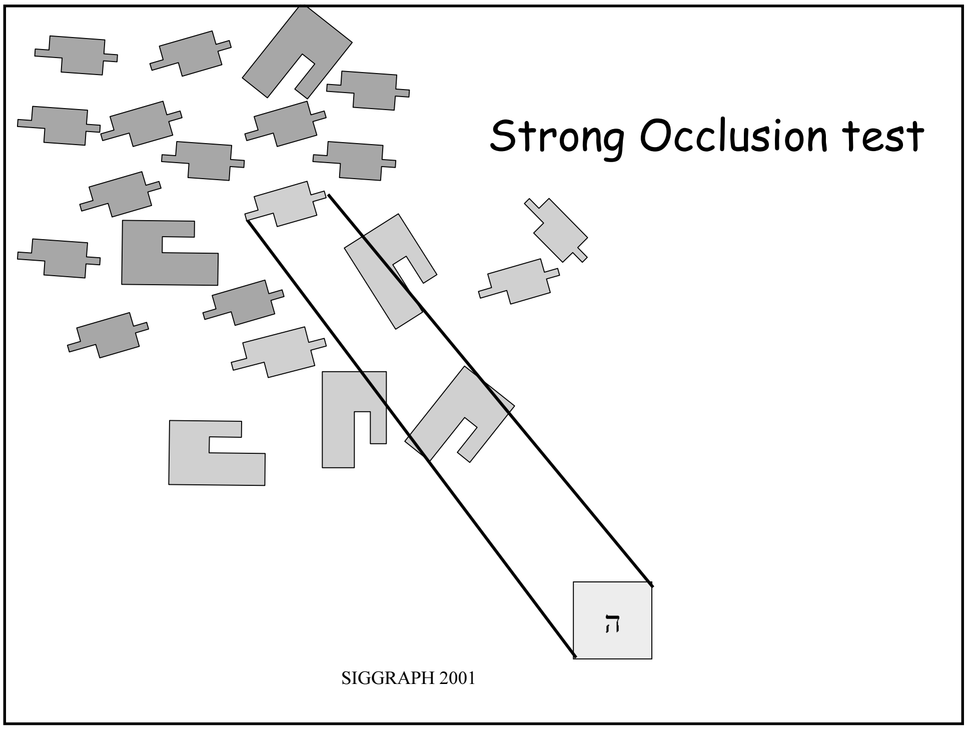
Is the green building visible from some point in the viewcell?

SIGGRAPH 2001



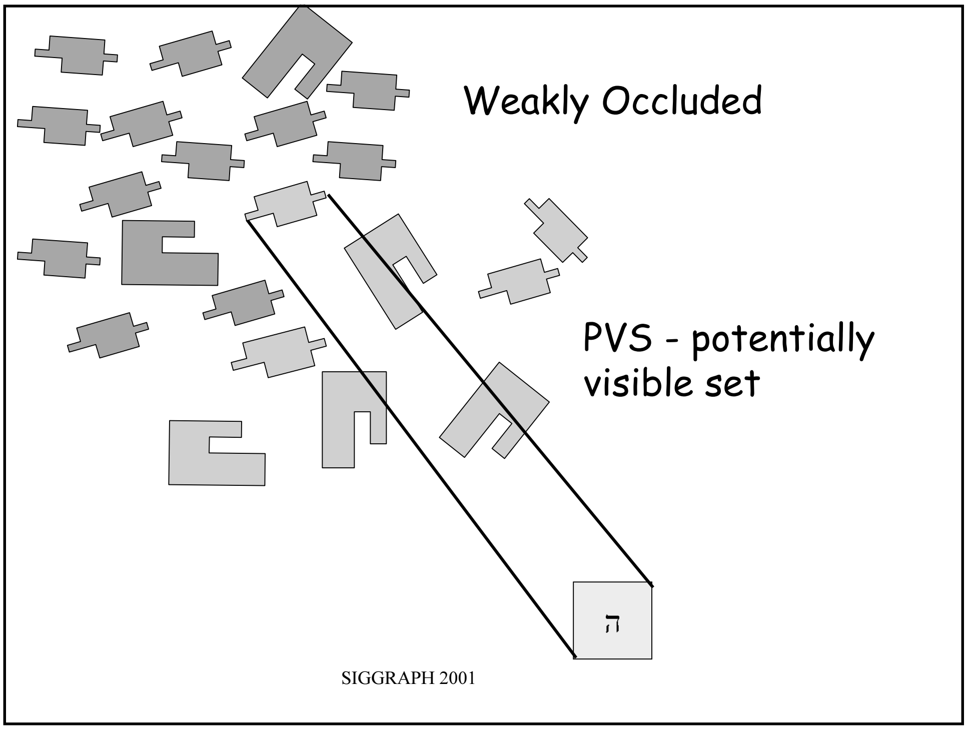
Sampling?

SIGGRAPH 2001



Strong Occlusion test

SIGGRAPH 2001

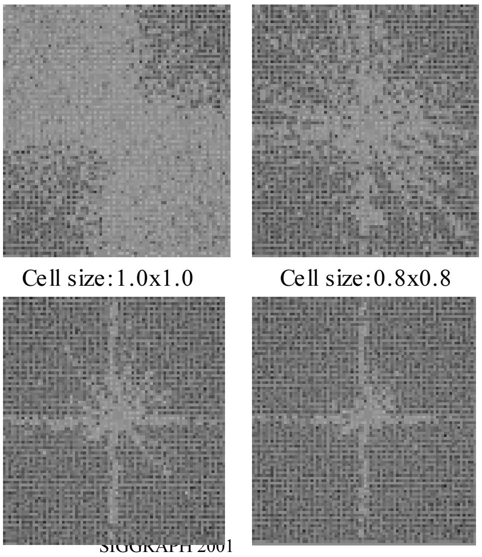


Weakly Occluded

PVS - potentially visible set

SIGGRAPH 2001

Cost effective analysis of the view cell size (Cohen-Or et al, EG98)



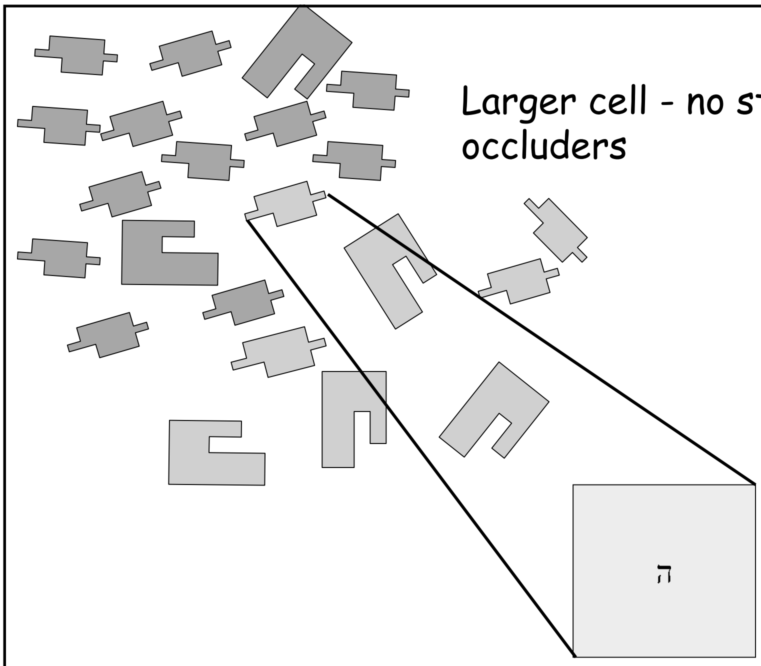
Cell size:1.0x1.0

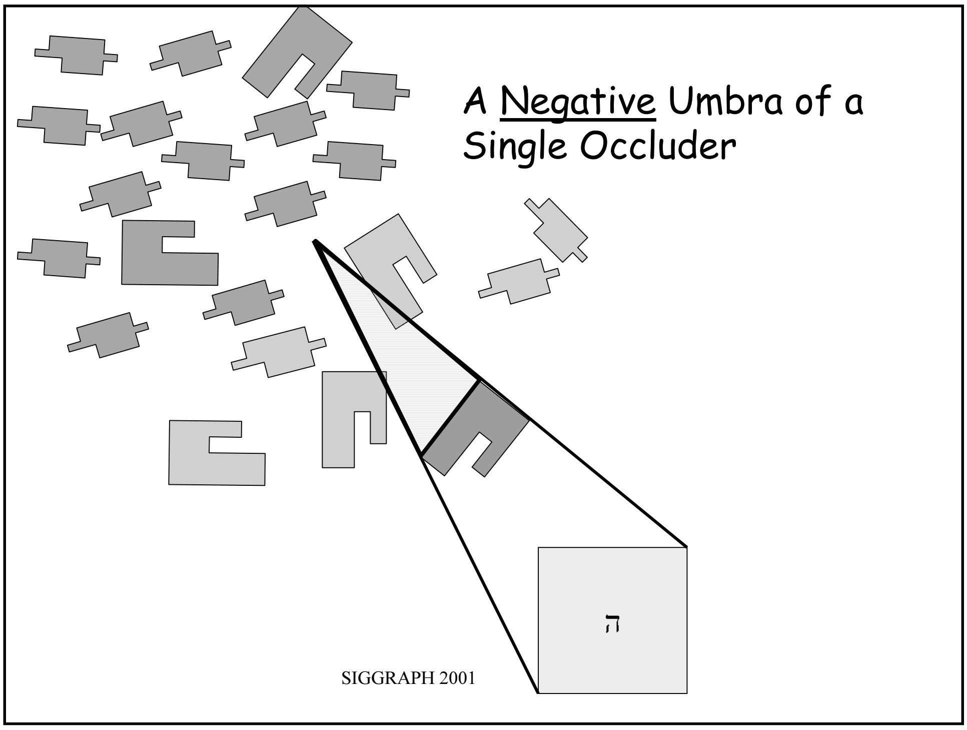
Cell size:0.8x0.8

Cell size:0.5x0.5

Cell size:0.2x0.2

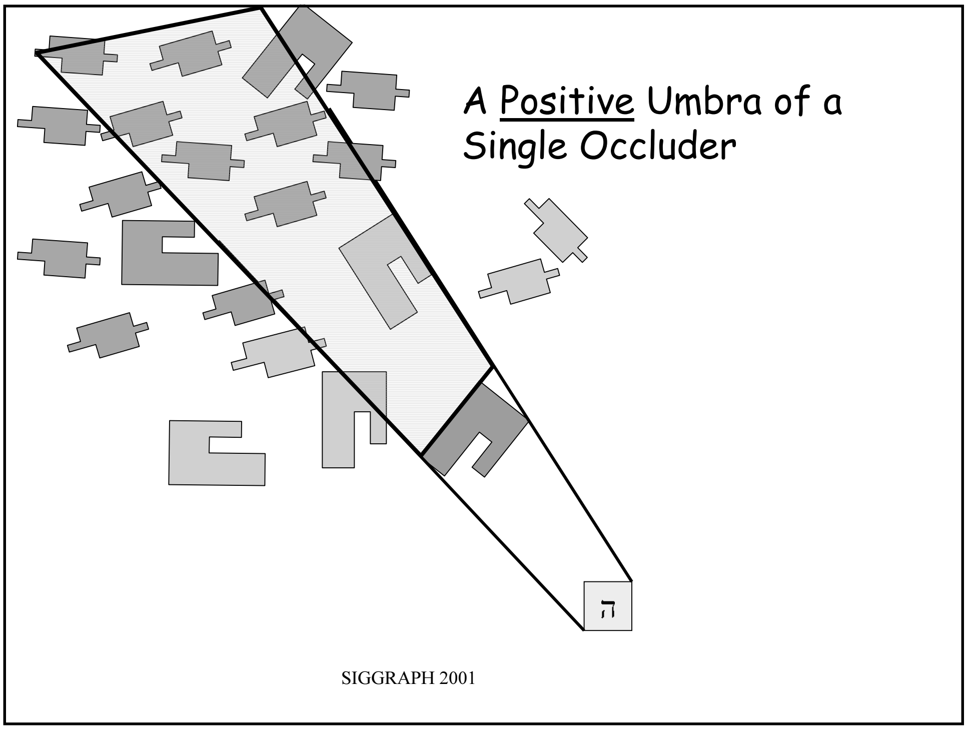
Larger cell - no strong occluders





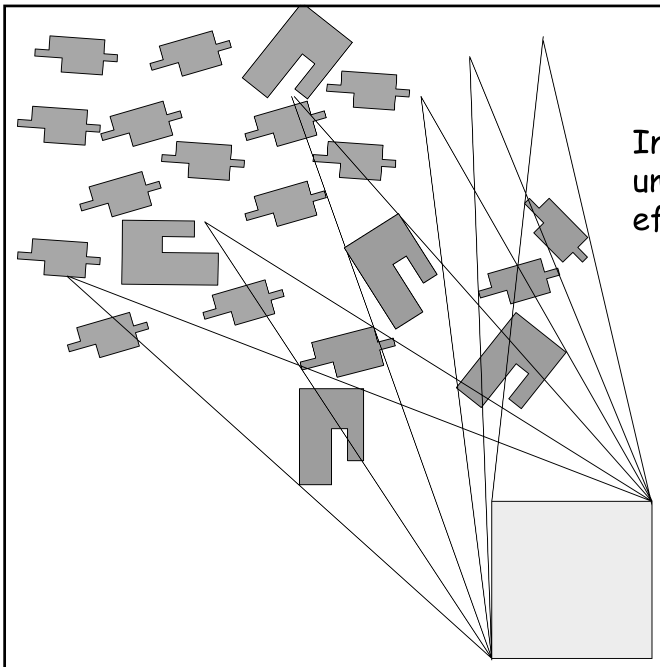
A Negative Umbra of a Single Occluder

SIGGRAPH 2001



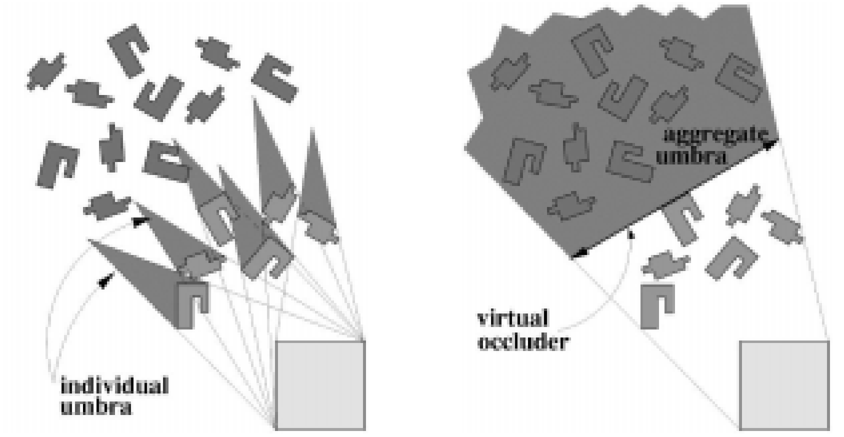
A Positive Umbra of a Single Occluder

SIGGRAPH 2001

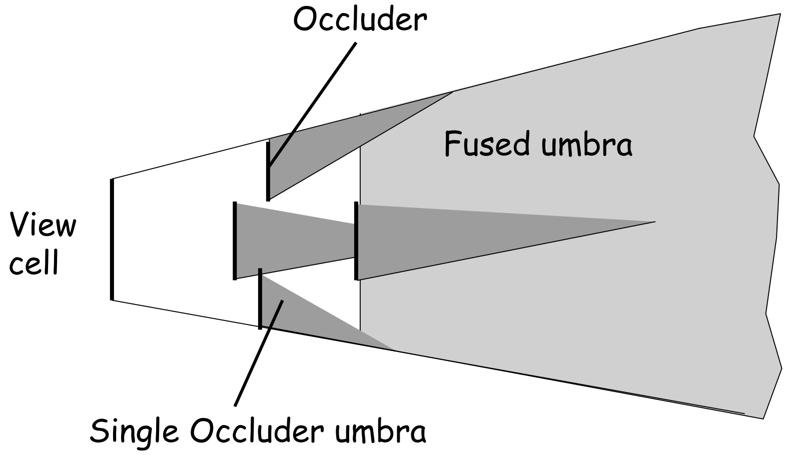


Individual negative umbrae are not effective...

Often the union of the umbrae of the individual objects is insignificant, while their aggregate umbra is large and can be represented by a single virtual occluder

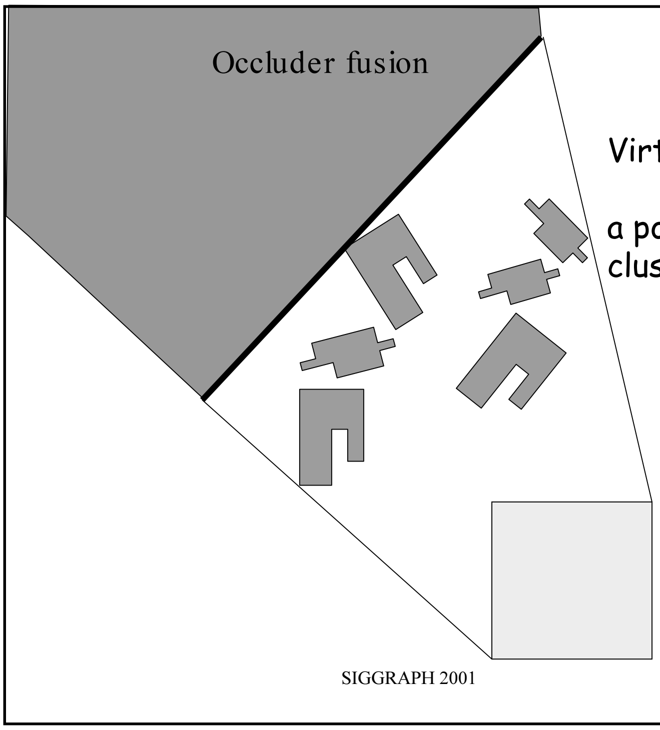


Occluder fusion



SIGGRAPH 2001

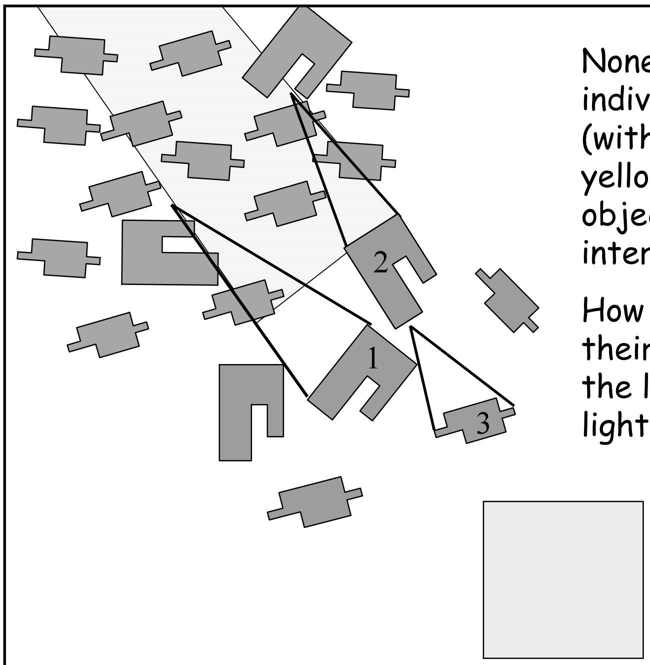
Occluder fusion



Virtual Occluders

a positive umbra of a cluster of occluders.

SIGGRAPH 2001



None of the individual umbrae (with respect to the yellow viewcell) of object 1,2 and 3 intersect.

How to aggregate their occlusion into the large umbra (in light blue) ?

Occluder Fusion

Fredo Durand et. al. *SIGGRAPH'2000*

Gernot Schaufler et. al. *SIGGRAPH'2000*

Peter Wonka et. al. *EGRW'2000*

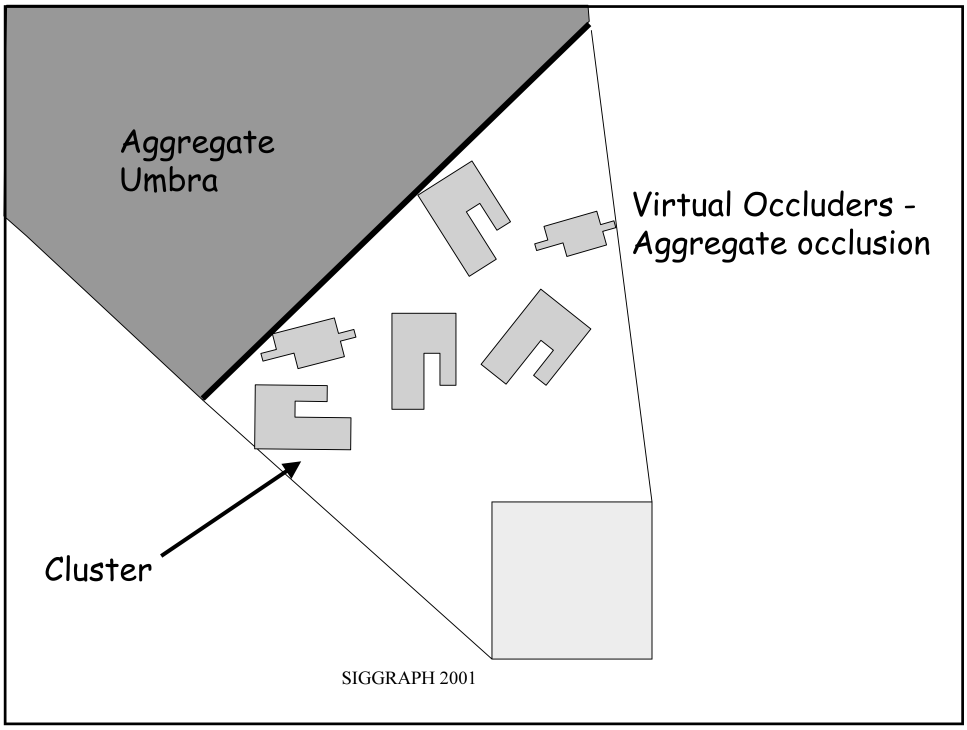
Vladlen Koltun et. al. *EGRW'2000*

Virtual Occluders:
A from-region visibility technique

Vladlen Koltun
Daniel Cohen-Or
Yiorgos Chrysanthou (UCL)

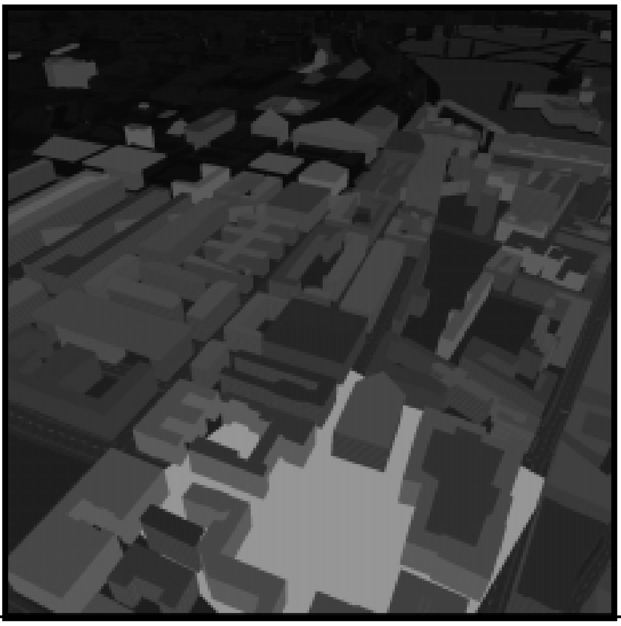
EGRW'2000

SIGGRAPH 2001

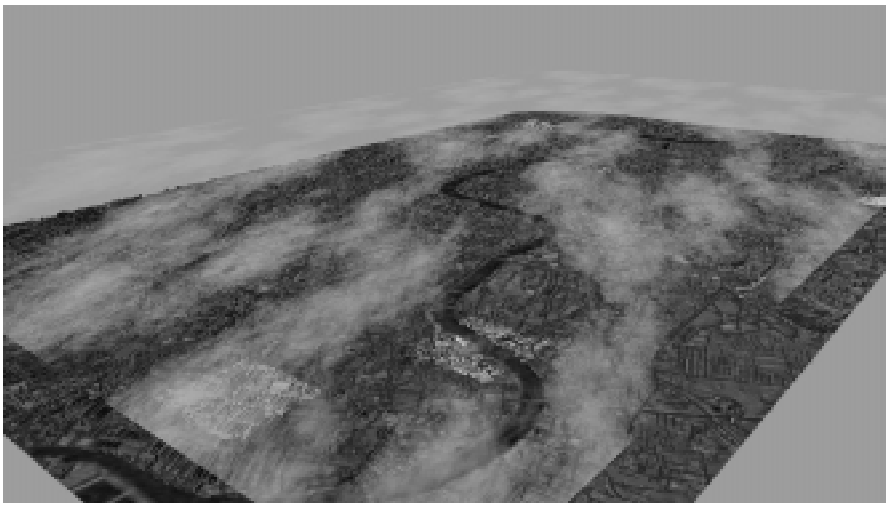


SIGGRAPH 2001

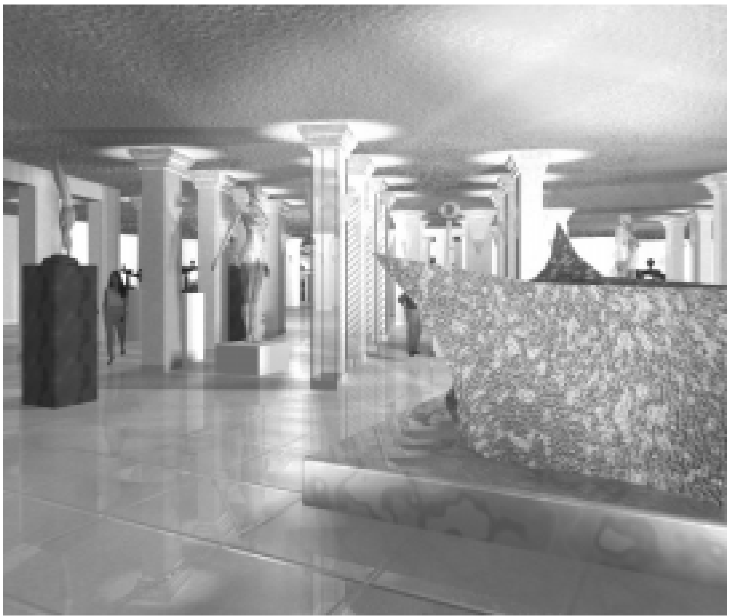
The effectiveness of Virtual Occluders



The London Model from UCL

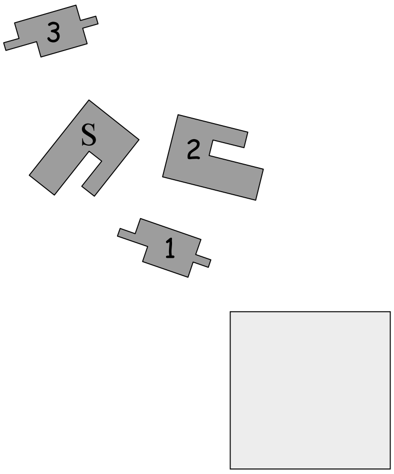


SIGGRAPH 2001



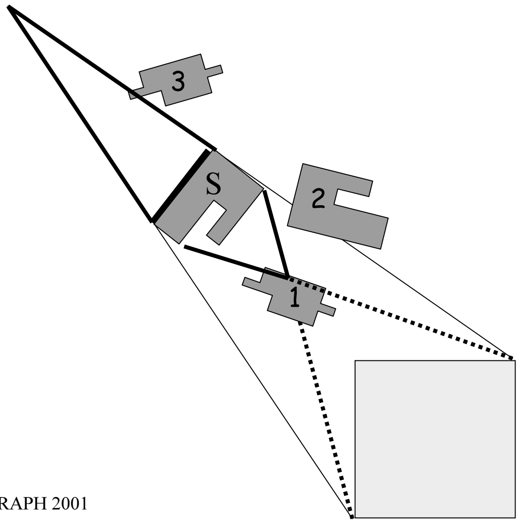
SIGGRAPH 2001

How to compute the virtual occluders

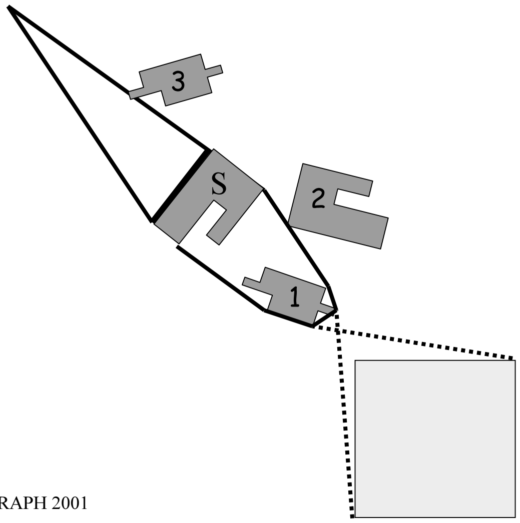


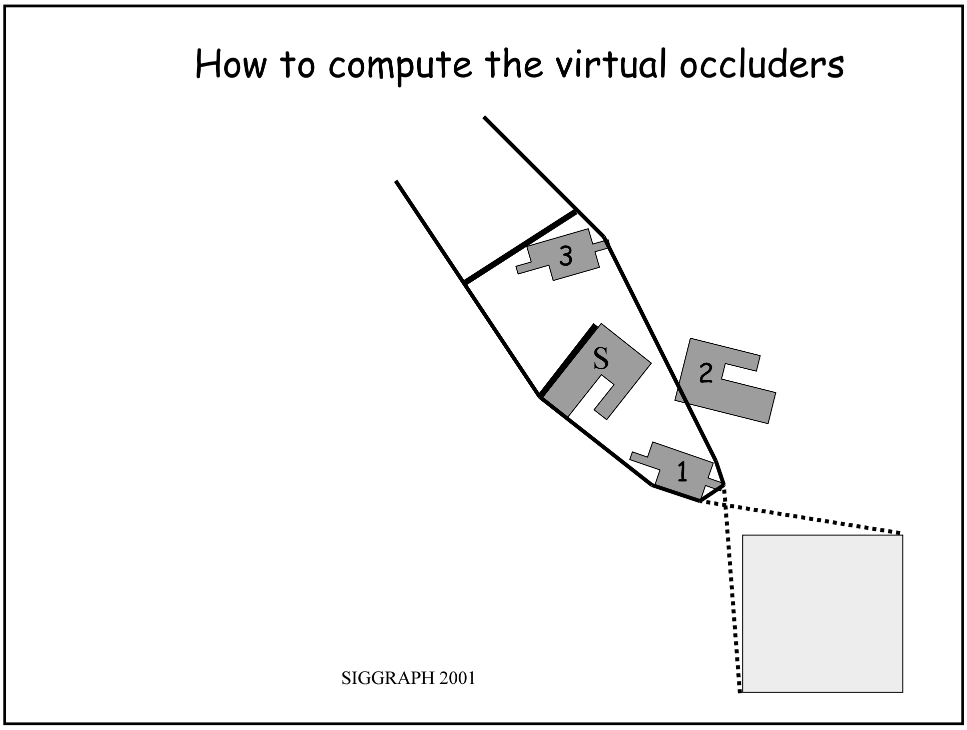
SIGGRAPH 2001

How to compute the virtual occluders



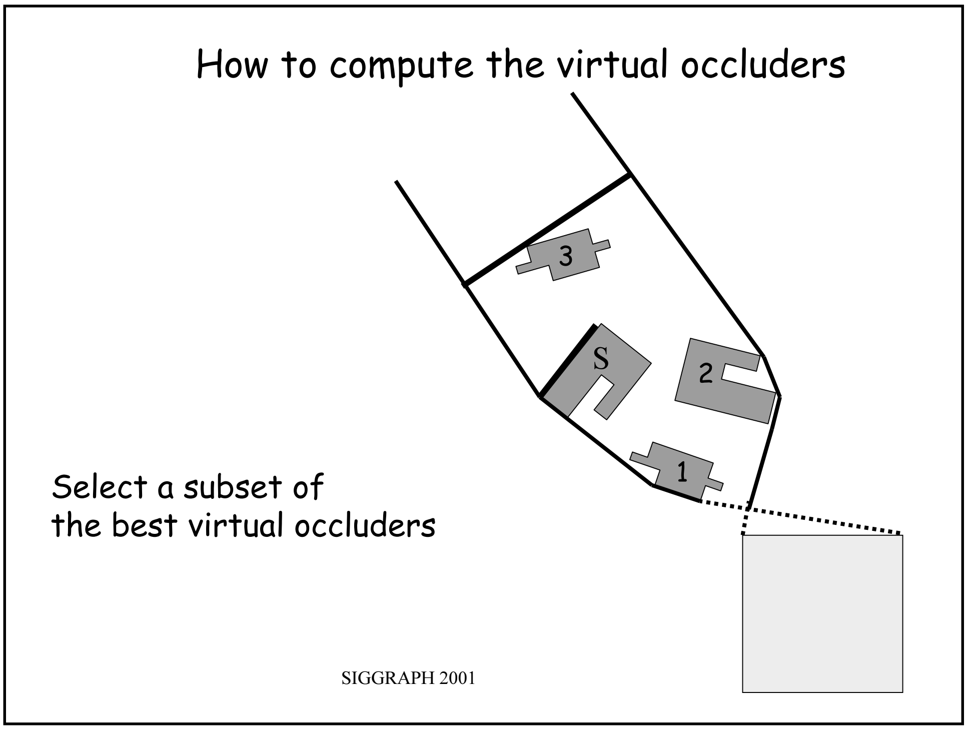
How to compute the virtual occluders





How to compute the virtual occluders

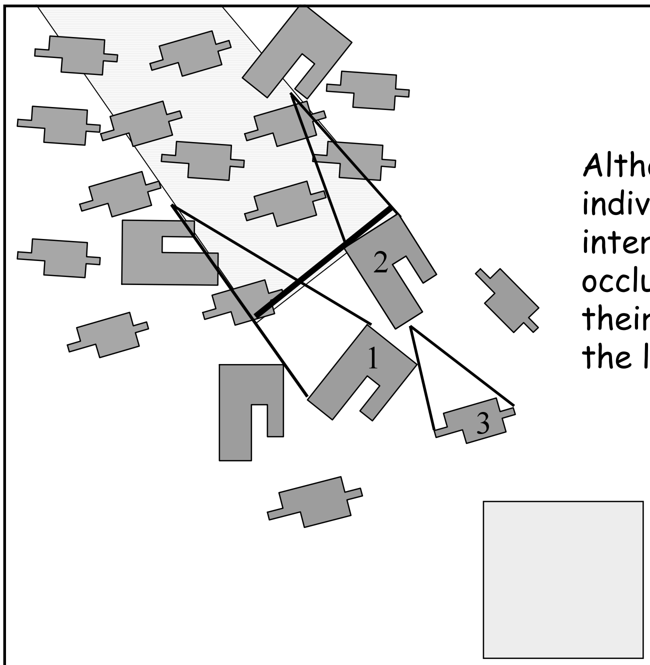
SIGGRAPH 2001



How to compute the virtual occluders

Select a subset of
the best virtual occluders

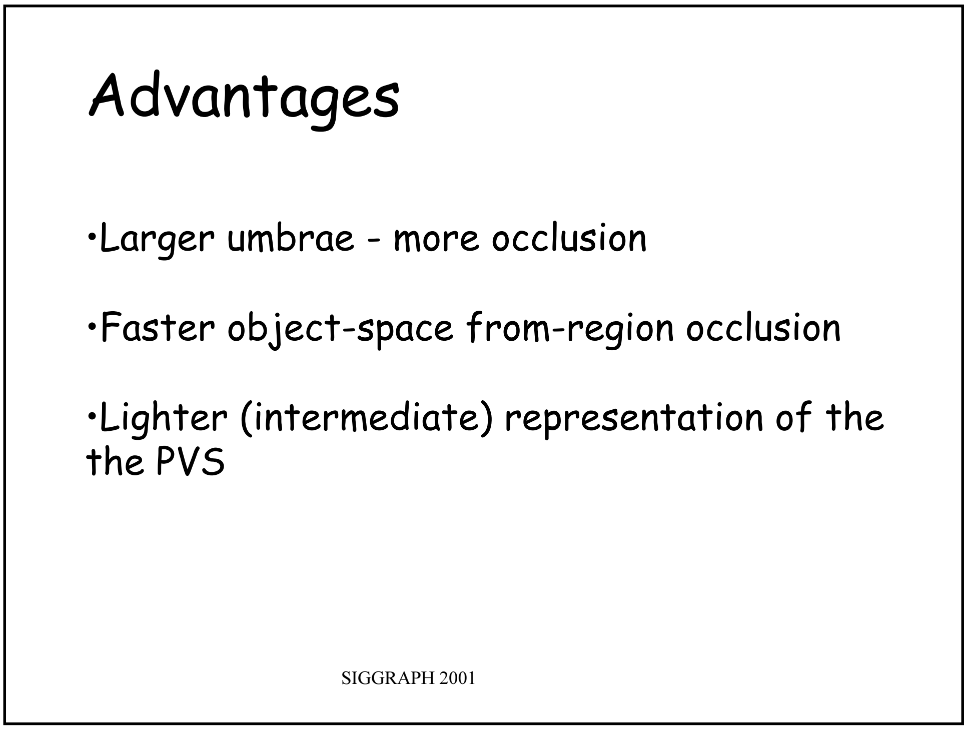
SIGGRAPH 2001



Although none of the individual umbrae intersect, the virtual occluder aggregates their occlusion into the large umbra.

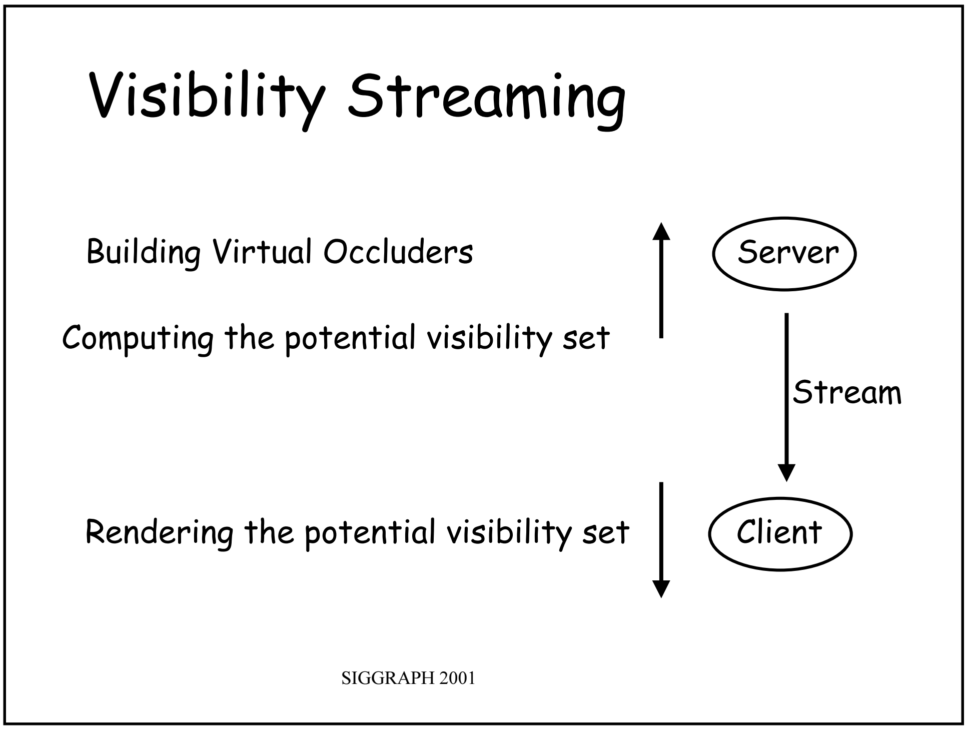
The Space Problem

- Precomputing the visibility sets requires a huge storage space
- This can be alleviated by using larger cells (negative umbrae)
- The PVS can be represented by an intermediate "light" set of Virtual Occluders.



Advantages

- •Larger umbrae - more occlusion
- •Faster object-space from-region occlusion
- •Lighter (intermediate) representation of the the PVS

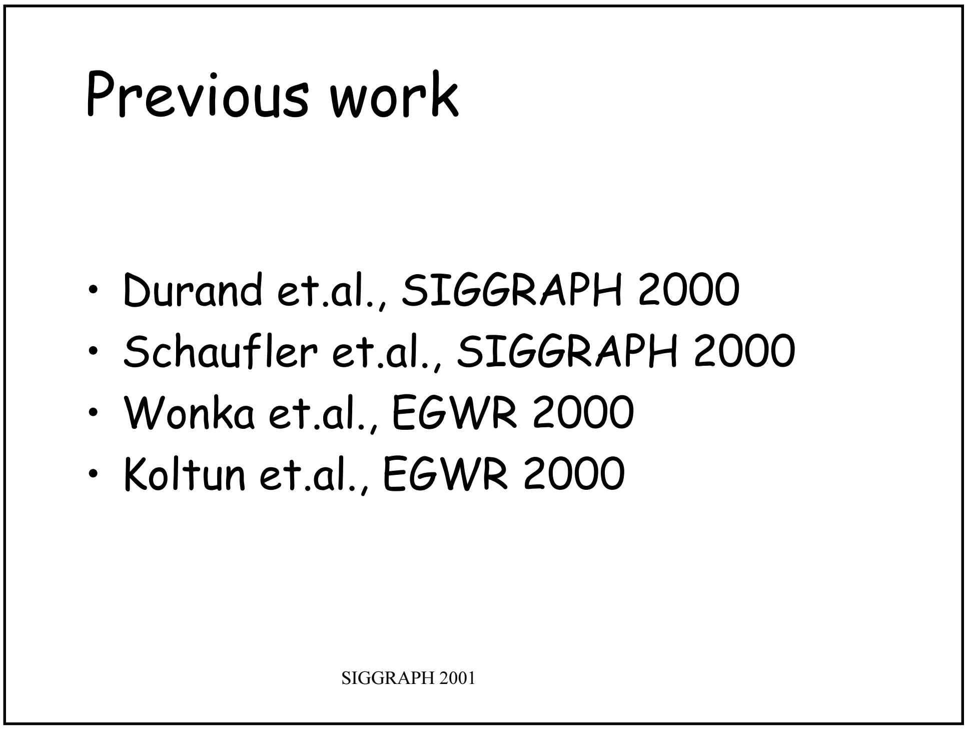


Visibility Streaming

Building Virtual Occluders

Computing the potential visibility set

Rendering the potential visibility set



Previous work

- Durand et.al., SIGGRAPH 2000
- Schaufler et.al., SIGGRAPH 2000
- Wonka et.al., EGWR 2000
- Koltun et.al., EGWR 2000

A Survey of Visibility for Walkthrough Applications

Daniel Cohen-Or[1,*] Yiorgos Chrysanthou[2,†] Cláudio T. Silva[3,‡]

[1]Tel Aviv University [2] University College London [3]AT&T Labs-Research

Abstract

The last few years have witnessed tremendous growth in the complexity of computer graphics models as well as network-based computing. Although significant progress has been made in the handling of specific types of large polygonal datasets (i.e., architectural models) on single graphics workstations, only recently have researchers started to turn their attention to more general solutions, which now include network-based graphics and virtual environments. The situation is likely to worsen in the future since, due to technologies such as 3D scanning, graphical models are becoming increasingly complex. One of the most effective ways of managing the complexity of virtual environments is through the application of smart visibility methods.

Visibility determination, the process of deciding what surfaces can be seen from a certain point, is one of the fundamental problems in computer graphics. It is required not only for the correct display of images but also for such diverse applications as shadow determination, global illumination, culling and interactive walkthrough. The importance of visibility has long been recognized, and much research has been done in this area in the last three decades. The proliferation of solutions, however, has made it difficult for the non-expert to deal with this effectively. Meanwhile, in network-based graphics and virtual environments, visibility has become a critical issue, presenting new problems that need to be addressed.

In this survey we review the fundamental issues in visibility and conduct an overview of the work performed in recent years.

1 Introduction

The term *visibility* is very broad and has many meanings and applications in various fields of computer science. Here, we focus on visibility algorithms in support of virtual reality applications. For a more general survey see [23] (also appears in [14]). For those interested in the computational geometry literature, see [21, 20, 22]. Zhang's thesis [80] contains a short survey of computer graphics visibility work. Moller and Haines [50, Chapter 7] cover several aspects of visibility culling.

We deal primarily with algorithms related to walkthrough applications where we assume that a scene consists of a very large number of primitives. Moreover, we assume that models keep getting larger and more complex and that user appetite will never be satisfied with the computational power available. For very complex models we can usually do better with a smart rendering algorithm than with faster machines.

One of the most interesting visibility problems in this context is the one of selecting a set of polygons from the model that is visible from a given viewpoint. More formally (after [21]), let the scene, $\mathcal{S}$, be composed of modeling primitives (*e.g.*, triangles) $\mathcal{S} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$, and a viewing frustum defining an eye position, a view direction, and a field of view. The visibility problem encompasses finding the visible fragments within the scene, that is, connected to the eyepoint by a line segment that meets the closure of no other primitive. One of the obstacles to solving the visibility problem is its complexity. For a scene with $n = O(|\mathcal{S}|)$ primitives, the complexity of the set of visible fragments might be as high as $O(n^2)$ (i.e., quadratic in the number of primitives in the input).

What makes visibility an interesting problem is that for large scenes, the number of visible fragments is usually much smaller than the total size of the input. For example, in a typical urban scenes, one can see only a very small portion of the entire model, regardless of one's location. Such scenes are said to be *densely occluded*, in the sense that from any given viewpoint, only a small fraction of the scene is visible [15]. Other examples include indoor scenes, where the walls of a room occlude most of the scene, and in fact, from any viewpoint inside the room, one may only see the details of that room or those visible through the *portals*, see Figure 1. A different example is a copying machine, shown in Figure 2, where from the outside one can only see its external parts. Although intuitive, this information is not available as part of the model representation, and only a non-trivial algorithm can determine it automatically. Note that one of its doors might be open.

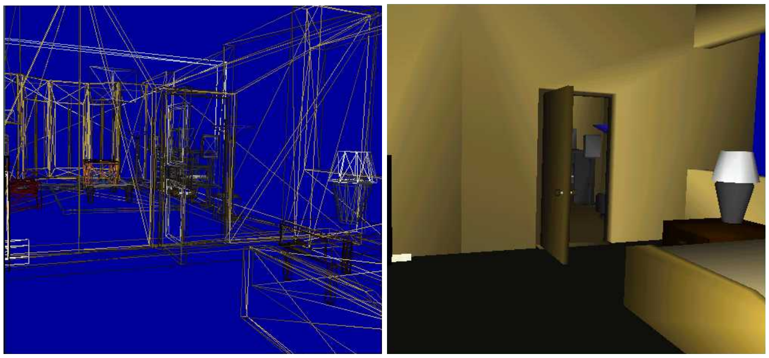
Visibility is not an easy problem, since a small change in the viewpoint might cause large changes in the visibility. It means that solving the problem at one point does not help much in solving it at a nearby point. An example of this can be seen in Figure 3. The *aspect graph*, described in Section 2, and the *visibility complex* (described in [23]) sheds light on the complex characteristics of visibility.

The rest of this paper is organized as follows. We first give a short description of the aspect graph, which is a fundamental concept in visibility, in Section 2. Then, we briefly review some 3D graphics hardware features which are important for visibility culling (Section 4). Next, we present a taxonomy of visibility culling algorithms in Section 5. This introductory part is then followed by a more detailed description and analysis of recent visibility-culling algorithms.

[*] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, daniel@math.tau.ac.il

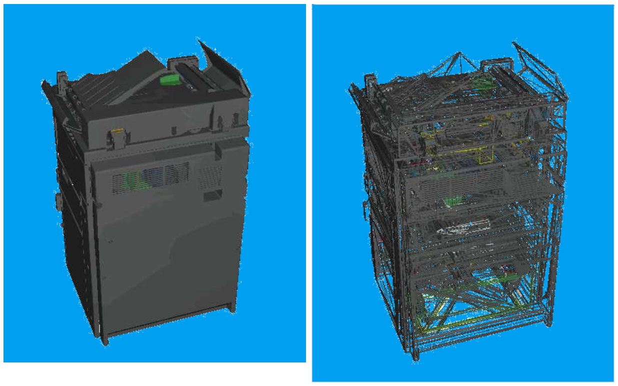
[†] Department of Computer Science, University College London, Gower Street, London WC1E 6BT, United Kingdom, y.chrysanthou@cs.ucl.ac.uk

[‡] AT&T Labs-Research, 180 Park Ave., PO Box 971, Florham Park, NJ 07932; csilva@research.att.com.



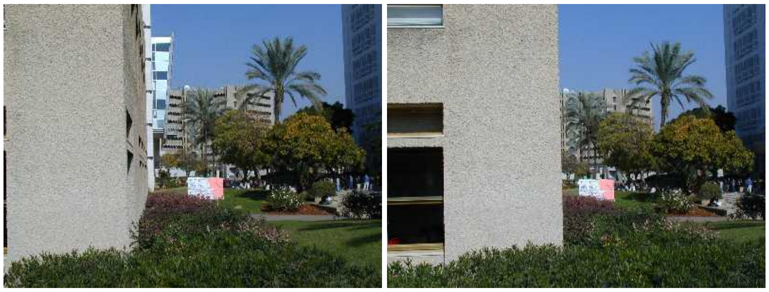
(a) (b)

Figure 1: With indoor scenes often only a very small part of the geometry is visible from any given viewpoint. Courtesy of Craig Gotsman, Technion.



(a) (b)

Figure 2: A copying machine; only a fraction of the geometry is visible from the outside. Courtesy of Craig Gotsman, Technion.



(a) (b)

Figure 3: A small change in the viewing position can cause large changes in the visibility.

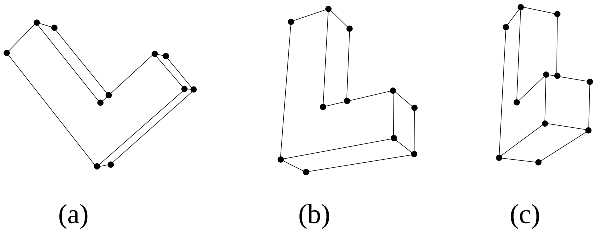


Figure 4: Two different view directions of an object have the same *aspect* if and only if the corresponding Image Structure Graphs are isomorphic. Note that (a) and (b) have the same aspect, which is different to (c).

2 The aspect graph

When dealing with visibility, it is useful to consider an important theoretical concept called the aspect graph [26]. Let us look at the two isomorphic graphs in Figure 4. They are a projection of a 3D object; however, we treat them as 2D entities. First, let us define the *Image Structure Graph* (ISG) as a planar graph, defined by the outlines of an image created by projecting a polyhedral object in a certain view direction. Then two different view directions of an object have the same *aspect* if and only if their corresponding ISGs are isomorphic. Now we can partition the viewspace into maximal connected regions in which the viewpoints have the same view or aspect. This partition is the VSP - *the visibility space partition*, where the boundary of a VSP region is called a *visual event* as it marks a change in visibility (see Figure 5).

The term, aspect graph, refers to the graph created by assigning a vertex to each region of the VSP, where the edges connect adjacent regions.

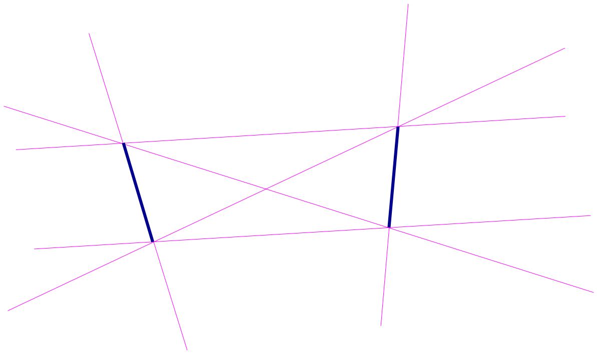


Figure 5: 2 polygons - 12 aspect regions.

Figures 5 and 6 show a visibility space partition in 2D, which is created by just two and three segments (the 2D counterparts of polygons), respectively. One can observe that the number of aspect regions is already large, and in fact, can be shown to grow quite rapidly.

Plantinga and Dyer [57] discuss aspect graphs and their worst-case complexity, including algorithms for efficiently computing aspect graphs. The worst complexity of aspect graphs is quite high, and in three dimensions, can be as large as $O(n^9)$. For a typical number of segments (say tens of thousands), in terms of space and time it turns out that computing the aspect graph is computationally impractical. (Plantinga [58] proposes an early conservative visibility algorithm based on his aspect graph work.)

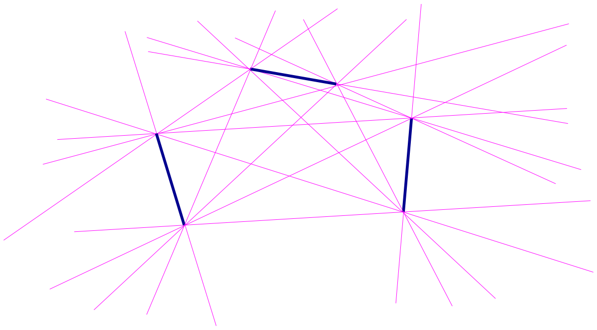


Figure 6: 3 polygons - “many” aspect regions.

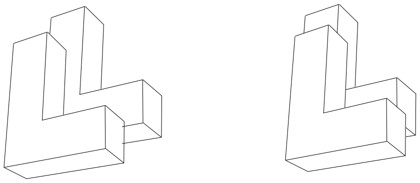


Figure 7: Different aspect regions can have equal sets of visible polygons.

However, as can be seen in Figure 7, different aspect regions can have equal sets of visible polygons. This means that there are far fewer different regions of different visibility sets than different aspects.

Looking once again at the aspect partition of the two segments in Figure 8, we can treat one as an occluder and the other as the occludee, defining their endpoint connecting lines as *supporting lines* and *separating lines*. These lines partition the space into three regions: (i) the region from which no portion of the occludee is visible, (ii) the region from which only a portion of the occludee is visible, and (iii) the region from which the occluder does not occlude any part of the occludee [17].

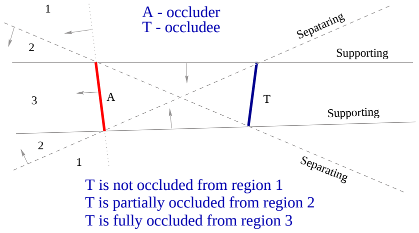

1
A - occluder
T - occludee
Sepatating
2
Supporting
3
A
T
Supporting
2
1
Separating
T is not occluded from region 1
T is partially occluded from region 2
T is fully occluded from region 3

Figure 8: Supporting and separating planes.

The 3D visibility complex [23] is another way of describing and studying the visibility of 3D space by a dual space of 3D lines, in which all the visibility events are described. This structure is global, spatially coherent and complete, since it encodes all the visibility relations in 3D. It allows efficient visibility computations, such as view extraction, computation of the aspect graph, discontinuity meshing and form-factor computation.

3 Hidden-surface removal methods

As mentioned in the introduction, one of the fundamental visibility problems in computer graphics is the determination of the visible parts of the scene, the so-called *hidden-surface removal* (HSR) (also known as visible-surface determination) algorithms. Assuming the scene is composed of and represented by triangles, these algorithms not only define the set of visible triangles, but also the exact portion of each visible triangle that has to be drawn into the image.

An early classification was proposed by Sutherland et al. [69]. Later it was reviewed in [28] and also in the computational geometry literature [22]. The HSR methods can be broadly classified into three groups: object precision methods, image precision methods and hybrid methods. Object precision methods compare objects to decide exactly which parts of each one is visible in the image. One of the first examples of this class was presented by Weiler and Atherton [74]. They used a general clipping method to partition polygons which were further away from the viewpoint using the boundaries of those closer, discarding the regions where they overlapped. Object precision algorithms can be considered as a continuous solution (to the extent that machine precision allows) but often suffer from scalability problems as the size of the environment grows, and are difficult to implement robustly.

Image precision algorithms on the other hand operate on the discrete representation of the image. The overall idea is to produce a solution at the resolution of the required image by determining the visible object at each pixel. Ray casting is one example of this class [3]. Other examples are the scan-line methods [9, 73], variations of which are popular in games and flight simulators, and the z-buffer [10] whose implementation in hardware has made it the de-facto standard HSR method today.

Finally, in the third class, are hybrid methods that combine object and image precision operations. Of most interest are the so-called list-priority algorithms. Their underlying idea is to quickly determine a partial order list of all polygons such that for any given pair p, q, if p can occlude some part of q, then p comes earlier in the list. In other words, if q is after p in the list, q cannot occlude p. Then during rendering, the ordered polygons are drawn back-to-front, thus occluding polygons are correctly drawn into the image, covering only those parts that are occluded. Some of the early methods were those of Schumacker et al. [61] and Newell et al. [54] and later Fuchs et al.'s BSP trees [29]. One of the additional features of the list-priority techniques is that they are able to correctly handle the rendering of transparent objects. Although the methods were originally designed for depth ordering of individual polygons, some of their ideas have been used in occlusion methods (i.e., [34]).

4 3D graphics hardware

In this section, we briefly review some common features of modern 3D graphics hardware which are helpful in visibility calculations.

We do not cover the important topic of efficient use of specific graphics hardware, in particular, the optimization of specific applications to specific hardware. A good starting point is the text by Moeller and Haines [50]. The interested reader should also consult the OpenGL tutorials given every year at Siggraph.

Hardware features for specific visibility calculations are usually bare-bones, because of the need for graphics hardware to be streamlined and very simple. Most often, by careful analysis of the hardware, it is possible to combine a software solution which exploits the basic hardware functionality, but at the same time also improves it considerably.

4.1 Graphics pipeline

The graphics pipeline is the term used for the path a particular primitive takes in the graphics hardware from the time the user defines it in 3D to the time it actually contributes to the color of a particular pixel on the screen. At a very high level, given a primitive, it must undergo several simple tasks before it is drawn on the screen.

Often, such as in the OpenGL graphics pipeline, a triangle primitive is first transformed from its local coordinate frame to a world coordinate frame; then it is transformed again to a normalized coordinate frame, where it is clipped to fit the view volume. At this point, a division by w is performed to obtain non-homogeneous normalized coordinates, which are then normalized again to be in screen-space. Depending on a set of user-defined state flags, the hardware can reject the primitive based (among other things) on the direction of its normal. This is called back-face culling, and is a very primitive form of visibility culling.

Once a primitive has passed all these phases, the rasterization phase can start. It is here that the colors (and other properties) of each pixel are computed. During rasterization, we usually refer to the primitives as "fragments". Modern graphics architectures have several per-fragment operations that can be performed on each fragment as they are generated.

As fragments are computed, they pass through further processing, and the hardware will incrementally fill several buffers in order to compute the image. The actual image we see on the screen is only one of these buffers: the color buffer. Other buffers include the stencil buffer and the depth (or z-) buffer. There are other buffers, such as the accumulation buffer, etc., but we do not use them in the rest of this paper. In OpenGL, updates to the different buffers can be toggled by a set of function calls, *e.g.* `glEnable(GL_DEPTH_TEST)`.

One view of the OpenGL buffers is as a simple processor with little memory (just a few bytes), and a limited instruction set. Recently, techniques for performing general computations using the OpenGL pipeline have been proposed. Two such examples are Peercy et al. [56] and Trendall and Stewart [71].

4.2 Stencil buffer

The stencil buffer is composed of a small set of bits (usually more than 4) that can be used to control which areas of the other buffers,)textite.g. color buffer), are currently active for drawing. A common use of the stencil buffer is to draw a piece of static geometry once (the cockpit of an airplane), and then mask the area so that no further changes can be made to those pixels.

But the stencil buffer is actually much more flexible, since it is possible to change the value of the pixels on the stencil buffer depending on the outcome of the test performed. For instance, a very useful computation that uses the stencil buffer is to compute the "depth-complexity" of a scene. For this, one can simply program the stencil buffer as follows:

```
glStencilFunc(GL_ALWAYS, ˜0, ˜0);
```

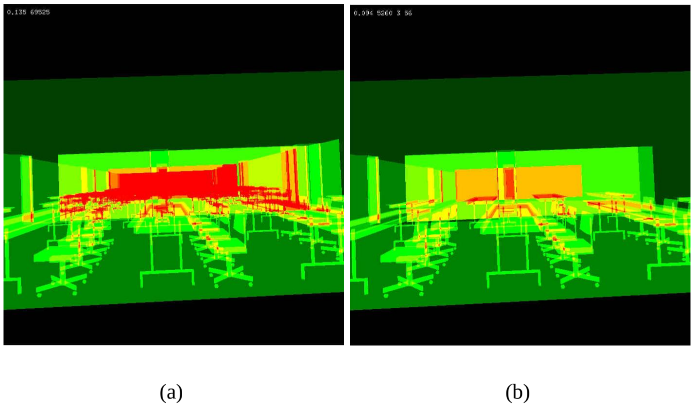


Figure 9: Depth complexity of the scene as rendered by (a) view-frustum culling, (b) a conservative occlusion culling technique. The depth complexity ranges from light green (low) to bright red (high). If the occlusion-culling algorithm were “exact”, (b) would be completely green.

```
  glStencilOp(GL_KEEP, GL_INCR, GL_INCR);
```

which essentially means the stencil buffer will get incremented every time a pixel is projected onto it. Figure 9 shows a visual representation of this. The stencil buffer is useful in several types of visibility computations, such as real-time CSG calculations [31], occluder calculations [25], and so on.

4.3 Z-buffer

The z-buffer is similar to the stencil buffer, but serves a more intuitive purpose. Basically, the z-buffer saves its “depth” at each pixel. The idea is that if a new primitive is obscured by a previously drawn primitive, the z-buffer can be used to reject the update. The z-buffer consists of a number of bits per pixel, usually 24 bits in most current architectures.

The z-buffer provides a brute-force approach to the problem of computing the visible surfaces. Just rendering each primitive, and the z-buffer will take care of not drawing in the color buffer of those primitives that are not visible. The z-buffer provides a great functionality, since (on fully hardware-accelerated architectures) it is able to solve the visibility problem (up to screen-space resolution) of a set of primitives in the time it would take to scan-convert them all.

As a visibility algorithm, the z-buffer has a few drawbacks. One drawback is that each pixel in the z-buffer is touched (potentially) as often as its depth complexity, although one simply needs the top surface of each pixel. Because of this potentially excessive overdrawing a lot of computation and memory bandwidth is wasted. A visibility pre-filtering technique, such as back-face culling, can be used to improve the speed of rendering with a z-buffer.

There have been several proposals for improving the z-buffer, such as the hierarchical z-buffer [35] (see Section 7.1 and related techniques). A simple, yet effective hardware technique for improving the performance of the visibility computations with a z-buffer has been proposed by Scott et al. [62], see Section 7.5.

5 Visibility culling algorithms

Visibility algorithms have recently regained attention in computer graphics as a tool for handling large and complex scenes, which consist of millions of polygons. In the early 1970s hidden surface removal (HSR) algorithms (see Section 3) were developed to solve the fundamental problem of determining the visible portions of the polygons in the image. In light of the Z-buffer being widely available, and exact visibility computations being potentially too costly, one idea is to use the Z-buffer as a filter, and design algorithms that lower the amount of overdraw by computing an approximation of the *visible set*. In more precise terms, define the visible set $\mathcal{V} \subset \mathcal{S}$ to be the subset of primitives which contribute to at least one pixel of the screen.

In computer graphics, visibility-culling research mainly focuses on algorithms for computing (hopefully tight) estimations of $\mathcal{V}$, then using the Z-buffer to obtain correct images.

5.1 View frustum and back-face culling

The simplest examples of visibility culling algorithms are back-face and view-frustum culling [28]. Back-face culling algorithms avoid rendering geometry that faces away from the viewer, while viewing-frustum culling algorithms avoid rendering geometry that is outside the viewing frustum. These culling operations can be left to the graphics hardware without affecting the final image. However, that comes at a great cost since the polygons will be processed through most of the pipeline only to be rejected just before scan converting.

Back-facing polygons can be identified with a simple dot product, since their normal points away from the view-point. On average we expect half the scene polygons to be back-facing, so ideally we would like to avoid processing all of them. Kumar et al. [45] present a method which has a sublinear number of polygons. The input model is partitioned into a hierarchy of clusters based on both similarity of orientation and physical proximity of the polygons. The viewspace is also partitioned with respect to the clusters. At each frame the viewpoint position is hierarchically compared with the clusters in order to quickly reject the bulk of the back-facing polygons. Frame-to-frame coherence is further used to accelerate

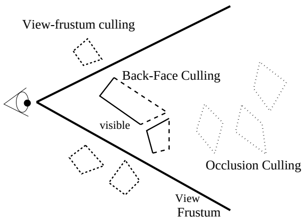


Figure 10: Three types of visibility culling techniques: (i) view frustum culling, (ii) back-face culling and (iii) occlusion culling.

the process.

View frustum culling is usually performed using either a hierarchy of bounding volumes or a spatial data structure, such as a KD-tree, octree or BSP tree. This is hierarchically compared with the view frustum to quickly reject parts of the scene that are clearly outside [13].

Slater et al. [65] present an alternative approach which makes heavy use of frame-to-frame coherence. It relies on the fact that the sets of objects that are completely outside, completely inside, or intersect the boundary of the view volume, change slowly over time. This coherence is exploited to develop an algorithm that quickly identifies these three sets of objects, and partitions those completely outside into subsets which are probabilistically sampled according to their distance from the view volume. A statistical object representation scheme is used to classify objects into the various sets. The algorithm is implemented in the context of a BSP tree.

Very recently, Assarsson and Möller [4] proposed a new view-frustum culling technique. Their work is based on shrinking the view frustum to enable the use of point-containment queries to efficiently accept or reject primitives.

5.2 Occlusion culling

Even though both of the above techniques are very effective in culling geometry, more complex techniques can lead to substantial improvements in rendering time. The term *Occlusion culling* is used for visibility techniques that avoid rendering primitives that are occluded by some other part of the scene. This technique is global as it involves interrelationship among polygons and is thus far more complex than local visibility techniques. The three kinds of visibility culling can be seen in Figure 10.

It is important to note the differences between occlusion culling and HSR. HSR algorithms determine which portions of the scene need to be drawn on the screen. These algorithms eventually remove the occluded parts, but in doing so, are expensive, since they usually have to touch all the primitives in $\mathcal{S}$ (and actually have a running time that is superlinear in the size of $\mathcal{S}$). Occlusion-culling techniques are supposed to be *output sensitive*, that is, their running time should be proportional to the size of $\mathcal{V}$, which for most complex scenes, is a small subset.

Let us define the following notation for a scene consisting of polygons.

- The *exact visibility set*, $\mathcal{V}$, is the set of all polygons which are at least partially visible, and only these polygons.

- The *approximate visibility set*, $\mathcal{A}$, is a set that includes most of the visible polygons plus maybe some hidden ones.

- The *conservative visibility set*, $\mathcal{C}$, is the set that includes at least all the visible objects plus maybe some additional invisible objects. It may classify an invisible object as visible, but may never classify a visible object as invisible.

5.3 Conservative visibility

A very important concept is the idea of *conservative visibility*. The idea is to design efficient output-sensitive algorithms for computing $\mathcal{C}$, then to use a standard HSR as a back-end for computing the correct image.

These methods yield a *potential visibility set* (PVS) which includes all the visible polygons, plus a small number of occluded polygons. Then the HSR processes the (hopefully small) excess of polygons included in the PVS. Conservative occlusion culling techniques have the potential to be significantly more efficient than the HSR algorithms. Conservative culling algorithms can also be integrated into the HSR algorithm, aiming towards an output sensitive algorithm [35].

To reduce the computational cost, the conservative occlusion culling algorithms usually use a hierarchical data structure where the scene is traversed top-down and tested for occlusion against a small number of selected occluders [18, 39]. In these algorithms the selection of the candidate occluders is done before the online visibility calculations. The efficiency of these methods is directly dependent on the number of occluders and their effectiveness. Since the occlusion is tested from a point, these algorithms are applied in each frame during the interactive walkthrough.

5.4 A taxonomy of occlusion culling techniques

In order to roughly classify the different visibility-culling algorithms, we will employ a loosely-defined taxonomy:

- *Conservative vs. Approximate.*

  Few visibility-culling algorithms attempt to find the exact visible set, since they are mostly used as a front-end for another hidden-surface removal algorithm, most often the Z-buffer. Most techniques described in this paper are conservative, that is, they overestimate the visible set. Only a few approximate the visible set, but are not guaranteed of finding all the visible triangles, e.g., PLP [43, 42] (there is also a conservative version of PLP which is described in [40]). Others can be tuned to be conservative or approximate depending on the time constraint and available resources. In an attempt to accelerate the culling step they might actually miss small visible primitives, such as HOM [81, 80], and also the OpenGL assisted occlusion culling of Bartz et al. [6, 5].

- *Point vs. Region.*

  The major difference here is whether the particular algorithm performs computations that depend on the exact location of the viewpoint, or performs bulk computations which can be re-used anywhere in a region of space.

Obviously, from-region algorithms perform their visibility computations on a region of space, that is, while the viewer is inside that region, these algorithms tend to render the same geometry. The strength of the from-region visibility set is that it is valid for a number of frames, and thus its cost is amortized over a number of frames (see Section 8).

Most other algorithms attempt to perform visible-set computations that depend on the exact location of the viewpoint.

- *Precomputed vs. Online.*

  Most techniques need some form of preprocessing, but what we mean by "precomputed" are the algorithms that actually store visibility computations as part of their preprocessing.

  Almost all of the from-region algorithms should be classified as "precomputed". A notable exception is [44].

  In general, the other algorithms described do their visibility computation "online", although much of the preprocessing might have been performed before. For instance, HOM [81, 80], DDO [7], Hudson et al. [39], Coorg and Teller [18], perform some form of occluder selection which might take a considerable amount of time (in the order of hours of preprocessing), but in general have to save very little information to be used during rendering.

- *Image space vs. Object space.*

  Almost all of the algorithms use some form of hierarchical data structure. We classify algorithms as operating in "image-space" versus "object-space" depending on where the actual visibility determination is performed.

  For instance, HOM [81, 80] and HZB [35, 36] perform the actual occlusion determination in image-space (e.g., in HOM, the occlusion maps are compared with a 2D image projection and not the 3D original representation.). Other techniques that explore image-space are DDO [7] (which also explores a form of object-space occlusion-culling by performing a view-dependent occluder generation) and [6, 5].

  Most other techniques work primarily in object-space.

- *Software vs. Hardware.*

  Several of the techniques described can take further (besides the final z-buffer pass) advantage of hardware assistance either for its precomputation or during the actual rendering.

  For instance, the from-region technique of Durand et al. [25] makes non-trivial use of the stencil buffer; HOM [81, 80] uses the texture hardware to generate mipmaps; [6, 5] uses the OpenGL selection mode; and Meissner et al. [49] uses the HP occlusion-culling test.

  The HP occlution-culling test [62] is not actually an algorithm on its own, but a building block for further algorithms. It is also exploited (and expanded) in [40].

- *Dynamic vs. Static scenes.*

  A few of the algorithms in the literature are able to handle dynamic scenes, such as [68] and HOM [81].

  One of the main difficulties is handling changes to object hierarchies that most visibility algorithms use. The more preprocessing used, the harder it is to extend the algorithm to handle dynamic scenes.

- *Individual vs. Fused occluders.*

  Given three primitives, A, B, and C, it might happen that neither A nor B occlude C, but together they do occlude C. Some occlusion-culling algorithms are able to perform *occluder-fusion*, while others are only able to exploit single primitive occlusion. citeCohen-Or:1998:CVA,Coorg:1997:ROC,ct-tccv-96 give examples of techniques that use a single (fixed number of) occluder(s). Papers [79, 35, 42] support occluder fusion.

5.5 Related problems

There are many other interesting visibility problems, for instance:

- **Shadow algorithms.** The parts that are not visible from the light source are in the shadow. So occlusion culling and shadow algorithms have a lot in common and in many ways are conceptually similar [78, 12]. It is interesting to note that conservative occlusion culling techniques have not been as widely used in shadow algorithms.

- **The Art Gallery Problem.** One classic visibility problem is that of positioning a minimal number of guards in a gallery so that they cover all the walls of the gallery. This class of problem has been extensively studied in computational geometry, see, for instance, O'Rourke [55].

  In this context, "cover" can have a different meaning. Much is known about this problem in 2D, but in 3D, it gets much harder. Fleishman et al. [27] proposes an algorithm for automatically finding a set of posing cameras which cover a 3D environment. Stuerzlinger [66] proposes a technique for a similar problem.

- **Radiosity solutions.** This is a much more difficult problem to compute accurately. In radiosity, energy needs to be transfered from each surface to every other *visible* surface in the environment [32, 37]. This requires a from-region visibility determination to be applied at each surface or patch. Exact solutions are not practical, and techniques such as clustering [64] are often used.

6 Object-space culling algorithms

Work on object-space occlusion culling dates back at least to the work of Teller and Sèquin [70] and Airey et al. [1] on indoor visibility.

The work of Teller and Sèquin is mostly based on 2D, since it deals with computing potentially visible sets for cells in an architectural environment. Their algorithm first subdivides space into cells using a 2D BSP tree. Then it uses the connectivity between the cells, and computes whether straight lines can hit a set of "portals" (mostly doors) in the model. They elegantly model the stabbing problem as a linear programming problem, and in each cell save the collection of potentially visible cells. Figure 11 shows one of the results presented in their

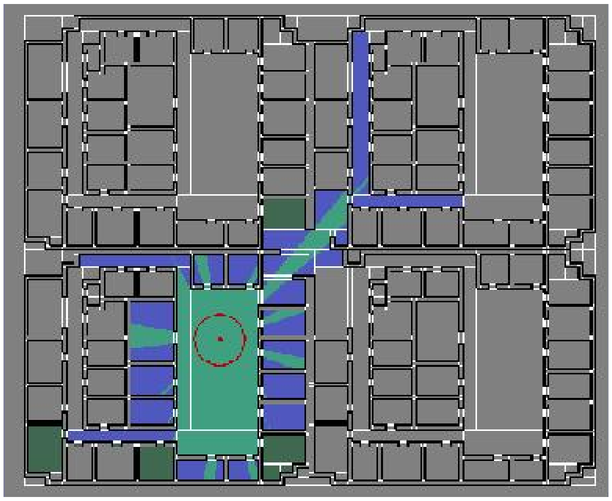


Figure 11: Results from [70] showing the potentially visible set from a given cell. Courtesy of Seth Teller, UC, Berkeley.

paper. The linear programming solution computes cell-to-cell visibility, which does not constrain the position of a viewer inside the cell, nor the direction in which he is looking, and thus is far too conservative. They also propose techniques which further constrain the PVS by computing eye-to-cell visibility, which take into consideration the view-cone emanating from the viewer.

Another technique that exploits cells and portals in models is described in Luebke and Georges [48]. Instead of precomputing the visibility, Luebke and Georges perform an on-the-fly recursive depth-first traversal of the cells using screen-space projections of the portals to overestimate the portal sequences. In their technique they use a “cull box” for each portal, which is the axial 2D bounding box of the projected vertices of the portal. Any geometry which is not inside a cull box of the portal cannot be visible. The basic idea is then to clip the portal1s cull boxes as the cells are traversed, and only to continue the traversal into cells which have a non-zero (intersection) portal-sequence. Their technique is simple and quite effective; the source code (an SGI Performer library) is available for download from David Luebke’s web page. [1]

6.1 Coorg and Teller

Coorg and Teller [17, 18] have proposed object-space techniques for occlusion culling. The technique in [18] is most suitable for use in the presence of large occluders in the scene. Their algorithm explores the visibility relationships between two convex objects as in Figure 12. In brief, while an observer is between the two supporting planes to the left of A, it is never possible to see B. The Coorg and Teller technique uses simple concepts such as this to develop a technique based on tracking visibility events among objects as the user moves and the relationships among objects change. The algorithm proposed in [17] is conservative, and explores temporal coherency as it tracks the visibility events.

In [17], Coorg and Teller give sufficiency conditions for computing the visibility of two objects (that is, whether one occludes the other), based on tracking relationships among the silhouette edges supporting and separating the planes of the different objects. They build an algorithm which incrementally tracks changes in those relationships. There, they also show how to use object hierarchies (based on octrees) to handle the potential quadratic complexity computational increase. One drawback of this technique (as pointed out by the authors in their subsequent work [18]) is precisely the fact that it needs to reconstruct the visibility information for a continuous sequence of viewpoints.

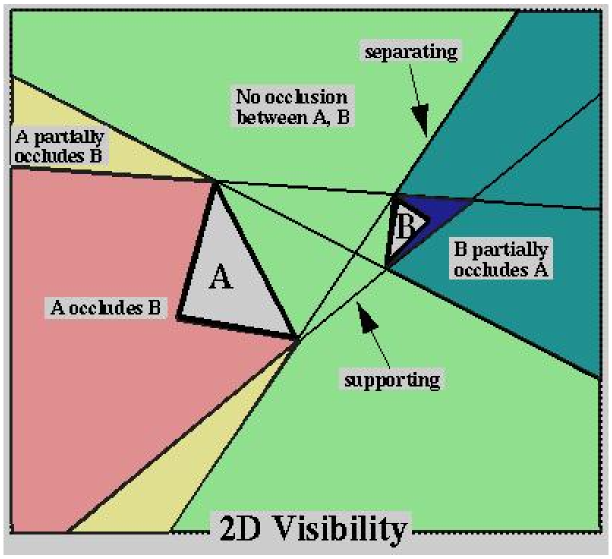


Figure 12: The figure highlights the visibility properties exploited by the algorithm of Coorg and Teller [17, 18]. While an observer is between the two supporting planes to the left of A, it is never possible to see B. Courtesy of Satyan Coorg, MIT.

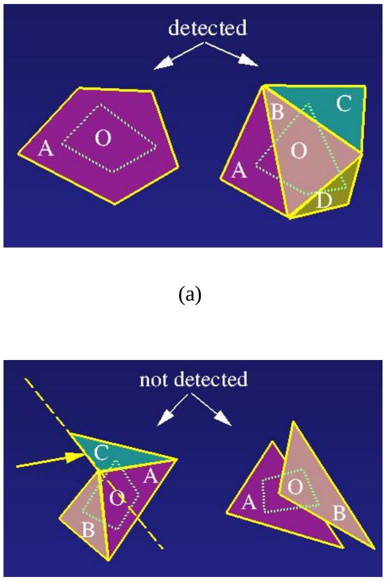
In [18], Coorg and Teller propose an improved algorithm. (It is still based on the visibility relationship shown in Figure 12.) Instead of keeping a large number of continuous visibility events, in [18], they dynamically choose a set of occluders, which is used to determine which portions of the rest of the scene cannot be seen. The scene is inserted into an object hierarchy, and the occluders are used to determine which portions of the hierarchy can be pruned, and not rendered.

Coorg and Teller [18] develop several useful building blocks for implementing this idea, including a simple scheme to determine when the fusion of multiple occluders can be added together (see Figure 13), and a fast technique for determining supporting and separating planes. They propose a simple metric for identifying the dynamic occluders which is based on approximating the solid angle an object subtends:

$$\frac{-A(\vec{N}\cdot\vec{V})}{||\vec{D}||^2}$$

, where A is the area of the occluder, $\vec{N}$ the normal, $\vec{V}$ the viewing direction, and $\vec{D}$ the vector from the viewpoint to the center of the occluder.

[1] Pfportals can be obtained at http://pfPortals.cs.virginia.edu.



(a)

(b)

Figure 13: The figure illustrates that the algorithm described in [18] can perform occlusion fusion if the occluders combine to be a larger “convex” occluder. Courtesy of Satyan Coorg, MIT.

6.2 Culling using shadow frusta

The work described by Hudson et al. in [39] is in several ways similar to the work of Coorg and Teller [18]. Their scheme also works by dynamically choosing a set of occluders, then using those occluders as the basis for culling the rest of the scheme. The differences between the two works lie primarily in the details. In [39], the authors propose extra criteria for choosing the occluders. Besides the Coorg and Teller solid-angle heuristic, they also propose taking into account the depth complexity and coherence of the occluders. They use a spatial partition of the scene, and for each cell, identifying the occluders that will be used anytime the viewpoint is inside that cell, and store them for later use.

A separate data structure, a hierarchy of bounding volumes is used for the occlusion culling. The way Hudson et al. determine which parts of the hierarchy are occluded is different to that of Coorg and Teller. For each of the *n* best occluders that fall within the view frustum, the authors build a shadow frustum using the viewpoint as the apex and passing through the occluder silhouette. The scene hierarchy is tested top-down against each of these shadow frusta. If a node of the hierarchy is found to be totally enclosed by one of the frusta then it is occluded and hence discarded (for this frame). If it is found not to intersect any of them then it totally visible and all the objects below it are rendered. If however it partially overlaps even one of them then its children need to be further tested. Interference detection techniques are used for speeding up the tests.

6.3 BSP tree culling

The method described in Hudson et al. [39] can be improved using BSP trees. Bittner et al. [8] combine the shadow frusta of the occluders into an *occlusion tree*. This is done in a very similar way to the SVBSP tree of Chin and Feiner [11]. The tree starts as a single *lit* (visible) leaf and occluders are inserted, in turn, into it. If an occluder reaches a *lit* leaf then it augments the tree with its shadow frustum; if it reaches a *shadowed* (invisible) leaf then it is just ignored since it means it already lies in an occluded region. Once the tree is built the scene hierarchy can be compared with it. The cube representing the top of the scene hierarchy is inserted into the tree. If it is found to be fully visible or fully occluded then we stop and act appropriately, otherwise its children are compared with the occlusion tree recursively. This method has an advantage over [39] in that instead of comparing the scene with each of the *N* shadow frusta, it is compared with one tree of depth (potentially) O(*N*).

The above technique is conservative; an alternative *exact* method was proposed much earlier by Naylor [53]. That involved a merging of the occlusion tree with the BSP tree representing the scene geometry.

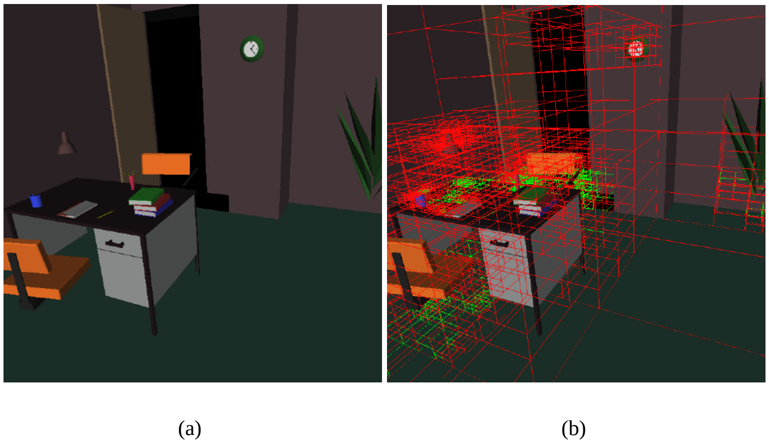
6.4 Prioritized-layered projection

Prioritized-Layered Projection (PLP) is a technique for fast rendering of high-depth complexity scenes. It works by *estimating* the visible polygons of a scene from a given viewpoint incrementally, one primitive at a time. On its own, PLP is not a conservative technique, but instead is suitable for the computation of partially correct images for use as part of time-critical rendering systems. At a very high level, PLP amounts to the modification of a simple view-frustum culling algorithm. However, it requires the computation of a special occupancy-based tessellation, and the assignment of a *solidity* value to each cell of the tessellation, which is used to compute a special ordering on how primitives get projected.

The core of the PLP algorithm consists of a space-traversal algorithm, which prioritizes the projection of the geometric primitives in such a way as to avoid (actually delay) projecting cells that have a small likelihood of being visible. Instead of explicitly overestimating, the algorithm works on a budget. At each frame, the user can provide the maximum number of primitives to be rendered, a polygon budget, and the algorithm will deliver what it considers to be the set of primitives which maximizes the image quality (using a solidity-based metric).

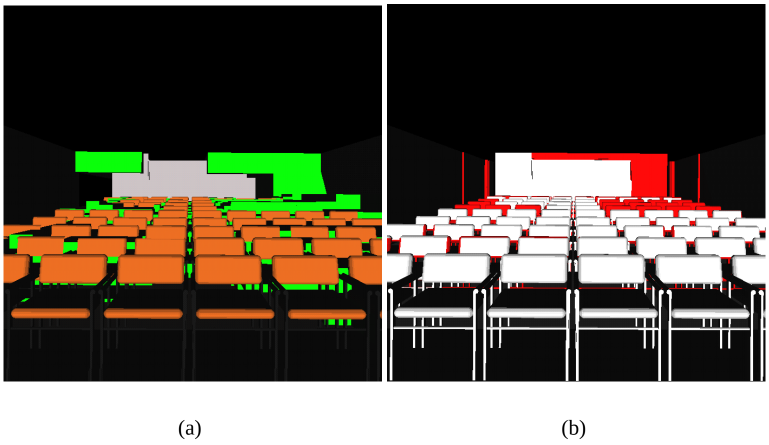
PLP is composed of two parts. First, PLP tessellates the space that contains the original input geometry with convex cells. During this one-time preprocessing, a collection of cells is generated in such a way as to roughly keep a uniform density of primitives per cell. The sampling leads to large cells in unpopulated areas, and small cells in areas that contain a lot of geometry. Using the number of modeling primitives assigned to a given cell (*e.g.*, tetrahedron), a *solidity* value ρ is defined. The accumulated solidity value used throughout the priority-driven traversal algorithm can be larger than one. The traversal algorithm prioritizes cells based on their solidity value. Preprocessing is fairly inexpensive, and can be done on large datasets (about one million triangles) in a couple of minutes.

The rendering algorithm traverses the cells in roughly front-to-back order. Starting from the seed cell, which in general contains the eye position, it keeps carving cells out of the tes-



(a) (b)

Figure 14: The Prioritized-Layered Projection Algorithm. PLP attempts to prioritize the rendering of geometry along layers of occlusion. Cells that have been projected by the PLP algorithm are highlighted in red wireframe and their associated geometry is rendered, while cells that have not been projected are shown in green. Notice that the cells occluded by the desk are outlined in green, indicating that they have not been projected.



(a) (b)

Figure 15: The input geometry is a model of an office. (a) snapshot of the PLP algorithm highlights the spatial tessellation used. The cells which have not been projected in the spatial tessellation are highlighted in green. (b) This figure illustrates the accuracy of PLP. Shown in red are the pixels which PLP misses. In white, we show the pixels PLP renders correctly.

sellation. The basic idea of the algorithm is to carve the tessellation along *layers of polygons.* We define the layering number $\zeta \in \aleph$ of a modeling primitive $\mathcal{P}$ in the following intuitive way. If we order each modeling primitive along each pixel by its positive (assume, without loss of generality, that $\mathcal{P}$ is in the view frustum) distance to the eye point, we define $\zeta(\mathcal{P})$ as the smallest rank of $\mathcal{P}$ over all the pixels to which it contributes. Clearly, $\zeta(\mathcal{P}) = 1$ if and only if $\mathcal{P}$ is visible. Finding rank 1 primitives is equivalent to solving the visibility problem. Instead of solving this difficult problem, the PLP algorithm uses simple heuristics. The traversal algorithm *attempts* to project the modeling primitives by layers, that is, all primitives of rank 1, then 2, and so on. We do this by always projecting the cell in the front $\mathcal{F}$ (we call *the front*, the collection of cells that are immediate candidates for projection) which is least likely to be occluded according to its solidity value. Initially, the front is empty, and as cells are inserted, we estimate its accumulated solidity value to reflect its position during the traversal. Every time a cell in the front is projected, all of the geometry assigned to it is rendered.

PLP is very effective in finding the visible polygons. For more details about PLP, including comprehensive results, see [43, 42].

7 Image-space occlusion culling

As the name suggests image-space algorithms perform the culling in the viewing coordinates. The key feature in these algorithms is that during rendering of the scene the image gets filled up and subsequent objects can be culled away quickly by the already-filled parts of the images. Since they operate on a discrete array of finite resolution they also tend to be simpler to implement and more robust than the object-space ones, which tend to have numerical precision problems.

Since testing each individual polygon against the image is too slow, almost all the algorithms that we will describe here, use conservative tests. They place a hierarchy on the scene, with the lowest level usually being the bounding boxes of individual objects, and they perform the occlusion test on that hierarchy. Approximate solutions can also be produced by some of the image-space algorithms by classifying as occluded geometry parts which are visible through an insignificant pixel count. This invariably results in an increase in running speed.

When the scenes are composed of many small primitives without well-defined large occluders then performing the culling in image-space becomes more attractive. The projections of many small and individually insignificant occluders can be accumulated on the image using standard graphics rasterizing hardware, to cover a significant part of the image which can then be used for culling. Another advantage of these methods is that the occluders do not have to be polyhedral; any object that can be rasterised can be used.

7.1 Hierarchical Z-buffer

The Hierarchical Z-buffer (HZB) [35, 36] is an extension of the popular HSR method, the Z-buffer. In this method, occlusion is determined by testing against the *Z-pyramid*. The Z-pyramid is a layered buffer with different resolution at each level. At the finest level it is just the content of the Z-buffer, each coarser level is created by halving the resolution in each dimension and each element holding the furthest Z-value in the corresponding 2x2 window of the finer level below. This is done all the way to the top, where it is just one value corresponding to the furthest Z-value in the buffer. During scan-conversion of the primitives, if the contents of the Z-buffer change then the new Z-values are propagated up the pyramid to the coarser levels.

In [35] the scene is arranged into an octree which is traversed top-down front-to-back and each node is tested for occlusion. If at any point a node is found to be occluded then it is skipped; otherwise any primitives associated with it are rendered and the Z-pyramid is updated. To determine whether a node is visible, each of its faces is tested hierarchically against the Z-pyramid. Starting from the coarsest level, the nearest Z value of the face is compared with the value in the Z-pyramid. If the face is found to be further away then it is occluded; otherwise it recursively descends down to finer levels until its visibility can be determined.

To allow for real-time performance, a modification of the hardware Z-buffer is suggested that allows for much of the culling processing to be done in the hardware. In the absence of the custom hardware the process can be somewhat accelerated through the use of temporal coherence, by first rendering the geometry that was visible from the previous frame and building the Z-pyramid from its Z-buffer.

7.2 Hierarchical occlusion map

The hierarchical occlusion map method [80] is similar in principle to the HZB, though, it was designed to work with current graphics hardware and also supports approximate visibility culling; objects that are visible through only a few pixels can be culled using an opacity threshold. The occlusion is arranged hierarchically in a structure called the *Hierarchical Occlusion Map* (HOM) and the bounding volume hierarchy of the scene is tested against it. However, unlike the HZB, the HOM stores only opacity information while the distance of the occluders (Z-values) is stored separately. The algorithm then needs to independently test objects for overlap with occluded regions of the HOM and for depth.

During preprocessing, a database of potential occluders is assembled. Then at run-time, for each frame, the algorithm performs two steps: construction of the HOM and occlusion culling of the scene geometry using the HOM.

To build the HOM, a set of occluders is selected from the occluder database and rendered into the frame-buffer. At this point only occupancy information is required; therefore texturing, lighting and Z-buffering are all turned off. The occluders are rendered as pure white on a black background. The result is read from the buffer and forms the highest resolution in the occlusion map hierarchy. The coarser levels are created by averaging squares of 2x2 pixels to form a map which has half the resolution on each dimension. Texturing hardware can provide some acceleration of the averaging if the size of the map is large enough to warrant the set-up cost of the hardware. As we proceed to coarser levels the pixels are not just black or white (occluded or visible) but can be shades of grey. The intensity of a pixel at such a level shows the opacity of the corresponding region.

An object is tested for occlusion by first projecting its bounding box onto the screen and finding the level in the hierarchy where the pixels have approximately the same size as the extent of the projected box. If the box overlaps pixels of the HOM which are not opaque, it means that the box cannot be culled. If the pixels are opaque (or have opacity above the specified threshold when approximate visibility is enabled)

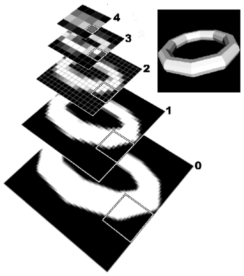


Figure 16: A hierarchy of occlusion maps created by recursively averaging blocks of pixels. Courtesy of Hansong Zhang, UNC.

then the object is projected on a region of the image that is covered. In this case a depth test is needed to determine whether the object is behind the occluders.

In paper [80] a number of methods are proposed for testing the depth of the objects against that of the occluders. The simplest test makes use of a plane placed behind all the occluders; any object that passes the opacity test is compared with this. Although this is fast and simple it can be over-conservative. An alternative is the *depth estimation buffer* where the screen space is partitioned into a set of regions and a separate plane is used for each region of the partition.

7.3 Directional discretized occluders

The Directional discretized occluders (DDOs) approach is similar to the HZB and HOM methods in that it also uses both object- and image-space hierarchies. In their preprocessing stage, Bernardini et al. [7] approximate the input model with an octree and compute simple, view-dependent polygonal occluders to replace the complex input geometry in subsequent visibility queries. Each face of every cell of the octree is regarded as a potential occluder and the solid angles spanning each of the two halfspaces on the two sides of the face are partitioned into regions. For each region, they compute and store a flag that records whether that face is a valid occluder for any viewpoint contained in that region. Each square, axis-aligned face is a view-dependent polygonal occluder that can be used in place of the original geometry in subsequent visibility queries.

The rendering algorithm visits the octree in a top-down, front-to-back order. Valid occluders found during the traversal are projected and added to a two-dimensional data structure, such as a quadtree. Each octree node is first tested against the current collection of projected occluders: if the node is not visible, traversal of its subtree stops. Otherwise, recursion continues and if a visible leaf node is reached, its geometry is rendered.

The DDO preprocessing stage is not inexpensive, and may take in the order of hours for models containing hundreds of thousands of polygons. However, the method does have several advantages if one can tolerate the cost of the preprocessing step. The computed occluders are all axis-aligned squares, a fact that can be exploited to design efficient data structures for visibility queries. The memory overhead of the DDOs is only six bitmasks per octree node. The DDO approach also benefits from *occluder fusion* and does not require any special or advanced graphics hardware. The approach could be used within the framework of other visibility culling methods as well. Culling methods which need to pre-select large occluders, (*e.g. Coorg and Teller* [18]), or which pre-render occluders to compute occlusion maps, (*e.g.* Zhang, *et Al.* [81]), could benefit from the DDO preprocessing step to reduce the overhead of visibility tests.

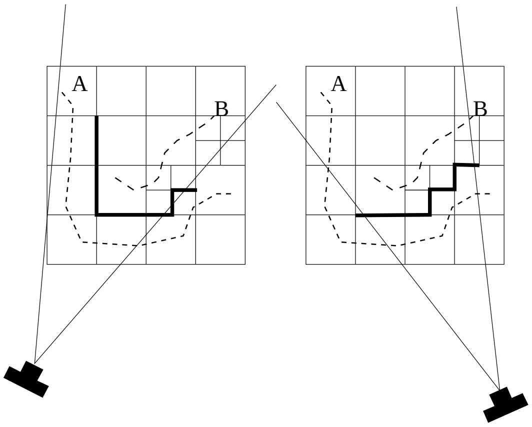


Figure 17: Illustration of the DDO approach. The input geometry, *A* and *B*, is drawn as dashed lines. The valid occluders for the two viewpoints are shown as thick solid lines. Courtesy of James Klosowski, IBM.

Figure 17 is a two-dimensional illustration of the DDO approach. The grid is a discretization of the space surrounding the scene; it represents our octree nodes. The input geometry, *A* and *B*, is shown using dashed lines. For the purpose of occlusion culling, the geometry *A* can be replaced by a simpler object (shown using thick solid lines) which is a subset of the grid edges, that is, the octree faces. The two figures show the same scene from different viewpoints and view directions. Note that the subset of grid edges that can act as occluders (in place of geometry *A*) changes as the viewpoint changes.

7.4 OpenGL-assisted occlusion culling

Bartz et al. in [6, 5] describe a different method of image-space culling. The scene is arranged in a hierarchical representation and tested against the occluded part of the image, which resembles the HZB and the HOM. However, in contrast to these methods, there is no hierarchical representation of the occlusion, rather OpenGL calls are used to query the hardware for visibility information. Both view-frustum and occlusion culling are done in that way.

For view-frustum culling the *OpenGL selection mode* is used. The selection mode can track a certain region of the screen and identify whether a given object is rendered onto it. By setting the tracked region to be the entire screen and rendering hierarchically the bounding volumes of the objects, it can quickly be decided on which to intersect the view volume. Of course the rendering of the bounding volumes here is purely for selecting the objects and does not contribute to the frame-buffer.

To test for occlusion, a separate buffer, the *virtual occlusion buffer*, is associated with the frame-buffer to detect the possible contribution of any object to the frame-buffer. This was implemented with a stencil buffer. The bounding boxes of the scene are hierarchically sent down the graphics pipeline. As they are rasterised, the corresponding pixels are set in the virtual occlusion buffer whenever the z-buffer test succeeds. The frame-buffer and the z-buffer remain unaltered throughout this process.

The virtual occlusion buffer is then read and any bounding box that has a footprint in it is considered to be (at least partially) visible and the primitives within it can be rendered. Since the operation of reading the virtual occlusion buffer can be very expensive, it was proposed to sample it by reading only spans from it. The sampling inevitably makes the algorithm a non-conservative test.

As in the methods above, approximate culling can be implemented if we allow boxes that have a small footprint in the occlusion buffer to be considered invisible. The performance of the algorithm depends on the hardware being used. In low- to mid-range graphics workstations where part of the rendering process is in software, the reduction in rendered objects can provide significant speed-ups. On high-end machines the set-up for reading the buffer becomes a more significant portion of the overall time, reducing the usefulness of the method.

7.5 Hardware assisted occlusion culling

Hardware vendors have started adopting occlusion-culling features into their designs. Greene et al. [35] report that the Kubota Pacific Titan 3000 was an early example of graphics hardware that supported occlusion-culling features.

A hardware feature available on HP machines (which seems quite similar to the Kubota Pacific Titan 3000) makes it possible to determine the visibility of objects as compared to the current values in the z-buffer. The idea is to add a feedback loop to the hardware which is able to check if changes will be made to the z-buffer when scan-converting a given primitive. One possible use of this hardware feature is to avoid rendering a very complex set model by first checking if it is potentially visible. In general this can be done with the HP occlusion-culling extension by checking whether an enveloping primitive (usually the bounding box of the object, but in general it might be more efficient to use an enclosing k-dop [41]) is visible, and only rendering the actual object if the simpler enclosing object is indeed visible.

The actual hardware feature as implemented on the HP fx series graphics accelerators is explained in [62] and [63]. One way to use the hardware is to query whether the bounding box of an object is visible. This can be done as follows:

```
glEnable(GL_OCCLUSION_TEST_HP);
glDepthMask(GL_FALSE);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
DrawBoundingBoxOfObject();
bool isVisible;
glGetBooleanv(GL_OCCLUSION_RESULT_HP, &isVisible);
glDisable(GL_OCCLUSION_TEST_HP);
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
```

Clearly, if the bounding box of an object is not visible, the object itself, which potentially could contain a large amount of geometry, must not be visible. This hardware feature is implemented in several of HP's graphics accelerators, for instance, the HP fx6 graphics accelerator. Severson [63] estimates that performing an occlusion-query with a bounding box of an object on the fx6 is equivalent to rendering about 190 25-pixel triangles. This indicates that a naive approach where objects are constantly checked for occlusion might actually hurt performance, and not achieve the full potential of the graphics board. In fact, it is possible to slow down the fx6 considerably if one is unlucky enough to project the polygons in a back-to-front order (because none of the primitives would be occluded).

Meissner et al. [49] propose an effective occlusion culling technique using this hardware test. In a preprocessing step, a hierarchical data structure is built which contains the input geometry. (In their paper, they propose several different data structures, and study their relative performance.) Their algorithm is as follows:

(1) traverse the hierarchical data structure to find the leaves which are inside the view frustum;

(2) sort the leaf cells by the distance between the viewpoint and their centroids;

(3) for each sorted cell, render the geometry contained in the cell **only** if the cell boundary is visible.

In their recent offerings, HP has improved the occlusion-culling features. The fx5 and fx10 hardware can perform several occlusion culling queries in parallel [19]. Also, HP reports that their OpenGL implementations have been changed to use the occlusion-culling features automatically when possible. For instance, before rendering a long display list, HP software would actually perform an occlusion query before rendering all the geometry.

ATI's HyperZ technology [51] is another example of a hardware-based occlusion-culling feature. HyperZ has three different optimizations which they claim greatly improve the performance of 3D applications. The main trust on all the optimizations is on lowering the memory bandwidth required for updating the Z-values (which they claim is the single largest user of bandwidth on their cards). One optimization is a technique for lossless compression of Z-values. Another is a "fast" Z-buffer clear, which performs a lazy clear of depth values. ATI also reports on the implementation of the hierarchical Z-buffer of Greene et al. [35] in hardware. Details of the actual features are sketchy, and at this point ATI has not exposed any of the functionality of their hardware to applications, that is, applications are blind, and should automatically get improved performance.

There are reports that other vendors, including SGI, Nvidia, and so on, are working on similar occlusion-culling features for their upcoming hardware.

7.6 Discussion

There are several other algorithms which are targeted at particular types of scenes. For example, the occluder shadows proposed by Wonka and Schmalstieg [75] specifically target urban environments. In this work the scene is partitioned in a regular 2D grid. During run-time a number of occluders are selected and their 'shadows' - the planes defined by the view-point and the top edge of each occluder - are rendered into an auxiliary buffer called the cull-map. Each pixel in the cull-map (image-space) corresponds to a grid cell of the scene grid (object-space). If the cull-map pixel is not covered then objects in the corresponding scene grid cell are potentially visible.

Hong et al. [38] use an image-based portal technique (similar in some respects to the cells- and-portals work of Luebke and Georges [48]) to be able to fly through a virtual human colon in real-time. The colon is partitioned into cells at preprocessing and these are used to accelerate the occlusion with the help of a Z-buffer at run-time.

One drawback of the techniques described in this section is that they rely on being able to read information from the graphics hardware. Unfortunately, on most current architectures, using any sort of feedback from the graphics hardware is quite slow and places a limit on the achievable frame rate of such techniques. As Bartz et al. [5] show, these methods are usually only effective when the scene complexity is above a large threshold.

There are other shortcomings to these techniques. One of the main problems is the need for preselection of occluders. Some techniques, such as HOM, need to create different versions of the actual objects (through some sort of "occlusion-preserving" simplification algorithm) to be able to generate the occlusion-maps. Another interesting issue is how to deal with dynamic scenes. The more preprocessing used, the more expensive it is to deal with dynamic environments.

The BSP tree method introduced by Naylor [53] already in 1992 can be thought of as somewhere between image-precision and object-precision, since although he used a 2D BSP tree in image-space for culling the 3D scene, this was done using object-precision operations rather than image-precision.

8 From-region visibility

In a typical visibility culling algorithm the occlusion is tested from a point [18, 39]. Thus, these algorithms are applied in each frame during the interactive walkthrough. A promising alternative is to find the PVS from a region or viewcell, rather than from a point. The computation cost of the PVS from a viewcell would then be amortized over all the frames generated from the given viewcell.

Effective methods have been developed for indoor scenes [70, 30], but for general arbitrary scenes, the computation of the visibility set from a region is more involved than from a point. Sampling the visibility from a number of view points within the region [33] yields an approximated PVS, which may then cause unacceptable flickering artifacts during the walkthrough. Conservative methods were introduced in [15, 59] which are based on the occlusion of individual large convex objects.

In these methods a given object or collection of objects is culled away if and only if they are fully occluded by a single convex occluder. It was shown that a convex occluder is effective only if it is larger than the viewcell [52]. However, this condition is rarely met in real applications. For example, the objects in Figure 18 are smaller than the viewcell, and their umbrae (with respect to the viewcell) are rather small. Their union does not occlude a significant portion of the scene (see in (a)), while their aggregate umbra is large (see in (b)).

Recently, new techniques were developed in which the visibility culling from a region is based on the combined occlusion of a collection of objects (occluder fusion). The collection or cluster of objects that contributes to the aggregate occlusion has to be neither connected nor convex. The effective from-region culling of these techniques is significantly larger than previous from-region visibility methods. Below, four techniques are described followed by a discussion.

8.1 Conservative volumetric visibility with occluder Fusion

Schaufler et al. [60] introduce a conservative technique for the computation of viewcell visibility. The method operates on a discrete representation of space and uses the opaque interior of objects as occluders. This choice of occluders facilitates their extension into adjacent opaque regions of space, in essence, maximizing their size and impact.

The method efficiently detects and represents the regions of space hidden by occluders and is the first to use the property that occluders can also be extended into empty space provided this space itself is occluded from the viewcell. This is proved to be effective for computing the occlusion by a set of occluders, successfully realizing occluder fusion.

Initially, the boundary of objects is rasterized into the discretization of space and the interior of these boundaries is filled with opaque voxels. For each viewcell, the occlusion detection algorithm iterates over these opaque voxels, and groups them with adjacent opaque voxels into effective blockers. Subsequently, a shaft is constructed around the viewcell and the blocker to delimit the region of space hidden by the blocker. This classification of regions of space into visible and hidden is noted in the spatial data structure. As regions of space have already been found to be hidden from the viewcell, extension of blockers into neighboring voxels can also proceed into these hidden regions realizing occluder fusion with all the occluders which caused this region to be hidden.

As an optimization, opaque voxels are used in the order from large to small and from front to back. Occluded opaque voxels are not considered further as blockers.

To recover the visibility status of objects in the original scene description, the space they occupy is looked up in the spatial data structure and, if all the voxels intersected by the object are classified as hidden, the object is guaranteed to be hidden as well.

The authors present specialized versions for the cases of 2D and 2 1/2D visibility, and motivate the ease of extension to 3D: because only two convex objects at a time are considered in the visibility classification (the viewcell and the occluder), the usual difficulties of extending visibility algorithms from 2D to 3D, caused by triple-edge events, are avoided. Example applications described in the paper include visibility preprocessing for real-time walkthroughs and reduction in the number of shadow rays required by a ray-tracer (see [60] for details).

8.2 Conservative visibility preprocessing using extended projections

Durand et al. [25] (see also [47]) present an extension of point-based image-space methods such as the Hierarchical Occlusion Maps [81] or the Hierarchical Z-buffer [35] to volumetric visibility from a view-cell, in the context of preprocessing PVS computation. Occluders and occludees are projected onto a plane, and an occludee is declared hidden if its projection is completely covered by the cumulative projection of occluders (and if it lies behind). The projection is however more involved in the case of volumetric visibility: to ensure conservativeness, the Extended Projection of an occluder underestimates its projection from any point in the view-cell, while the Extended Projection of an occludee is an overestimation (see Figure 20(a)). A discrete (but conservative) pixel-based representation of extended projections is used, called an *extended*

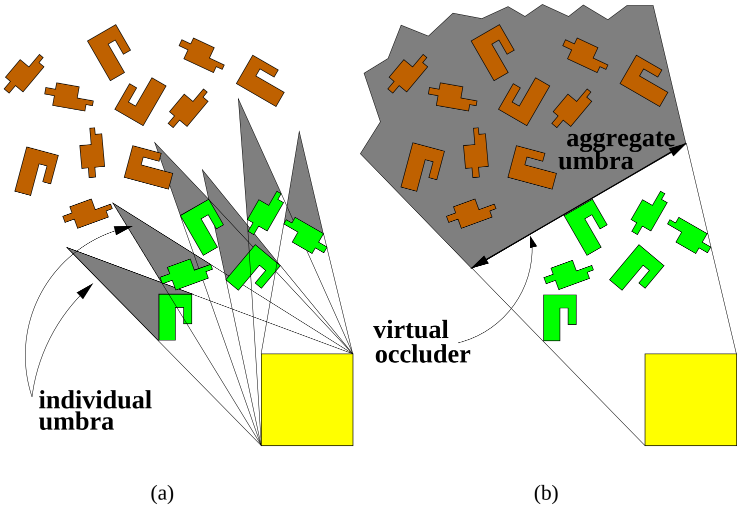


Figure 18: The union of the umbrae of the individual objects is insignificant, while their aggregate umbra is large and can be represented by a single virtual occluder.

depth map. Extended projections of multiple occluders aggregate, allowing occluder-fusion, that is, the cumulative occlusion caused by multiple occluders. For convex view-cells, the extended projection of a convex occluder is the intersection of its projections from the vertices of the cell. This can be computed efficiently using the graphics hardware (stencil buffer) and a conservative rasterization. Concave occluders intersecting the projection plane are sliced (see [25] for details).

A single set of six projection planes can be used, as demonstrated by an example involving a city database. The position of the projection plane is however crucial for the effectiveness of Extended Projections. This is why a reprojection operator was developed for hard-to-treat cases. It permits a group of occluders to be projected onto one plane where they aggregate, and then reproject this aggregated representation onto a new projection plane (see Figure 20(b)). This re-projection is used to define an occlusion-sweep where the scene is swept by parallel planes leaving the cell. The cumulative occlusion obtained on the current plane is reprojected onto the next plane as well as new occluders. This allows the handling of very different cases such as the occlusion caused by leaves in a forest.

8.3 Virtual occluders

Koltun et al. [44] introduce the notion of from-region virtual occluders. Given a scene and a viewcell, a virtual occluder is a view-dependent (simple) convex object, which is guaranteed to be fully occluded from any given point within the viewcell and which serves as an effective occluder from the given viewcell. Virtual occluders compactly represent the aggregate occlusion for a given cell. The introduction of such view-dependent virtual occluders enables to apply an effective region-to-region or cell-to-cell culling technique and to efficiently compute a potential visibility set (PVS) from a region/cell. The paper presents an object-space technique that synthesizes such virtual occluders by aggregating the visibility of a set of individual occluders. It is shown that only a small set of virtual occluders is required to compute the PVS efficiently on-the-fly during the real-time walkthrough.

In the preprocessing stage several objects are identified as seed objects. For each seed object, a cluster of nearby objects is constructed so that a single virtual occluder faithfully represents the occlusion of this cluster of objects. At first, the cluster is defined to include only the seed object. Then, iteratively, at each step, more objects which satisfy a geometric criterion are added to the cluster of occluders, thus augmenting the aggregate umbra of the cluster. The virtual occluder is placed just behind the furthest object in the cluster, and is completely contained in the aggregate umbra of the cluster (see Figs. 18 and 21).

One virtual occluder is stored at each step of the iteration. As a result, at the end of the process, there is a large and highly redundant group of virtual occluders. This group can be well represented by a small subset of the most effective virtual occluders.

In the real-time rendering stage, the PVS of a viewcell is computed just before the walkthrough enters the viewcell. It is done by hierarchically testing the scene-graph nodes against the virtual occluders. Since only a very small number of them are used, this test is extremely fast.

The 3D problem is solved by a 2.5D implementation, which proves to be effective for most typical scenes, such as urban and architectural walkthroughs. The 2.5D implementation performs a series of slices in the height dimension, and uses the 2D algorithm to construct 2D virtual occluders in each slice. These occluders are then extended to 3D by giving them the height of their respective slices.

8.4 Occluder fusion for urban walkthroughs

Wonka et al. [76] present an approach based on the observation that it is possible to compute a conservative approximation of the umbra for a viewcell from a set of discrete point samples placed on the viewcell's boundary. A necessary, though

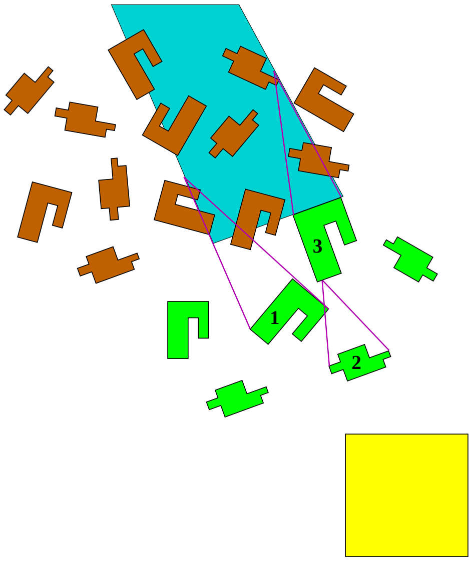


Figure 19: The individual umbrae (with respect to the yellow viewcell) of objects 1, 2 and 3 do not intersect, but yet their occlusion can be aggregated into a larger umbra.)

not sufficient condition that an object is occluded is that it is completely contained in the intersection of all sample points' umbrae. Obviously, this condition is not sufficient as there may be viewing positions between the sample points where the considered object is visible.

However, shrinking an occluder by ε provides a smaller umbra with a unique property: an object classified as occluded by the shrunk occluder will remain occluded with respect to the original larger occluder when moving the viewpoint no more than ε from its original position.

Consequently, a point sample used together with a shrunk occluder is a conservative approximation for a small view cell with radius ε centered at the sample point. If the original view cell is covered with sample points so that every point on the boundary is contained in an ε -neighborhood of at least one sample point, then an object lying in the intersection of the umbrae from all sample points is occluded for the original viewcell.

Using this idea, multiple occluders can be considered simultaneously. If the object is occluded by the joint umbra of the shrunk occluders for every sample point of the viewcell, it is occluded for the whole view cell. In that way, occluder fusion for an arbitrary number of occluders is implicitly performed (see Figure 22 and Figure 23).

8.5 Discussion

When the visibility from a region is concerned, occlusion caused by individual occluders in a general setting is insignificant. Thus, it is essential to take advantage of aggregate occlusion caused by groups of nearby objects. The above four papers address the problem of occlusion aggregation also referred to as occluder fusion.

All four techniques are conservative; they aggregate occlusion in most cases, but not in all possible ones. In some techniques, the criterion to fuse two occluders or to aggregate their occlusions is based on the intersection of two umbrae. However, in [44, 77], a more elaborate criterion is used, which permits aggregation of occlusions even in cases where the umbrae are not necessarily intersected. These cases are illustrated in Figure 19.

To cope with the complexity of the visibility in 3D scenes, all the techniques use some discretizations.

The first method discretizes the space into voxels, and operates only on voxels. This leads to the underestimation of occlusion when the umbra of occluders is relatively small and partially overlaps some large voxels, but does not completely contain any. The advantage of this approach is its generality: it can be applied to any representation of 3D scenes, and not necessarily polygonal.

The second method discretizes the space in two ways. First, it projects all objects onto a discrete set of projection planes, and second, the representation of objects in those planes is also discrete. Moreover, 3D projections are replaced by two 2D projections (see Figure 20), to avoid performing analytical operations on objects in 3D space. The advantage of this algorithm is that, since most operations are performed in image-space, they can be hardware-assisted to shorten the preprocessing time.

The third method is object-space analytical in the 2D case. It treats the 3D cases as a 2.5D scene and solves it by a series of 2D cases by discretizing the height dimension. It is shown that in practice the visibility of 2.5D entities approximate well the visibility of the original 3D models.

The forth method samples the visibility from a viewcell from a discrete number of sample points. Although it underestimates occlusion, it is also a conservative method. This may be insignificant in the case of close and large occluders, but in cases where the occlusion is created by a large number of small occluders, the approximation might be too crude.

Something that could prove useful when computing visibility from a region is a method for depth-ordering objects with respect to the region. Finding such an ordering can be a challenging task, if at all possible, since it might vary at different sample points in the given region. Chrysanthou in [12] (Section 3.2) suggests a hybrid method based on Graph Theory and

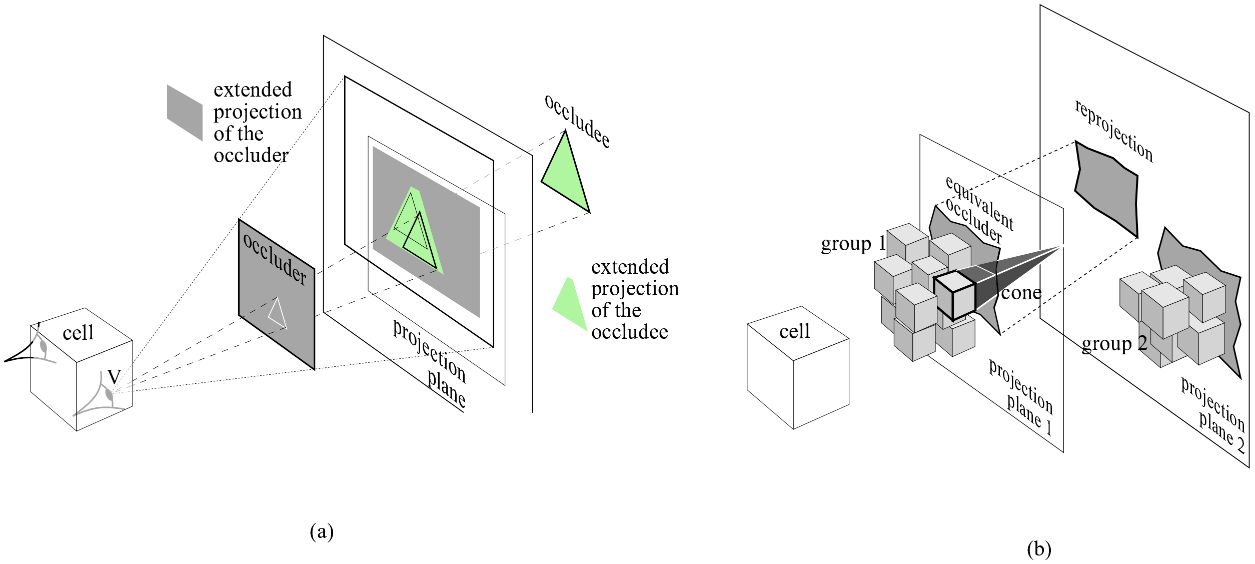


Figure 20: (a) Principle of Extended Projections. The Extended Projection of the occluder is the intersection of its projections from all the points in the viewing cell, while the Extended Projection of the occludee is the union of its projections. (b) If plane 2 is used for projection, the occlusion of group 1 is not taken into account. The shadow cone of the cube shows that its Extended Projection would be void, since it vanishes in front of the plane. The same constraint applies for group 2 and plane 1. We thus project group 1 onto plane 1, then reproject this aggregate projection onto plane 2. Courtesy of Fredo Durand, MIT.

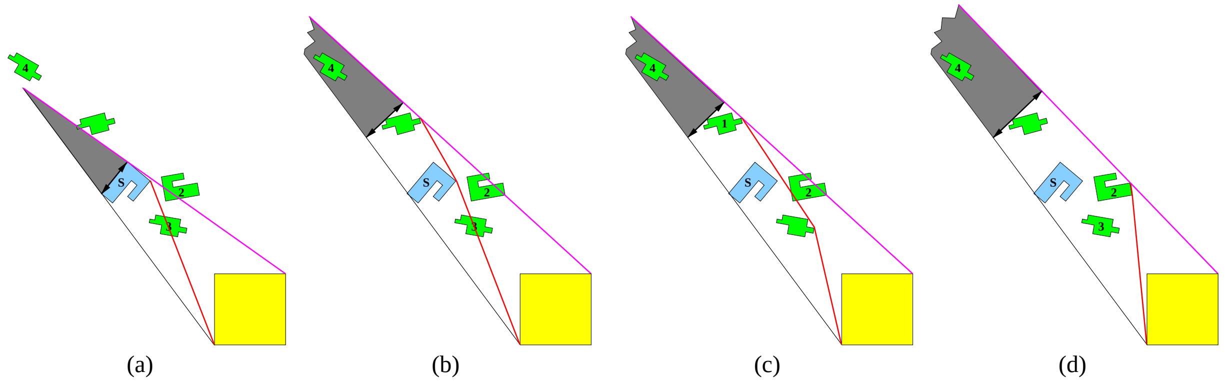


Figure 21: Growing the virtual occluders by intersecting objects with the active separating and supporting lines.

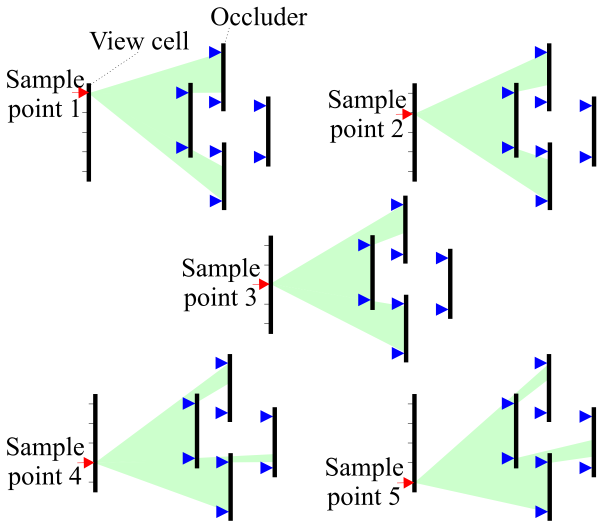


Figure 22: Sampling of the occlusion from five sampling points. Courtesy of Peter Wonka, Vienna University of Technology.

BSP trees which will sort a set of polygons as far as possible and report unbreakable cycles where they are found.

8.6 Approximate from-region visibility

In [2] a scheme to combine approximate occlusion culling with levels-of-detail (LOD) techniques is presented. The idea is to identify partially-occluded objects in addition to fully-occluded ones. The assumption is that partially-occluded objects take less space on the screen, and therefore can be rendered using a lower LOD. The authors use the term *Hardly-Visible Set* (HVS) to describe a set consisting of both fully and partially visible objects.

A set of occluders is selected and simplified to a collection of Partially-overlapping boxes. Occlusion culling is performed from the viewcell using these boxes as occluders to find the "fully-visible" part of the HVS. It is performed considering only occlusion by individual boxes [15, 59]. There is no occlusion fusion, but a single box may represent several connected occluder objects.

To compute partially-visible objects, all the occluders (boxes) are enlarged by a certain small degree, and occlusion culling is performed again using these magnified occluders. The objects that are occluded by the enlarged occluders and not by the original ones are considered to be partially occluded from the viewcell, and are thus candidates to be rendered at a lower LOD.

Several parts of the HVS are computed by enlarging the occluders several times, each time by a different degree, thus, classifying objects with a different degree of visibility. During real-time rendering, the LOD is selected with respect to the degree of visibility of the objects.

It should be noted that this basic assumption of the degree of visibility is solely heuristic, since an object partially occluded from a region does not mean it is partially occluded from any point within the region. It could be fully visible at one point and partially visible or occluded at another.

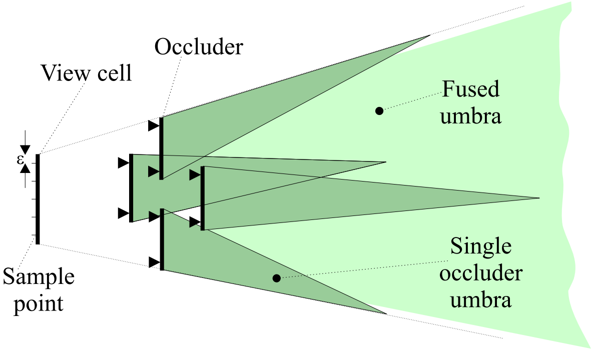


Figure 23: The fused umbra from the five points (in the figure above) is the intersection of the individual umbrae. It is larger than the union of umbrae of the original viewcell. Courtesy of Peter Wonka.

In [33] another approximate from-region visibility technique is proposed. Casting rays from a five-dimensional space samples the visibility. The paper discusses how to minimize the number of rays cast to achieve a reliable estimate of the visibility from a region.

8.7 The PVS storage space problem

Precomputing the PVS from a region requires solving a prominent space problem. The scene is partitioned into viewcells and for each cell a PVS is precomputed and stored readily for the online rendering stage. Since the number of viewcells is inherently large, the total size of all the visibility sets is much larger than the original size of the scene. Aside for a few exceptions this problem has not received enough attention yet. Van de Panne and Stewart [72] present a technique to compress precomputed visibility sets by clustering objects and viewcells of similar behavior. Gotsman et al. [33] present a hierarchical scheme to encode the visibility efficiently. Cohen-Or et al. [16, 15] deal with the transmission of visibility sets from the server to the client and in [15, 52] discuss the selection of the best viewcell size in terms of the size of the PVS.

A completely different approach was taken by Koltun et al. [44]. The PVS of each viewcell does not need to be stored explicitly. An intermediate representation that requires much less storage space than the PVS is created and used to generate the PVS on-the-fly during rendering.

9 Conclusion

In this paper, we have surveyed most of the visibility literature available in the context of walkthrough applications. We see that a considerable amount of knowledge has been assembled in the last decade; in particular, the number of papers in the area has increased substantially in the last couple of years. It is hard to say exactly where the field is heading, but there are some interesting trends and open problems.

It seems further research is necessary into techniques which lower the amount of preprocessing required. Also, memory is a big issue for large scenes, especially in the context of from-region techniques.

It is expected that more hardware features which can be used to improve visibility computations will be available. At present, a major impediment is the fact that reading back information from the graphics boards is very slow. It is expected that this will get much faster, enabling improvements in hardware-based visibility culling algorithms. The efficient handling of dynamic scenes is an open area of research at this point.

Acknowledgements

We would like to thank all the authors who have helped us with material from their work. In particular, we thank Fredo Durand, Vladlen Koltun, Pere Brunet, Seth Teller, James Klosowski, Graig Gotsman, Gernot Schaufler and Peter Wonka, for helping us with text and figures from their papers, and Joseph Mitchell for useful comments on this survey.

This research was partially supported by the Israel Science Foundation founded by The Israel Academy of Sciences and Humasnities-Center of Excellence Program, and by a grant of the Israeli Ministry of Science.

References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):41–50, March 1990.

[2] C. Andujar, C. Saona-Vazquez, I. Navazo, and P. Brunet. Integrating occlusion culling and levels of details through hardly-visible sets. *Computer Graphics Forum*, 19(3), 2000.

[3] A. Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[4] U. Assarsson and T. Moller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1), 2000.

[5] D. Bartz, M. Meiner, and T. Httner. Opengl-assisted occlusion culling for large polygonal models. *Computer & Graphics*, 23(5):667–679, 1999.

[6] D. Bartz, M. Messner, and T. Httner. Extending graphics hardware for occlusion queries in opengl. In *Proc. Workshop on Graphics Hardware '98*, pages 97–104, 1998.

[7] F. Bernardini, J. T. Klosowski, and J. El-Sana. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum*, 19(3), 2000.

[8] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98*, pages 207–219, June 1998.

[9] W. Jack Bouknight. A procedure for generation of three-dimensional half-toned computer graphics presentations. *Communications of the ACM*, 13(9):527–536, September 1970.

[10] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.d. thesis, University of Utah, December 1974.

[11] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *ACM Computer Graphics*, 23(3):99–106, 1989.

[12] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation*. PhD thesis, Queen Mary and Westfield College, University of London, February 1996.

[13] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[14] D. Cohen-Or, Y. Chrysanthou, C. Silva, and G. Drettakis. Visibility, problems, techniques and applications. SIGGRAPH 2000 Course Notes, July 2000.

[15] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[16] D. Cohen-Or and E. Zadicario. Visibility streaming for network-based walkthroughs. *Graphics Interface '98*, pages 1–7, June 1998.

[17] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 78–87, 1996.

[18] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. *1997 Symposium on Interactive 3D Graphics*, pages 83–90, April 1997.

[19] R. Cunniff. Visualize fx graphics scalable architecture. In *presentation at Hot3D Proceedings, part of Graphics Hardware Workshop*, 2000.

[20] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[21] D. P. Dobkin and S. Teller. Computer graphics. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 42, pages 779–796. CRC Press LLC, Boca Raton, FL, 1997.

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

[23] F. Durand. *3D Visibility: Analytical study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France, July 1999.

[24] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 89–100. ACM SIGGRAPH, Addison Wesley, August 1997.

[25] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[26] D. Eggert, K. Bowyer, and C. R. Dyer. Aspect graphs: State-of-the-art and applications in digital photogrammetry. In *Proc. ISPRS 17th Cong.: Int. Archives Photogrammetry Remote Sensing*, pages 633–645, 1992.

[27] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. In *Proceedings of Pacific Graphics 99*, pages 12–20, October 1999.

[28] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990. Overview of research to date.

[29] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *ACM Computer Graphics*, 14(3):124–133, 1980.

[30] T. A. Funkhouser. Database management for interactive display of large architectural models. *Graphics Interface*, pages 1–8, May 1996.

[31] J. Goldfeather, J. P. M. Hultquist, and H. Fuchs. Fast constructive-solid geometry display in the Pixel-Powers graphics system. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 107–116, August 1986.

[32] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 212–22, July 1984.

[33] C. Gotsman, O. Sudarsky, and J. Fayman. Optimized occlusion culling. *Computer & Graphics*, 23(5):645–654, 1999.

[34] N. Greene. Hierarchical polygon tiling with coverage masks. *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 65–74. ACM SIGGRAPH, Addison Wesley, August 1996.

[35] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231–240, 1993.

[36] N. Greene and M. Kass. Error-bounded antialiased rendering of complex environments. *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 59–66. ACM SIGGRAPH, ACM Press, July 1994.

[37] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *ACM Computer Graphics*, volume 25, pages 197–206, July 1991.

[38] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 27–34. ACM SIGGRAPH, Addison Wesley, August 1997.

[39] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1997.

[40] J. T. Klosowski and C. T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. Technical report. submitted for publication, 2000.

[41] J. T. Klosowski, M. Held, Joseph S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, January-March 1998.

[42] J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, April - June 2000. ISSN 1077-2626.

[43] J. T. Klosowski and C. T. Silva. Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99*, pages 115–122, October 1999.

[44] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, June 2000. ISBN 3-211-83535-0.

[45] S. Kumar, D. Manocha, W. Garrett, and M. Lin. Hierarchical back-face computation. *Computers and Graphics*, 23(5):681–692, October 1999.

[46] F.-A. Law and T.-S. Tan. Preprocessing occlusion for real-time selective refinement (color plate S. 221). In Stephen N. Spencer, editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 47–54, New York, April 26–28 1999. ACM Press.

[47] H. L. Lim. Toward a fuzzy hidden surface algorithm. In *Computer Graphics International*, Tokyo, 1992.

[48] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, April 1995.

[49] M. Meissner, D. Bartz, T. Huttner, G. Muller, and J. Einighammer. Generation of subdivision hierarchies for efficient occlusion culling of large polygonal models. *Computer & Graphics*, To appear.

[50] T. Moeller and E. Haines. *Real-Time Rendering*. A.K. Peters Ltd., 1999.

[51] S. Morein. Ati radeon hyper-z technology. In *presentation at Hot3D Proceedings, part of Graphics Hardware Workshop*, 2000.

[52] B. Nadler, G. Fibich, S. Lev-Yehudi, and D. Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computer & Graphics*, 23(5):655–666, 1999.

[53] B. F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, 1992.

[54] M. E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. *Proc. ACM Nat. Mtg.*, 1972.

[55] J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[56] M. S. Peercy, M. Olano, J. Airey, and P. Jeffrey Ungar. Interactive multi-pass programmable shading. *Proceedings of SIGGRAPH 2000*, pages 425–432, July 2000.

[57] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *Internat. J. Comput. Vision*, 5(2):137–160, 1990.

[58] H. Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3D scenes. In *Proceedings of Graphics Interface '93*, pages 166–173, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[59] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. *Computer & Graphics*, 23(5):635–644, 1999.

[60] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000.

[61] R. Schumacker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX,, September 1969.

[62] N. Scott, D. Olsen, and E. Gannet. An overview of the visualize fx graphics accelerator hardware. *The Hewlett-Packard Journal*, May:28–34, 1998.

[63] K. Severson. VISUALIZE Workstation Graphics for Windows NT. HP product literature.

[64] F. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In Robert Cook, editor, *ACM Computer Graphics*, Annual Conference Series, pages 145–152. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[65] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic cashing scheme. In S. Wilbur and M. Bergamasco, editors, *Proceedings of Framework for Immersive Virtual Environments FIVE*, December 1996.

[66] W. Stuerzlinger. Imaging all visible surfaces. *Graphics Interface '99*, pages 115–122, June 1999.

[67] B. Smits, J. Arvo, and D. Greenberg. A clustering algorithm for radiosity in complex environments. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435–442. ACM SIGGRAPH, ACM Press, July 1994.

[68] O. Sudarsky and C. Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29, January - March 1999.

[69] I. E. Sutherland, R. F. Sproull, and R. A. Schumaker. A characterization of ten hidden surface algorithms. *ACM Computer Surveys*, 6(1):1–55, March 1974.

[70] S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):61–69, July 1991.

[71] C. Trendall and A. James Stewart. General calculations using graphics hardware with applications to interactive caustics. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 287–298, June 2000.

[72] M. van de Panne and J. Stewart. Efficient compression techniques for precomputed visibility. In *Proceedings of Eurographics Workshop on Rendering '99*, 1999.

[73] G. S. Watkins. A real-time visible surface algorithm. Technical Report UTECH-CSc-70-101, University of Utah, Salt Lake City, Utah, 1970.

[74] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *ACM Computer Graphics*, 11(2):214–222, July 1977.

[75] P. Wonka and D. Schmalstieg. Occluder shadows for fast wakthroughs of urban environments. In Hans-Peter Seidel and Sabine Coquillart, editors, *Computer Graphics Forum*, volume 18, pages C51–C60. Eurographics Association and Blackwell Publishers Ltd 1999, 1999.

[76] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 71–82, June 2000. ISBN 3-211-83535-0.

[77] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. Technical Report TR-186-2-00-06, Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186, A-1040 Vienna, Austria, March 2000. human contact: technical-report@cg.tuwien.ac.at.

[78] A. Woo, P. Poulin, and A. Fourier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–31, 1990.

[79] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Computer Graphics (Proceedings of SIGGRAPH 97*, pages 77–88, 1997.

[80] H. Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. Ph.D. thesis, Department of Computer Science, UNC-Chapel Hill, 1998.

[81] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, August 1997.

Visibility, Problems, Techniques and Applications

Analytical visibility

Frédo Durand
MIT- Lab for Computer Science

Introduction

- Why bother with analytical visibility?
- Help understand
  - What are the problems
  - What do we want to do?
  - What is possible?
  - What is costly?
- Offer insights
- Can be simplified for practical solutions

Plan

- Spaces
- Visual events
- Aspects
- The Aspect Graph
- The Visibility Skeleton
- 3D vs. 2D

Spaces

- Analytical tools
  - To understand what's going on
- Computational tools
  - Some computations are easier in a certain space

Spaces

- Object-space
- Image-space
- Viewpoint
- Line-space

Object-space vs. Image-space

- [Sutherland et al. 1974]
- Aka image-precision and object-precision

Object-space

- 3D space where the scene is defined
- E.g., triangle is occluded if it is inside the pyramid

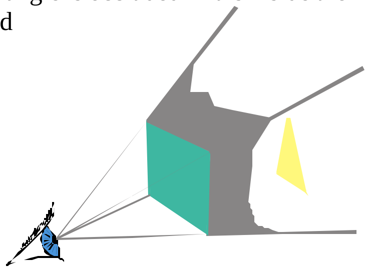
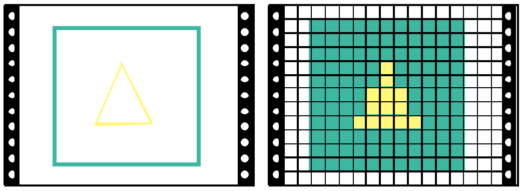


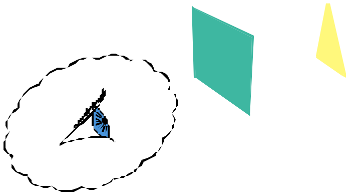
Image-space

- Computation performed in the plane of the image
- E.g. is triangle inside rectangle?
- Usually discretized in pixels



Viewpoint space

- Space of all possible viewpoints
- Often same as object-space
- Is the current viewpoint one of the viewpoints where triangle is occluded?

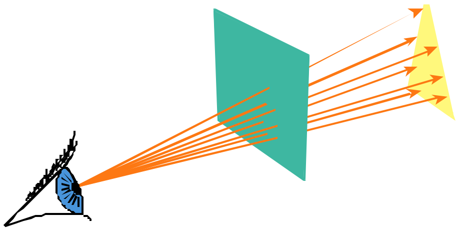


Viewpoint space

- Space of all possible viewpoints
- Often same as object-space
- But can be restrained
  - Orthographic projection (viewing sphere)
  - Limited degrees of freedom
- Fastest viewpoint-space method ever:
  - Precompute everything for every viewpoint!

Line space

- Visibility expressed in terms of rays
- E.g. are all rays between the eye and the triangle blocked by the rectangle?

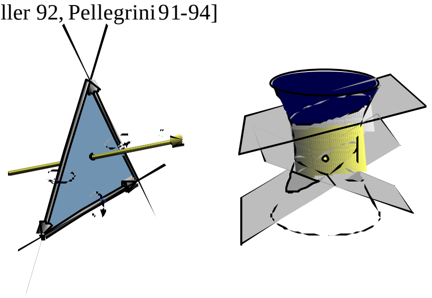


Line space

- Visibility expressed in terms of rays
- Paradigm: ray-casting
- Line space is 4D
  - E.g. intersection with two planes
  - Or one direction + intersection with one plane
- Set of line through a point is a 2D manifold in 4D
  - Defines a view
- Ray-space is 5D
  - Line + origin

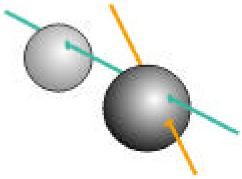
Line Space

- Plücker space
  - [Teller 92, Pellegrini 91-94]



Classification of lines

- Line intersecting an object
- Line intersecting two objects



Typical advantages and drawbacks

- Image-space
  - + Robust, easier to code, occluder fusion, can use polygon soup
  - – Limited to one viewpoint, aliasing
- Object-space
  - + Precision, can handle from-region visibility
  - – often robustness problems
- Viewpoint space
  - + Super efficient at runtime
  - – Costly storage and precomputation, no dynamic object
- Line space
  - + Natural space, simple atomic operation (ray-casting), arbitrary geometry
  - – 4D, often requires sampling (non conservative), or too complex

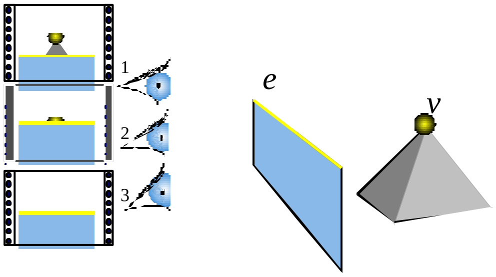
Visual event

- Where does visibility change?
- How does it change?

- Qualitative approach

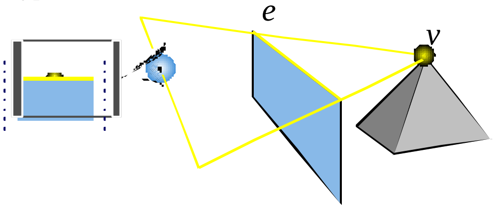
Visual event

- Appearance-disappearance of objects (qualitative change of a view)



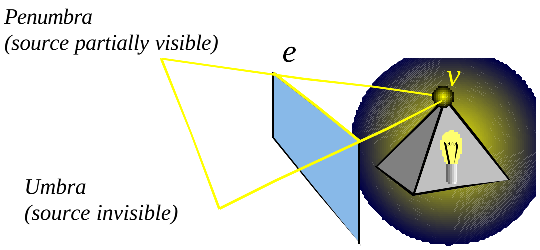
Visual event

- Appearance-disappearance of objects (qualitative change of a view)
- « Wedge » defined by a vertex and an edge
- Type EV

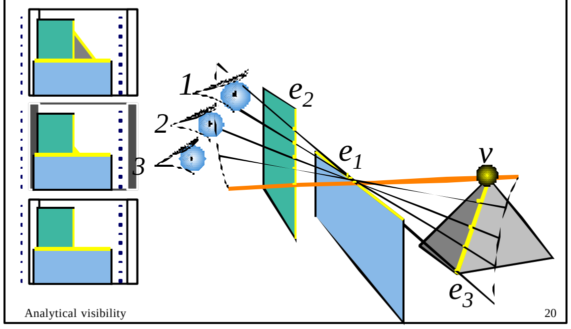


Visual event

- Appearance-disappearance of objects
- Limits of umbra

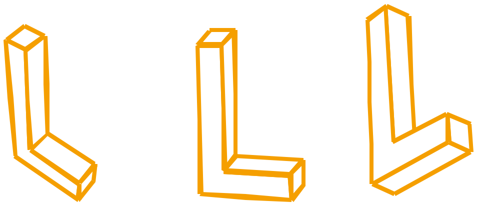


Triple-edge event



Aspect

- “Qualitative” view
- Topological



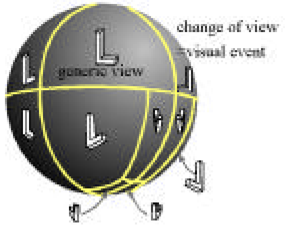
Qualitatively equivalent

(same aspect)

Qualitatively different (different aspect)

Aspect Graph

- Aspect graph
  - Characterization of the set of possible views of an object
  - [Koenderink and Van Doorn 79, Plantinga and Dyer 90, Gigus *et al.* 90-91, Petitjean *et al.* 92]



Size of the Aspect Graph

For a polygonal scene with *n* edges

- O(n^3) visual events
- O(n^6) for orthographic views
- O(n^9) for perspective views

- A more reasonable estimate may be around < O(n^4) and O(n^6), but still very costly!

Aspect graph for walkthroughs

- [Plantinga 93]
- Pre-compute all visibility events
- While walking through, you know when the set of visible objects changes
- Unfortunately, very costly pre-processing and storage

Aspect graph for walkthroughs II

(Forward reference)

- [Coorg and Teller 97]
- Local and linearized version of the aspect graph

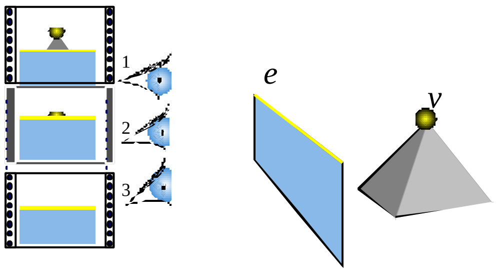
Visibility Skeleton

Goal

- Global visibility structure
  - views
  - limits of shadows
  - appearance of objects
  - mutual visibility
- Characterise the changes in visibility
  - where?
  - how?

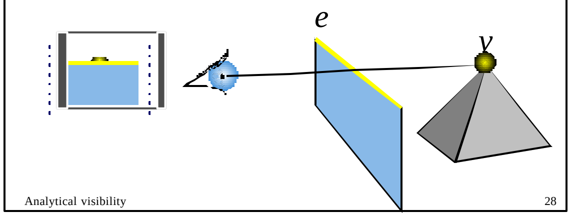
Visual event

- Appearance-disappearance of objects (qualitative change of a view)



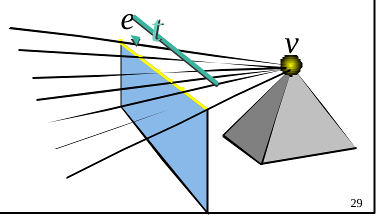
Critical line

- Line going through *e* and *v*



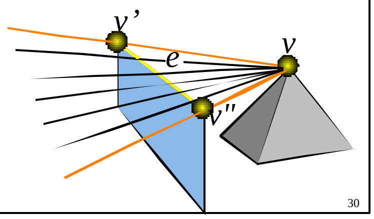
Critical lines

- 1D set of lines going through *e* and *v* (1 degree of freedom)



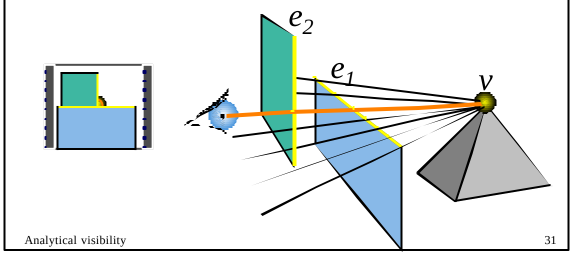
Extremal stabbing line

- 1D set of lines going through *e* and *v* (1 degree of freedom)
- Extremity: extremal stabbing line (VV) (0 degree of freedom)



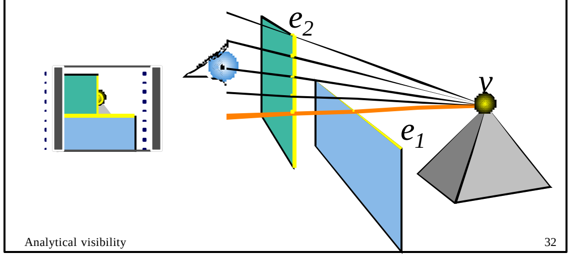
Extremal stabbing line

- Type *VEE* (0 degree of freedom)



Adjacent critical line set

- Generated by the second edge
- Same extremity ve_1e_2

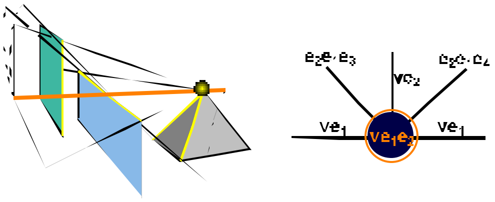


Summary

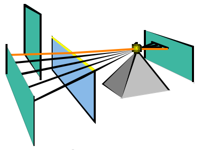
- Visual events
  - EV, EEE
- 1D critical line sets
- Extremal stabbing lines
  - VV, VEE, E4
- Adjacencies
  - catalogue

Graph in line space

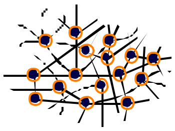
Extremal stabbing line	=	Node
Critical line set	=	Arc



Visibility skeleton



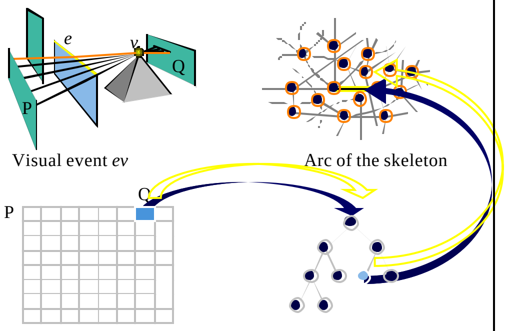
Scene



Graph in line space

- Problem: how to access the information?

Visibility skeleton



Visual event *ev*

Arc of the skeleton

Array indexed by the polygons

Search tree

Results

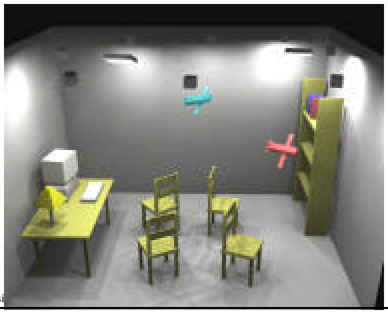
- Scenes up to 1500 polygons
  - 1.2 million of nodes
  - 32 minutes for computation
- Memory
  - $O(n^4)$ in theory, $\boldsymbol{n^2}$ observed
- Time
  - $O(n^5)$ in theory, $\boldsymbol{n^{2.4}}$ observed

Use of the skeleton

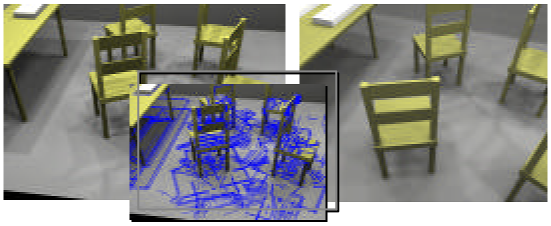
- Exact computation of form-factors
  - point-polygon
- Discontinuity meshing
  - scene subdivision along shadow boundaries
  - also for indirect lighting
- Refinement criterion
  - perceptual metric
  - error estimation

Results

- 492 polygons : 10 minutes 23 seconds



Comparison



With skeleton
10 minutes 23 seconds

[Gibson 96]
1 hour 57 minutes

Discussion

- General structure of global visibility
- Simple and local
  - on-demand construction
- Future work issues
  - robustness (partial treatment)
  - complexity: scalability
    (quadratic growth is unacceptable)

3D is much harder than 2D

- line space grows from 2D to 4D
- A line is a hyperplane in 2D, not in 3D
- Visual events are simple in 2D : lines
  They can be curved ruled surfaces in 3D
- Combinatorial explosion in 3D

Siggraph'2000 course notes on Visibility

A Multidisciplinary Survey of Visibility

Frédo DURAND

Extract of The PhD dissertation

3D Visibility: Analytical Study and Applications

Université Joseph Fourier, Grenoble, France
prepared at *i*MAGIS-GRAVIR/IMAG-INRIA,
under the supervision of Claude Puech and George Drettakis.

TABLE OF CONTENTS

CHAPTER 1

Introduction

> Il déduisit que la bibliothèque est totale, et que ses étagères consignent toutes les combinaisons possibles des vingt et quelques symboles orthographiques (nombre quoique très vaste, non infini), c'est à dire tout ce qu'il est possible d'exprimer dans toutes les langues.
>
> Jorge Luis BORGES, *La bibliothèque de Babel*

A VAST AMOUNT OF WORK has been published about visibility in many different domains. Inspiration has sometimes traveled from one community to another, but work and publications have mainly remained restricted to their specific field. The differences of terminology and interest together with the obvious difficulty of reading and remaining informed of the cumulative literature of different fields have obstructed the transmission of knowledge between communities. This is unfortunate because the different points of view adopted by different domains offer a wide range of solutions to visibility problems. Though some surveys exist about certain specific aspects of visibility, no global overview has gathered and compared the answers found in those domains. The second part of this thesis is an attempt to fill this vacuum. We hope that it will be useful to students beginning work on visibility, as well as to researchers in one field who are interested in solutions offered by other domains. We also hope that this survey will be an opportunity to consider visibility questions under a new perspective.

1 Spirit of the survey

This survey is more a "horizontal" survey than a "vertical" survey. Our purpose is not to precisely compare the methods developed in a very specific field; our aim is to give an overview which is as wide as possible.

We also want to avoid a catalogue of visibility methods developed in each domain: Synthesis and comparison are sought. However, we believe that it is important to understand the specificities of visibility problems as encountered in each field. This is why we begin this survey with an overview of the visibility questions as they arise field by field. We will then present the solutions proposed, using a classification which is not based on the field in which they have been published.

Our classification is only an analysis and organisation tool; as any classification, it does not offer infallible nor strict categories. A method can gather techniques from different categories, requiring the presentation of a single paper in several chapters. We however attempt to avoid this, but when necessary it will be indicated with cross-references.

We have chosen to develop certain techniques with more details not to remain too abstract. A section in general presents a paradigmatic method which illustrates a category. It is then followed by a shorter description of related methods, focusing on their differences with the first one.

We have chosen to mix low-level visibility acceleration schemes as well as high-level methods which make use of visibility. We have also chosen not to separate exact and approximate methods, because in many cases approximate methods are “degraded” or simplified versions of exact algorithms.

In the footnotes, we propose some thoughts or references which are slightly beyond the scope of this survey. They can be skipped without missing crucial information.

2 Flaws and bias

This survey is obviously far from complete. A strong bias towards computer graphics is clearly apparent, both in the terminology and number of references.

Computational geometry is insufficiently treated. In particular, the relations between visibility queries and range-searching would deserve a large exposition. 2D visibility graph construction is also treated very briefly.

Similarly, few complexity bounds are given in this survey. One reason is that theoretical bounds are not always relevant to the analysis of the practical behaviour of algorithms with “typical” scenes. Practical timings and memory storage would be an interesting information to complete theoretical bounds. This is however tedious and involved since different machines and scenes or objects are used, making the comparison intricate, and practical results are not always given. Nevertheless, this survey could undoubtedly be augmented with some theoretical bounds and statistics.

Terrain (or height field) visibility is nearly absent of our overview, even though it is an important topic, especially for Geographical Information Systems (*GIS*) where visibility is used for display, but also to optimize the placement of fire towers. We refer the interested reader to the survey by de Floriani *et al.* [FPM98].

The work in computer vision dedicated to the acquisition or recognition of shapes from shadows is also absent from this survey. See *e.g.* [Wal75, KB98].

The problem of aliasing is crucial in many computer graphics situations. It is a large subject by itself, and would deserve an entire survey. It is however not strictly a visibility problem, but we attempt to give some references.

Neither practical answers nor advice are directly provided. The reader who reads this survey with the question “what should I use to solve my problem” in mind will not find a direct answer. A practical guide to visibility calculation would unquestionably be a very valuable contribution. We nonetheless hope that the reader will find some hints and introductions to relevant techniques.

3 Structure

This survey is organised as follows. Chapter 2 introduces the problems in which visibility computations occur, field by field. In chapter 3 we introduce some preliminary notions which will we use to analyze and classify the methods in the following chapters. In chapter 4 we survey the classics of hidden-part removal. The following chapters present visibility methods according to the space in which the computations are performed: chapter 5 deals with object space, chapter 6 with image-space, chapter 7 with viewpoint-space and finally chapter 8 treats line-space methods. Chapter 9 presents advanced issues: managing precision and dealing with moving objects. Chapter 10 concludes with a discussion..

In appendix 12 we also give a short list of resources related to visibility which are available on the web. An index of the important terms used in this survey can be found at the end of this thesis. Finally, the references are annotated with the pages at which they are cited.

CHAPTER 2

Visibility problems

S’il n’y a pas de solution, c’est qu’il n’y a pas de problème

LES SHADOKS

VISIBILITY PROBLEMS arise in many different contexts in various fields. In this section we review the situations in which visibility computations are involved. The algorithms and data-structures which have been developed will be surveyed later to distinguish the classification of the methods from the context in which they have been developed. We review visibility in computer graphics, then computer vision, robotics and computational geometry. We conclude this chapter with a summary of the visibility queries involved.

1 Computer Graphics

For a good introduction on standard computer graphics techniques, we refer the reader to the excellent book by Foley *et al.* [FvDFH90] or the one by Rogers [Rog97]. More advanced topics are covered in [WW92].

1.1 Hidden surface removal

View computation has been the major focus of early computer graphics research. Visibility was a synonym for the determination of the parts/polygons/lines of the scene visible from a viewpoint. It is beyond the scope of this survey to review the huge number of techniques which have been developed over the years. We however review the great classics in section 4. The interested reader will find a comprehensive introduction to most of the algorithms in [FvDFH90, Rog97]. The classical survey by Sutherland *et al.* [SSS74] still provides a good classification of the techniques of the mid seventies, a more modern version being the thesis of Grant [Gra92]. More theoretical and computational geometry methods are surveyed in [Dor94, Ber93]. Some aspects are also covered in section 4.1. For the specific topic of real time display for flight simulators, see the overview by Mueller [Mue95].

The interest in hidden-part removal algorithms has been renewed by the recent domain of *non-photorealistic rendering*, that is the generation of images which do not attempt to mimic reality, such as cartoons, technical illustrations or paintings [MKT^{+}97, WS94]. Some information which are more topological are required such as the visible silhouette of the objects or its connected visible areas.

View computation will be covered in chapter 4 and section 1.4 of chapter 5.

1.2 Shadow computation

The efficient and robust computation of shadows is still one of the challenges of computer graphics. Shadows are essential for any realistic rendering of a 3D scene and provide important clues about the relative positions of objects[1]. The drawings by da Vinci in his project of a *treatise on painting* or the construction by Lambert in *Freye Perspective* give evidence of the old interest in shadow computation (Fig. 2.1). See also the book by Baxandall [Bax95] which presents very interesting insights on shadows in painting, physics and computer science.

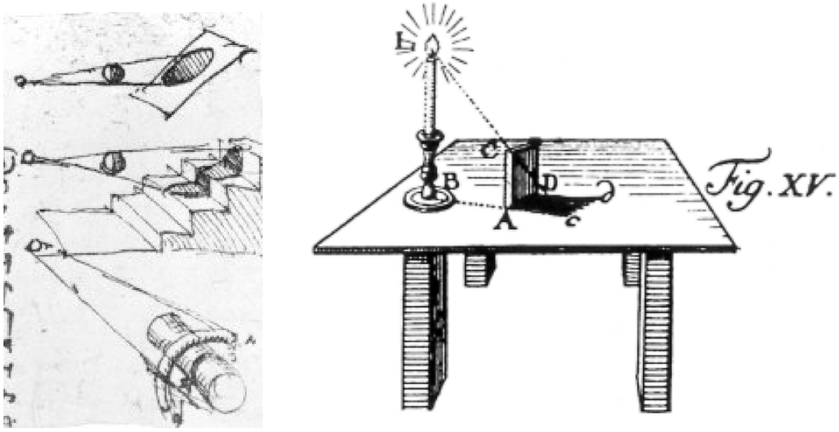


Figure 2.1: (a) Study of shadows by Leonardo da Vinci (Manuscript *Codex Urbinas*). (a) Shadow construction by Johann Heinrich Lambert (*Freye Perspective*).

Hard shadows are caused by point or directional light sources. They are easier to compute because a point of the scene is either in full light or is completely hidden from the source. The computation of hard shadows is conceptually similar to the computation of a view from the light source, followed by a reprojection. It is however both simpler and much more involved. Simpler because a point is in shadow if it is hidden from the source by any object of the scene, no matter which is the closest. Much more involved because if reprojection is actually used, it is not trivial by itself, and intricate sampling or field of view problems appear.

Soft shadows are caused by line or area light sources. A point can see all, part, or nothing of such a source, defining the regions of total lighting, penumbra and umbra. The size of the zone of penumbra varies depending on the relative distances between the source, the blocker and the receiver (see Fig. 2.2). A single view from the light is not sufficient for their computation, explaining its difficulty.

An extensive article exists [WPF90] which surveys all the standard shadows computation techniques up to 1990.

Shadow computations will be treated in chapter 5 (section 4.1, 4.2, 4.4 and 5), chapter 6 (section 2.1 , 6 and 7) and chapter 7 (section 2.3 and 2.4).

The inverse problem has received little attention: a user imposes a shadow location, and a light position is deduced. It will be treated in section 5.6 of chapter 5. This problem can be thought as the dual of sensor placement or good viewpoint computation that we will introduce in section 2.3.

1.3 Occlusion culling

The complexity of 3D scenes to display becomes larger and larger, and can not be rendered at interactive rates, even on high-end workstations. This is particularly true for applications such as CAD/CAM where the

[1] The influence of the quality of shadows on the perception of the spatial relationships is however still a controversial topic. see *e.g.* [Wan92, KKMB96]

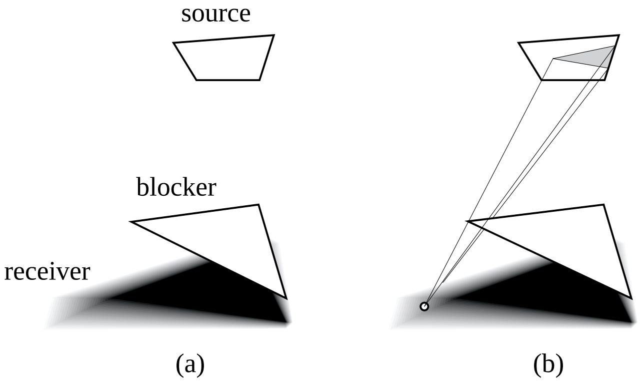


Figure 2.2: (a) Example of a soft shadow. Notice that the size of the zone of penumbra depends on the mutual distances (the penumbra is wider on the left). (b) Part of the source seen from a point in penumbra.

databases are often composed of millions of primitives, and also in driving/flight simulators, and in walkthroughs where a users want to walk through virtual buildings or even cities.

Occlusion culling (also called *visibility culling*) attempts to quickly discard the hidden geometry, by computing a superset of the visible geometry which will be sent to the graphics hardware. For example, in a city, the objects behind the nearby facades can be “obviously” rejected.

An occlusion culling algorithm has to be *conservative*. It may declare potentially visible an object which is in fact actually hidden, since a standard view computation method will be used to finally display the image (typically a z-buffer [FvDFH90]).

A distinction can be made between *online* and *offline* techniques. In an online occlusion culling method, for each frame the objects which are obviously hidden are rejected on the fly. While offline Occlusion culling precomputations consist in subdividing the scene into cells and computing for each cell the objects which may be visible from inside the cell. This set of visible object is often called the *potentially visible sets* of the cell. At display time, only the objects in the potentially visible set of the current cell are sent to the graphics hardware [2].

The landmark paper on the subject is by Clark in 1976 [Cla76] where he introduces most of the concepts for efficient rendering. The more recent paper by Heckbert and Garland [HG94] gives a good introduction to the different approaches for fast rendering. Occlusion culling techniques are treated in chapter 5 (section 4.4, 6.3 and 7), chapter 6 (section 3 and 4), chapter 7 (section 4) and chapter 8 (section 1.5).

1.4 Global Illumination

Global illumination deals with the simulation of light based on the laws of physics, and particularly with the interactions between objects. Light may be blocked by objects causing shadows. Mirrors reflect light along the symmetric direction with respect to the surface normal (Fig. 2.3(a)). Light arriving at a *diffuse* (or lambertian) object is reflected equally in all directions (Fig. 2.3(b)). More generally, a function called *BRDF* (Bidirectional Reflection Distribution Function) models the way light arriving at a surface is reflected (Fig. 2.3(c)). Fig 2.4 illustrates some bounces of light through a scene.

Kajiya has formalised global illumination with the *rendering equation* [Kaj86]. Light traveling through a point in a given direction depends on all the incident light, that is, it depends on the light coming from all the points which are visible. Its solution thus involves massive visibility computations which can be seen as the equivalent of computing a view from each point of the scene with respect to every other.

The interested reader will find a complete presentation in the books on the subject [CW93b, SP94, Gla95].

Global illumination method can also be applied to the simulation of sound propagation. See the book by Kutruff [Kut91] or [Dal96, FCE$^+$98]. See section 4.3 of chapter 5. Sound however differs from light because

[2] Occlusion-culling techniques are also used to decrease the amount of communication in multi-user virtual environments: messages and updates are sent between users only if they can see each other [Fun95, Fun96a, CT97a, MGBY99]. If the scene is too big to fit in memory, or if it is downloaded from the network, occlusion culling can be used to load into memory (or from the network) only the part of the geometry which may be visible [Fun96c, COZ98].

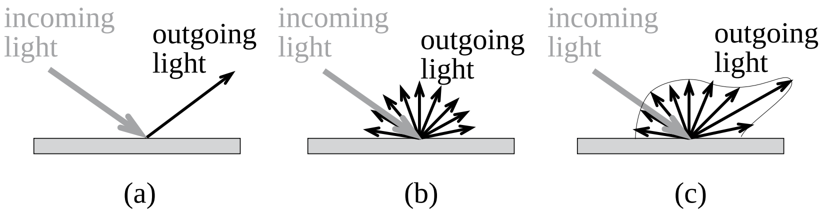


Figure 2.3: Light reflection for a given incidence angle. (a) Perfect mirror reflection. (b) Diffuse reflection. (c) General bidirectional reflectance distribution function (BRDF).

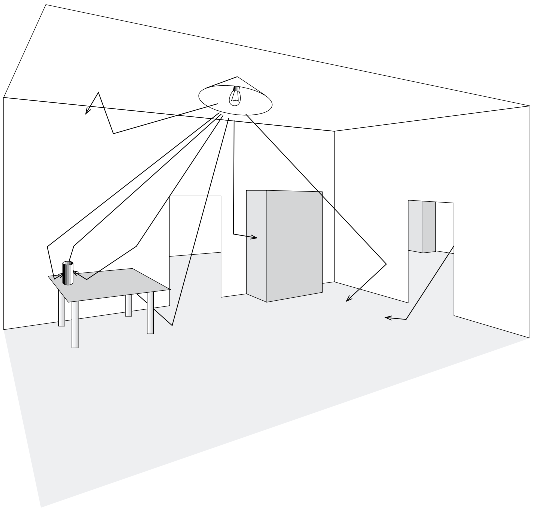


Figure 2.4: Global illumination. We show some paths of light: light emanating from light sources bounces on the surfaces of the scene (We show only one outgoing ray at each bounce, but light is generally reflected in all direction as modeled by a BRDF).

the involved wavelength are longer. Diffraction effects have to be taken into account and binary straight-line visibility is a too simplistic model. This topic will be covered in section 2.4 of chapter 6.

In the two sections below we introduce the global illumination methods based on ray-tracing and finite elements.

1.5 Ray-tracing and Monte-Carlo techniques

Whitted [Whi80] has extended the ray-casting developed by Appel [App68] and introduced recursive *ray-tracing* to compute the effect of reflecting and refracting objects as well as shadows. A ray is simulated from the viewpoint to each of the pixels of the image. It is intersected with the objects of the scene to compute the closest point. From this point, *shadow rays* can be sent to the sources to detect shadows, and reflecting or refracting rays can be sent in the appropriate direction in a recursive manner (see Fig. 2.5). A complete presentation of ray-tracing can be found on the book by Glassner [Gla89] and an electronic publication is dedicated to the subject [Hai]. A comprehensive index of related paper has been written by Speer [Spe92a]

More complete global illumination simulations have been developed based on the Monte-Carlo integration framework and the aforementioned rendering equation. They are based on a probabilistic sampling of the illumination, requiring to send even more rays. At each intersection point some rays are stochastically sent to sample the illumination, not only in the mirror and refraction directions. The process then continues recursively. It can model any BRDF and any lighting effect, but may be noisy because of the sampling.

Those techniques are called *view dependent* because the computations are done for a unique viewpoint. Veach's thesis [Vea97] presents a very good introduction to Monte-Carlo techniques.

The atomic and most costly operation in ray-tracing and Monte-Carlo techniques consists in computing the

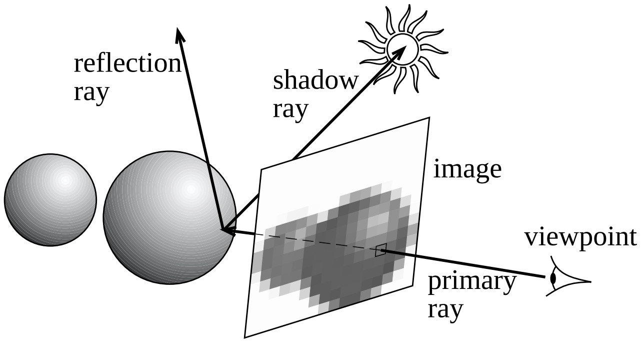


Figure 2.5: Principle of recursive ray-tracing. Primary rays are sent from the viewpoint to detect the visible object. Shadow rays are sent to the source to detect occlusion (shadow). Reflection rays can be sent in the mirror direction.

first object hit by a ray, or in the case of rays cast for shadows, to determine if the ray intersects an object. Many acceleration schemes have thus been developed over the two last decades. A very good introduction to most of these techniques has been written by Arvo and Kirk [AK89].

Ray-shooting will be treated in chapter 5 (section 1 and 4.3), chapter 6 (section 2.2), chapter 8 (section 1.4 and 3) and chapter 9 (section 2.2).

1.6 Radiosity

Radiosity methods have first been developed in the heat transfer community (see *e.g.* [Bre92]) and then adapted and extended for light simulation purposes. They assume that the objects of the scene are completely diffuse (incoming light is reflected equally in all directions of the hemisphere), which may be reasonable for architectural scene. The geometry of the scene is subdivided into patches, over which radiosity is usually assumed constant (Fig. 2.6). The light exchanges between all pairs of patches are simulated. The *form factor* between patches *A* and *B* is the proportion of light leaving *A* which reaches *B*, taking occlusions into account. The radiosity problem then resumes to a huge system of linear equations, which can be solved iteratively. Formally, radiosity is a finite element method. Since lighting is assumed directionally invariant, radiosity methods provide *view independent* solutions, and a user can interactively walk through a scene with global illumination effects. A couple of books are dedicated to radiosity methods [SP94, CW93b, Ash94].

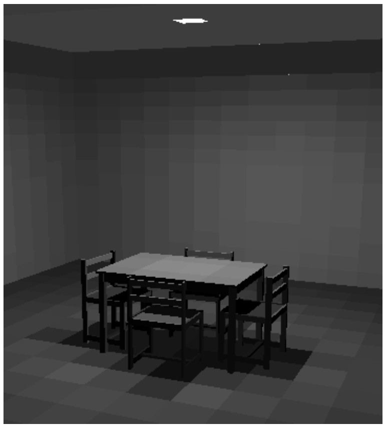


Figure 2.6: Radiosity methods simulate diffuse interreflexions. Note how the subdivision of the geometry is apparent. Smoothing is usually used to alleviate most of these artifacts.

Form factor computation is the costliest part of radiosity methods, because of the intensive visibility computations they require [HSD94]. An intricate formula has been derived by Schroeder and Hanrahan [SH93] for the form factor between two polygons in full visibility, but no analytical solution is known for the partially occluded case.

Form factor computation will be treated in chapter 4 (section 2.2), chapter 5 (section 6.1 and 7), in chapter 6 (section 2.3), chapter 7 (section 2.3), chapter 8 (section 2.1) and chapter 9 (section 2.1).

Radiosity needs a subdivision of the scene, which is usually grid-like: a quadtree is adaptively refined in the regions where lighting varies, typically the limits of shadows. To obtain a better representation, *discontinuity meshing* has been introduced. It tries to subdivides the geometry of the scene along the discontinuities of the lighting function, that is, the limits of shadows.

Discontinuity meshing methods are presented in chapter 5 (section 5.3), chapter 7 (section 2.3 and 2.4), chapter 8 (section 2.1) and chapter 9 (section 1.3, 1.5 and 2.4) [3].

1.7 Image-based modeling and rendering

3D models are hard and slow to produce, and if realism is sought the number of required primitives is so huge that the models become very costly to render. The recent domain of *image-based rendering and modeling* copes with this through the use of image complexity which replaces geometric complexity. It uses some techniques from computer vision and computer graphics. Texture-mapping can be seen as a precursor of image-based techniques, since it improves the appearance of 3D scenes by projecting some images on the objects.

View warping [CW93a] permits the reprojection of an image with depth values from a given viewpoint to a new one. Each pixel of the image is reprojected using its depth and the two camera geometries as shown in Fig. 2.7. It permits re-rendering of images at a cost which is independent of the 3D scene complexity. However, sampling questions arise, and above all, gaps appear where objects which were hidden in the original view become visible. The use of multiple base images can help solve this problem, but imposes a decision on how to combine the images, and especially to detect where visibility problems occur.

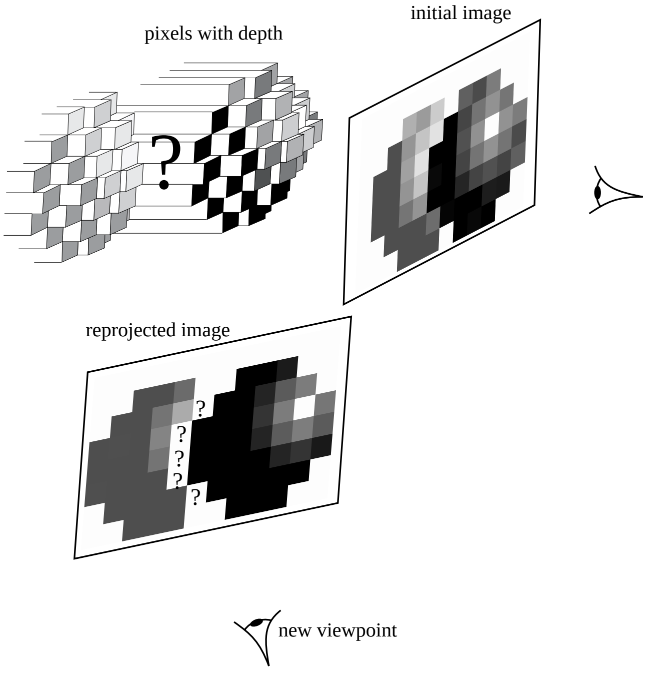


Figure 2.7: View warping. The pixels from the initial image are reprojected using the depth information. However, some gaps due to indeterminate visibility may appear (represented as “?” in the reprojected image)

Image-based modeling techniques take as input a set of photographs, and allow the scene to be seen from new viewpoints. Some authors use the photographs to help the construction of a textured 3D model [DTM96].

[3] Recent approaches have improved radiosity methods through the use of non constant bases and hierarchical representations, but the cost of form factor computation and the meshing artifact remain. Some non-diffuse radiosity computations have also been proposed at a usually very high cost. For a short discussion of the usability of radiosity, see the talk by Sillion [Sil99].

Other try to recover the depth or disparity using stereo vision [LF94, MB95]. Image warping then allows the computation of images from new viewpoints. The quality of the new images depends on the relevance of the base images. A good set of cameras should be chosen to sample the scene accurately, and especially to avoid that some parts of the scene are not acquired because of occlusion.

Some image-based rendering methods have also been proposed to speedup rendering. They do not require the whole 3D scene to be redrawn for each frame. Instead, the 2D images of some parts of the scene are cached and reused for a number of frames with simple transformation (2D rotation and translation [LS97], or texture mapping on flat [SLSD96, SS96a] or simplified [SDB97] geometry). These image-caches can be organised as *layers*, and for proper occlusion and parallax effects, these layers have to be wisely organised, which has reintroduced the problem of depth ordering.

These topics will be covered in chapter 4 (section 4.3), chapter 5 (section 4.5), chapter 6 (section 5) and chapter 8 (section 1.5).

1.8 Good viewpoint selection

In production animation, the camera is placed by skilled artists. For others applications such as games, teleconference or 3D manipulation, its position is also very important to permit a good view of the scene and the understanding of the spatial positions of the objects.

This requires the development of methods which automatically optimize the viewpoint. Visibility is one of the criteria, but one can also devise other requirements to convey a particular ambiance [PBG92, DZ95, HCS96].

The visual representation of a graph (graph drawing) in 3D raises similar issues, the number of visual alignments should be minimized. See section 1.5 of chapter 7.

We will see in section 2.3 that the placement of computer vision offers similar problems. The corresponding techniques are surveyed in chapter 5 (section 4.5 and 5.5) and chapter 7 (section 3).

2 Computer Vision

An introduction and case study of many computer vision topics can be found in the book by Faugeras [Fau93] or the survey by Guerra [Gue98]. The classic by Ballard and Brown [BB82] is more oriented towards image processing techniques for vision.

2.1 Model-based object recognition

The task of object recognition assumes a database of objects is known, and given an image, it reports if the objects are present and in which position. We are interested in model-based recognition of 3D objects, where the knowledge of the object is composed of an explicit model of its shape. It first involves low-level computer vision techniques for the extraction of features such as edges. Then these features have to be compared with corresponding features of the objects. The most convenient representations of the objects for this task represent the possible views of the object (*viewer centered* representation) rather than its 3D shape (*object-centered* representation). These views can be compared with the image more easily (2D to 2D matching as opposed to 3D to 2D matching). Fig. 2.8 illustrates a model-based recognition process.

One thus needs a data-structure which is able to efficiently represent all the possible views of an object. Occlusion has to be taken into account, and views have to be grouped according to their similarities. A class of similar views is usually called an *aspect* . A good viewer-centered representation should be able to *a priori* identify all the possible different views of an object, detecting “where” the similarity between nearby views is broken.

Psychological studies have shown evidences that the human visual system possesses such a viewer-centered representation, since objects are more easily recognised when viewed under specific viewpoints [Ull89, EB92].

A recent survey exists [Pop94] which reviews results on all the aspects of object recognition. See also the book by Jain and Flynn [JF93] and the survey by Crevier and Lepage [CL97]

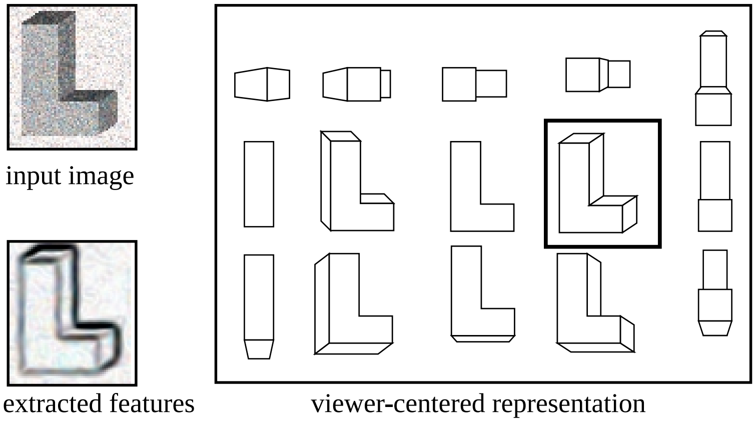


Figure 2.8: Model-based object recognition. Features are extracted from the input image and matched against the viewer-centered representation of an L-shaped object.

Object recognition has led to the development of one of the major visibility data structures, the *aspect graph*[4] which will be treated in sections 1 of chapter 7 and section 1.4 and 2.4 of chapter 9.

2.2 Object reconstruction by contour intersection

Object reconstruction takes as input a set of images to compute a 3D model. We do not treat here the reconstruction of volumetric data from slices obtained with medical equipment since it does not involve visibility.

We are interested in the reconstruction process based on contour intersection. Consider a view, from which the contour of the object has been extracted. The object is constrained to lie inside the cone defined by the viewpoint and this contour. If many images are considered, the cones can be intersected and a model of the object is estimated [SLH89]. The process is illustrated in Fig. 2.9. This method is very robust and easy to implement especially if the intersections are computed using a volumetric model by removing voxels in an octree [Pot87].

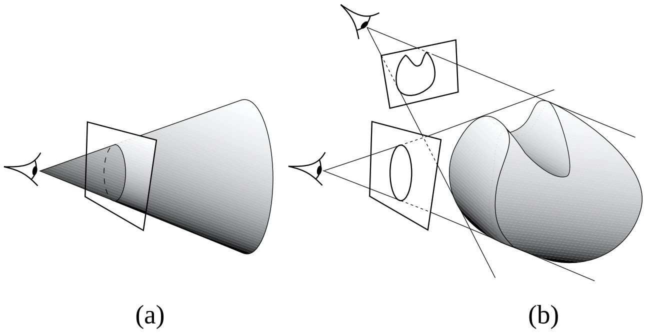


Figure 2.9: Object reconstruction by contour intersection. The contour in each view defines a general cone in which the object is constrained. A model of the object is built using the intersection of the cones. (a) Cone resulting from one image. (b) Intersection of cones from two images.

However, how close is this model to the actual object? Which class of objects can be reconstructed using this technique? If an object can be reconstructed, how many views are needed? This of course depends on self-occlusion. For example, the cavity in a bowl can never be reconstructed using this technique if the camera is constrained outside the object. The analysis of these questions imposes involved visibility considerations, as will be shown in section 3 of chapter 5.

[4] However viewer centered representation now seem superseded by the use of geometric properties which are invariant by some geometric transformation (affine or perspective). These geometric *invariants* can be used to guide the recognition of objects [MZ92, Wei93].

2.3 Sensor placement for known geometry

Computer vision tasks imply the acquisition of data using any sort of sensor. The position of the sensor can have dramatic effects on the quality and efficiency of the vision task which is then processed. *Active vision* deals with the computation of efficient placement of the sensors. It is also referred to as *viewpoint planning*.

In some cases, the geometry of the environment is known and the sensor position(s) can be preprocessed. It is particularly the case for robotics applications where the same task has to be performed on many avatars of the same object for which a CAD geometrical model is known.

The sensor(s) can be mobile, for example placed on a robot arm, it is the so called "camera in hand". One can also want to design a fixed system which will be used to inspect a lot of similar objects.

An example of sensor planning is the monitoring of a robot task like assembly. Precise absolute positioning is rarely possible, because registration can not always be performed, the controllers used drift over time and the object on which the task is performed may not be accurately modeled or may be slightly misplaced [HKL98, MI98]. Uncertainties and tolerances impose the use of sensors to monitor the robot Fig. 2.10 and 2.11 show examples of sensor controlled task. It has to be placed such that the task to be performed is visible. This principally requires the computation of the regions of space from which a particular region is not hidden. The tutorial by Hutchinson *et al.* [HH96] gives a comprehensive introduction to the visual control of robots.

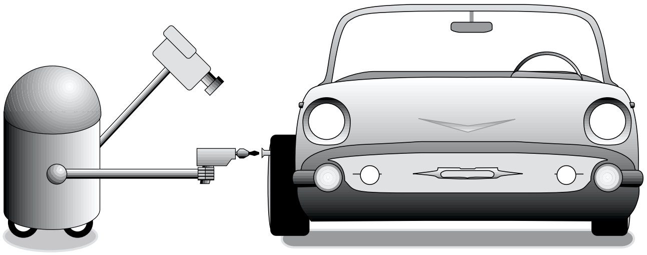


Figure 2.10: The screwdriver must be placed very precisely in front of the screw. The task is thus controlled by a camera.

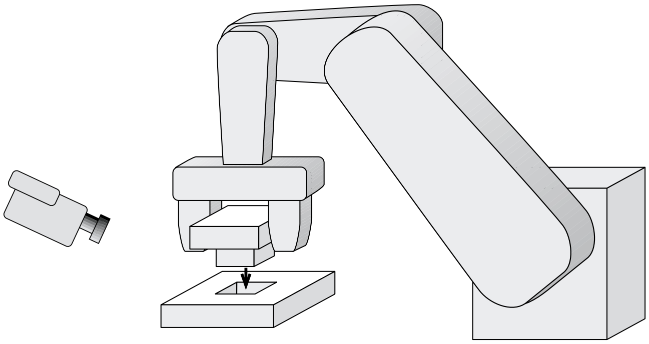


Figure 2.11: The insertion of this peg into the hole has to be performed very precisely, under the control of a sensor which has to be carefully placed.

Another example is the inspection of a manufactured part for quality verification. Measurements can for example be performed by triangulation using multiple sensors. If the geometry of the sensors is known, the position of a feature projecting on a point in the image from a given sensor is constrained on the line going through the sensor center and the point in the image. With multiple images, the 3D position of the feature is computed by intersecting the corresponding lines. Better precision is obtained for 3 views with orthogonal directions. The sensors have to be placed such that each feature to be measured is visible in at least two images. Visibility is a crucial criterion, but surface orientation and image resolution are also very important.

The illumination of the object can also be optimized. One can require that the part to be inspected be well illuminated. One can maximize the contrast to make important features easily recognisable. The optimization of viewpoint and illumination together of course leads to the best results but has a higher complexity.

See the survey by Roberts and Marshall [RM97] and by Tarabanis *et al.* [TAT95]. Section 5.5 of chapter 5 and section 3 of chapter 7 deal with the computation of good viewpoints for known environment.

2.4 Good viewpoints for object exploration

Computer vision methods have been developed to acquire a 3D model of an unknown object. The choice of the sequence of sensing operations greatly affects the quality of the results, and active vision techniques are required.

We have already reviewed the contour intersection method. We have evoked only the theoretical limits of the method, but an infinite number of views can not be used! The choice of the views to be used thus has to be carefully performed as function of the already acquired data.

Another model acquisition technique uses a laser plane and a camera. The laser illuminates the object along a plane (the laser beam is quickly rotated over time to generate a plane). A camera placed at a certain distance of the laser records the image of the object, where the illumination by the laser is visible as a slice (see Fig. 2.12). If the geometry of the plane and camera is known, triangulation can be used to infer the coordinates of the illuminated slice of the object. Translating the laser plane permits the acquisition of the whole model. The data acquired with such a system are called *range images*, that is, an image from the camera location which provides the depth of the points.

Two kinds of occlusion occur with these system: some part of an illuminated slice may not be visible to the camera, and some part of the object can be hidden to the laser, as shown in Fig. 2.12.

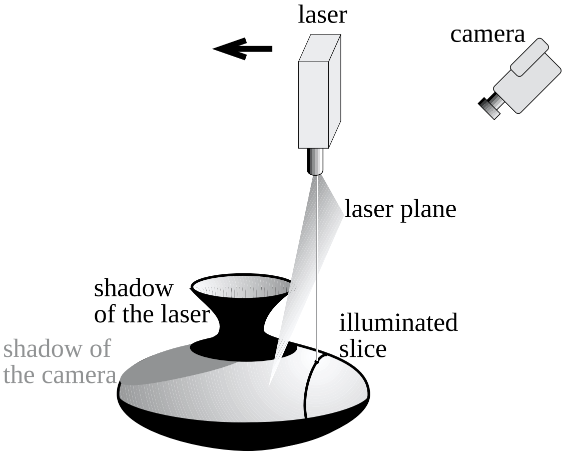


Figure 2.12: Object acquisition using a laser plane. The laser emits a plane, and the intersection between this plane and the object is acquired by a camera. The geometry of the slice can then be easily deduced. The laser and camera translate to acquire the whole object. Occlusion with respect to the laser plane (in black) and to the camera (in grey) have to be taken into account.

These problems are referred to as *best-next-view* or *purposive viewpoint adjustment*. The next viewpoint has to be computed and optimized using the data already acquired. Previously occluded parts have to be explored.

The general problems of active vision are discussed in the report written after the *1991 Active Vision Workshop* [AAA$^+$92]. An overview of the corresponding visibility techniques is given in [RM97, TAT95] and they will be discussed in section 4.5 of chapter 5.

3 Robotics

A comprehensive overview of the problems and specificities of robotics research can be found in [HKL98]. A more geometrical point of view is exposed in [HKL97]. The book by Latombe [Lat91] gives a complete and comprehensive presentation of motion planning techniques.

A lot of the robotics techniques that we will discuss treat only 2D scenes. This restriction is quite understandable because a lot of mobile robots are only allowed to move on a 2D floorplan.

As we have seen, robotics and computer vision share a lot of topics and our classification to one or the other specialty is sometimes arbitrary.

3.1 Motion planning

A robot has a certain number of degrees of freedom. A variable can be assigned to each degree of freedom, defining a (usually multidimensional) *configuration space*. For example a two joint robot has 4 degrees of freedom, 2 for each joint orientation. A circular robot allowed to move on a plane has two degrees of freedom if its orientation does not have to be taken into account. Motion planning [Lat91] consists in finding a path from a start position of the robot to a goal position, while avoiding collision with obstacles and respecting some optional additional constraints. The optimality of this path can also be required.

The case of articulated robots is particularly involved because they move in high dimensional configuration spaces. We are interested here in robots allowed to translate in 2D euclidean space, for which orientation is not considered. In this case the motion planning problem resumes to the motion planning for a point, by "growing" the obstacles using the Minkovski sum between the robot shape and the obstacles, as illustrated in Fig. 2.13.

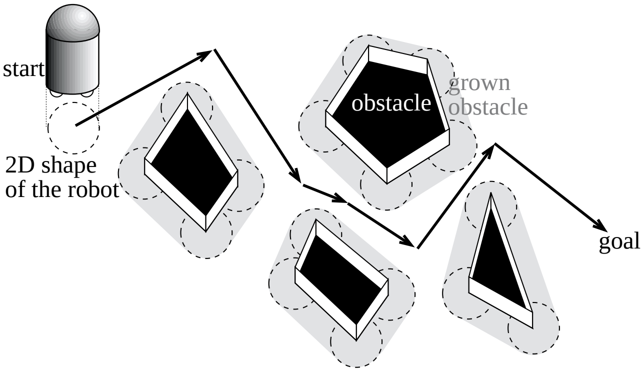


Figure 2.13: Motion planning on a floorplan. The obstacles are grown using the Minkovski sum with the shape of the robot. The motion planning of the robot in the non-grown scene resumes to that of its centerpoint in the grown scene.

The relation between euclidean motion planning and visibility comes from this simple fact: A point robot can move in straight line only to the points of the scene which are visible from it.

We will see in Section 2 of chapter 5 that one of the first global visibility data structure, the *visibility graph* was developed for motion planning purposes. [5]

3.2 Visibility based pursuit-evasion

Recently motion planning has been extended to the case where a robot searches for an intruder with arbitrary motion in a known 2D environment. A mobile robot with 360° field of view explores the scene, "cleaning" zones. A zone is cleaned when the robot sees it and can verify that no intruder is in it. It remains clean if no intruder can go there from an uncleaned region without being seen. If all the scene is cleaned, no intruder can have been missed. Fig. 2.14 shows an example of a robot strategy to clean a simple 2D polygon.

If the environment contains a "column" (that is topologically a hole), it can not be cleaned by a single robot since the intruder can always hide behind the column.

Extensions to this problem include the optimization of the path of the robot, the coordination of multiple robots, and the treatment of sensor limitations such as limited range or field of view.

[5] Assembly planning is another thematic of robotics where the ways to assemble or de-assemble an object are searched [HKL98]. The relationship between these problems and visibility would deserve exploration, especially the relation between the possibility to translate a part and the visibility of the hole in which it has to be placed.

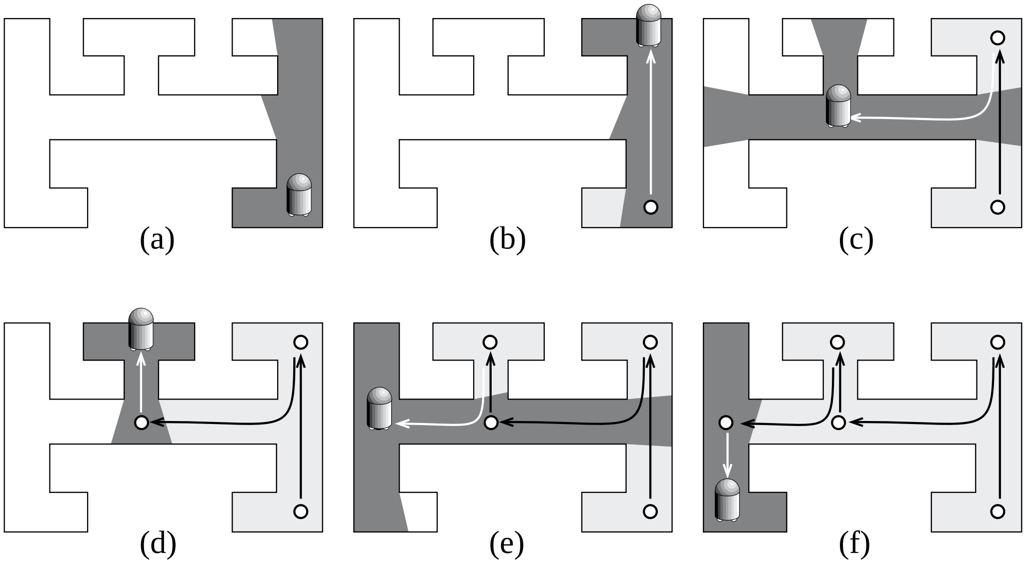


Figure 2.14: The robot has to search for an unknown intruder. The part of the scene visible from the robot is in dark grey, while the “cleaned” zone is in light grey. At no moment can an intruder go from the unknown region to the cleaned region without being seen by the robot.

Pursuit evasion is somehow related to the art-gallery problem which we will present in section 4.3. A technique to solve this pursuit-evasion problem will be treated in section 2.2 of chapter 7.

A related problem is the tracking of a mobile target while maintaining visibility. A target is moving in a known 2D environment, and its motion can have different degrees of predictability (completely known motion, bound on the velocity). A strategy is required for a mobile tracking robot such that visibility with the target is never lost. A perfect strategy can not always be designed, and one can require that the probability to lose the target be minimal. See section 3.3 of chapter 7.

3.3 Self-localisation

A mobile robot often has to be localised in its environment. The robot can therefore be equipped with sensor to help it determine its position if the environment is known. Once data have been acquired, for example in the form of a range image, the robot has to infer its position from the view of the environment as shown in Fig. 2.15. See the work by Drumheller [Dru87] for a classic method.

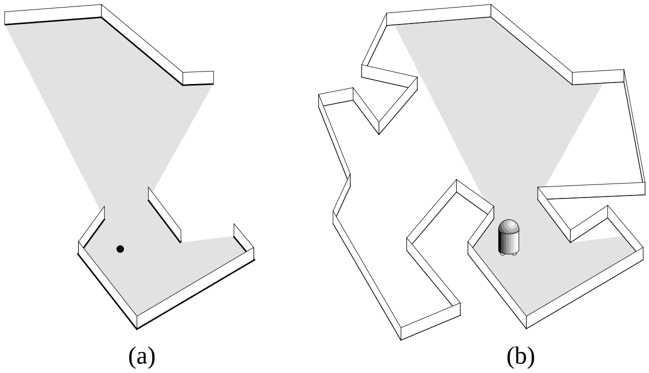


Figure 2.15: 2D Robot localisation. (a) View from the robot. (b) Deduced location of the robot.

This problem is in fact very similar to the recognition problem studied in computer vision. The robot has to “recognise” its view of the environment. We will see in section 2.1 of chapter 7 that the approaches developed are very similar.

4 Computational Geometry

The book by de Berg *et al.* [dBvKOS97] is a very comprehensive introduction to computational geometry. The one by O'Rourke [O'R94] is more oriented towards implementation. More advanced topics are treated in various books on the subject [Ede87, BY98]. Computational geometry often borrows themes from robotics.

Traditional computational geometry deals with the theoretical complexity of problems. Implementation is not necessarily sought. Indeed some of the algorithms proposed in the literature are not implementable because they are based on too intricate data-structures. Moreover, very good theoretical complexity sometimes hides a very high constant, which means that the algorithm is not efficient unless the size of the input is very large. However, recent reports [Cha96, TAA+96, LM98] and the CGAL project [FGK+96] (a robust computational geometry library) show that the community is moving towards more applied subjects and robust and efficient implementations.

4.1 Hidden surface removal

The problem of hidden surface removal has also been widely treated in computational geometry, for the case of *object-precision* methods and polygonal scenes. It has been shown that a view can have $O(n^2)$ complexity, where n is the number of edges (for example if the scene is composed of rectangles which project like a grid as shown in Fig. 2.16). Optimal $O(n^2)$ algorithms have been described [McK87], and research now focuses on *output-sensitive* algorithms, where the cost of the method also depends on the complexity of the view: a hidden surface algorithms should not spend $O(n^2)$ time if one object hides all the others.

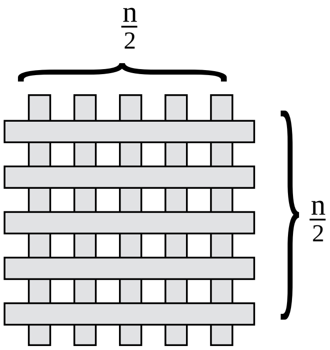


Figure 2.16: Scene composed of n rectangles which exhibits a view with complexity $O(n^2)$: the planar map describing the view has $O(n^2)$ segments because of the $O(n^2)$ visual intersections.

The question has been studied in various context: computation of a single view, preprocessing for multiple view computation, and update of a view along a predetermined path.

Constraints are often imposed on the entry. Many papers deal with axis aligned rectangles, terrains or c-oriented polygons (the number of directions of the planes of the polygons is limited).

See the thesis by de Berg [Ber93] and the survey by Dorward [Dor94] for an overview. We will survey some computational geometry hidden-part removal methods in chapter 4 (section 2.3 and 8), chapter 5 (section 1.5) and chapter 8 (section 2.2).

4.2 Ray-shooting and lines in space

The properties and algorithms related to lines in 3D space have received a lot of attention in computational geometry.

Many algorithms have been proposed to reduced the complexity of ray-shooting (that is, the determination of the first object hit by a ray). Ray-shooting is often an atomic query used in computational geometry for hidden surface removal. Some algorithms need to compute what is the object seen behind a vertex, or behind the visual intersection of two edges.

Work somehow related to motion planning concerns the *classification* of lines in space: Given a scene composed of a set of lines, do two query lines, have the same class, *i.e.* can we continuously move the first one to the other without crossing a line of the scene? This problem is related to the partition of rays or lines according to the object they see, as will be shown in section 2.2.

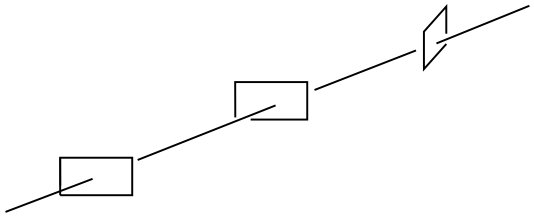


Figure 2.17: Line stabbing a set of convex polygons in 3D space

Given a set of convex objects, the *stabbing problems* searches for a line which intersects all the objects. Such a line is called a *stabbing line* or *stabber* or *transversal* (see Fig. 2.17). Stabbing is for example useful to decide if a line of sight is possible through a sequence of doors [6].

We will not survey all the results related to lines in space; we will consider only those where the data-structures and algorithms are of a particular interest for the comprehension of visibility problems. See chapter 8. The paper by Pellegrini [Pel97b] reviews the major results about lines in space and gives the corresponding references.

4.3 Art galleries

In 1973, Klee raised this simple question: how many cameras are needed to guard an art gallery? Assume the gallery is modeled by a 2D polygonal floorplan, and the camera have infinite range and 360 ° field of view. This problem is known as the *art gallery* problem. Since then, this question has received considerable attention, and many variants have been studied, as shown by the book by O'Rourke [O'R87] and the surveys on the domain [She92, Urr98]. The problem has been shown to be NP-hard.

Variation on the problem include mobile guards, limited field of view, rectilinear polygons and illumination of convex sets. The results are too numerous and most often more combinatorial than geometrical (the actual geometry of the scene is not taken into account, only its adjacencies are) so we refer the interested reader to the aforementioned references. We will just give a quick overview of the major results in section 3.1 of chapter 7.

The art gallery problem is related to many questions raised in vision and robotics as presented in section 2 and 3, and recently in computer graphics where the acquisition of models from photographs requires the choice of good viewpoints as seen in section 1.7.

4.4 2D visibility graphs

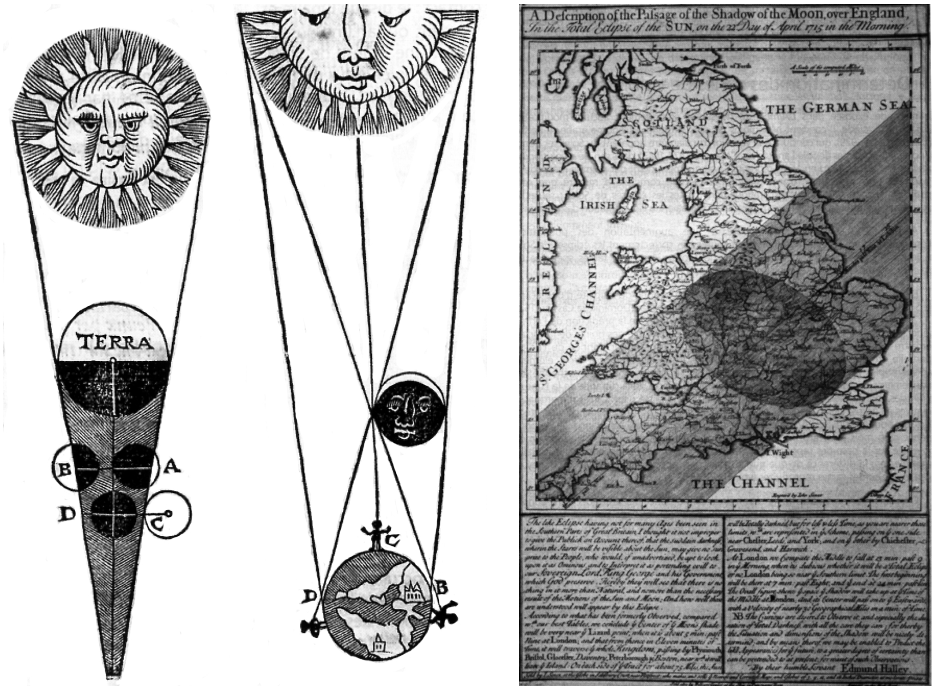
Another important visibility topic in computational geometry is the computation of *visibility graphs* which we will introduce in section 2. The characterisation of such graphs (given an abstract graph, is it the visibility graph of any scene?) is also explored, but the subject is mainly combinatorial and will not be addressed in this survey. See *e.g.* [Gho97, Eve90, OS97].

[6] Stabbing can also have an interpretation in statistics to find a linear approximation to data with imprecisions. Each data point together with its precision interval defines a box in a multidimensional space. A stabber for these boxes is a valid linear approximation.

5 Astronomy

5.1 Eclipses

Solar and lunar eclipse prediction can be considered as the first occlusion related techniques. However, the main issue was focused on planet motion prediction rather than occlusion.


TERRA
B
A
D
C
D
C
B
A Description of the Passage of the Shadow of the Moon over England, In the Total Eclipse of the SUN, on the 22d Day of April 1715 in the Morning
THE GERMAN SEA
SCOTLAND
THE IRISH SEA
S. GEORGES CHANNEL
THE CHANNEL
FRANCE
Edmund Halley

(a) (b)

Figure 2.18: Eclipses. (a) Lunar and Solar eclipse by Purbach. (b) Prediction of the 1715 eclipse by Halley.

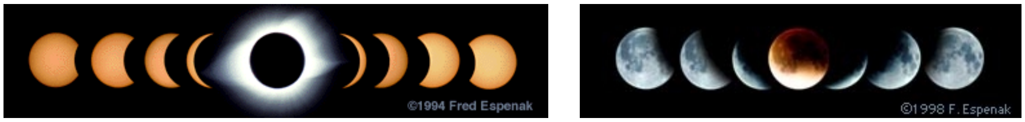


Figure 2.19: 1994 solar eclipse and 1993 lunar eclipse. Photograph Copyright 1998 by Fred Espenak (NASA/Goddard Space Flight Center).

See *e.g.*
http://sunearth.gsfc.nasa.gov/eclipse/eclipse.html
http://www.bdl.fr/Eclipse99

5.2 Sundials

Sundials are another example of shadow related techniques.

see *e.g.*
http://www.astro.indiana.edu/personnel/rberring/sundial.html
http://www.sundials.co.uk/2sundial.htm

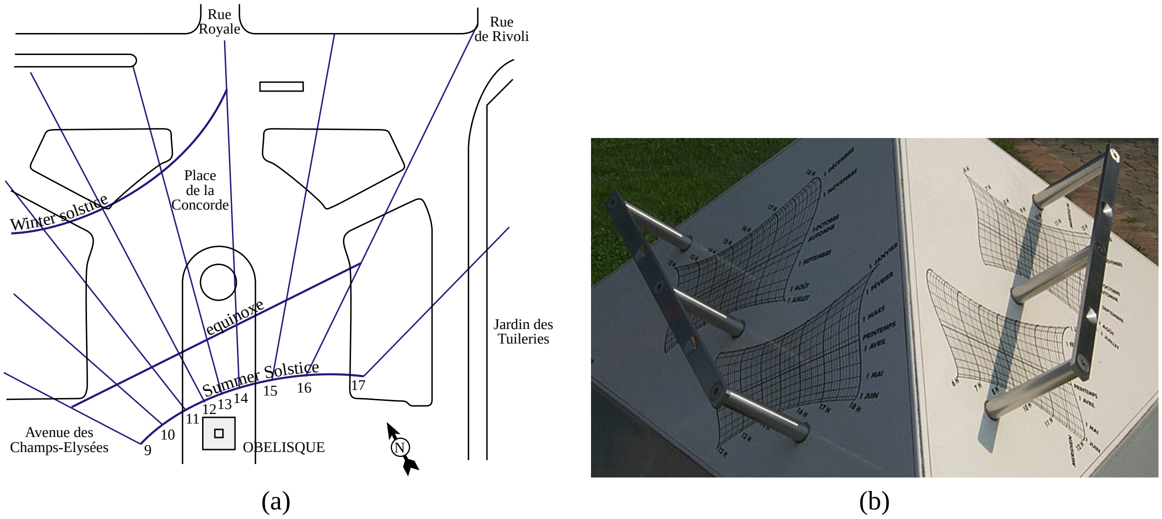


Figure 2.20: (a) Project of a sundial on the Place de la Concorde in Paris. (b) Complete sundial with analemmas in front of the CICG in Grenoble.

6 Summary

Following Grant [Gra92], visibility problems can be classified according to their increasing dimensionality: The most atomic query is ray-shooting. View and hard shadow computation are two dimensional problems. Occlusion culling with respect to a point belong to the same category which we can refer to as classical visibility problems. Then comes what we call *global* visibility issues[7]. These include visibility with respect to extended regions such as extended light sources or volumes, or the computation of the region of space from which a feature is visible. The mutual visibility of objects (required for example for global illumination simulation) is a four dimensional problem defined on the pairs of points on surfaces of the scene. Finally the enumeration of all possible views of an object or the optimization of a viewpoint impose the treatment of two dimensional view computation problems for all possible viewpoints.

[7] Some author also define occlusion by other objects as global visibility effects as opposed to backface culling and silhouette computation.

CHAPTER 3

Preliminaries

> On apprend à reconnaître les forces sous-jacentes ; on apprends la préhistoire du visible. On apprend à fouiller les profondeurs, on apprend à mettre à nu. On apprend à démontrer, on apprend à analyser
>
> Paul KLEE, *Théorie de l'art moderne*

BEFORE presenting visibility techniques, we introduce a few notions which will be useful for the understanding and comparison of the methods we survey. We first introduce the different spaces which are related to visibility and which induce the classification that we will use. We then introduce the notion of *visual event*, which describes "where" visibility changes in a scene and which is central to many methods. Finally we discuss some of the differences which explain why 3D visibility is much more involved than its 2D counterpart.

1 Spaces and algorithm classification

In their early survey Sutherland, Sproull and Schumacker [SSS74] classified hidden-part removal algorithms into *object space* and *image-space* methods. Our terminology is however slightly different from theirs, since they designated the *precision* at which the computations are performed (at the resolution of the image or exact), while we have chosen to classify the methods we survey according to the space in which the computations are performed.

Furthermore we introduce two new spaces: the space of all viewpoints and the space of lines. We will give a few simple examples to illustrate what we mean by all these spaces.

1.1 Image-space

In what follow, we have classified as *image-space* all the methods which perform their operations in 2D projection planes (or other manifolds). As opposed to Sutherland *et al.*'s classification [SSS74], this plane is not restricted to the plane of the actual image. It can be an intermediate plane. Consider the example of hard shadow computation: an intermediate image from the point light source can be computed.

Of course if the scene is two dimensional, image space has only one dimension: the angle around the viewpoint.

Image-space methods often deal with a discrete or *rasterized* version of this plane, sometimes with a depth information for each point. Image-space methods will be treated in chapter 6.

1.2 Object-space

In contrast, object space is the 3 or 2 dimensional space in which the scene is defined. For example, some hard shadow computation methods use *shadow volumes* [FvDFH90, WPF90]. These volumes are truncated frusta defined by the point light source and the occluding objects. A portion of space is in shadow if it lies inside a shadow volume. Object-space methods will be treated in chapter 5.

1.3 Viewpoint-space

We define the *viewpoint space* as the set of all possible viewpoints. This space depends on the projection used. If perspective projection is used, the viewpoint space is equivalent to the object space. However, if orthographic (also called parallel) projection is considered, then a view is defined by a direction, and the viewpoint space is the set S^2 of directions, often called *viewing sphere* as illustrated in Fig. 3.1. Its projection on a cube is sometimes used for simpler computations.

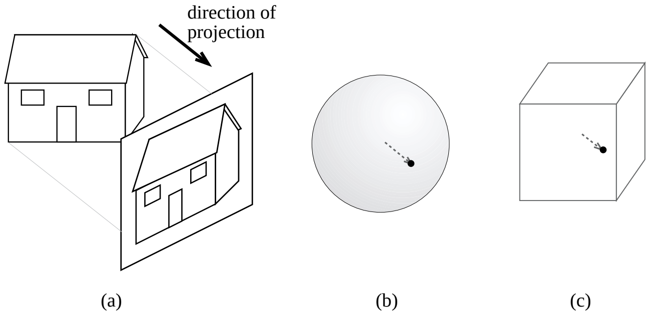


Figure 3.1: (a) Orthographic view. (b) Corresponding point on the viewing sphere and (c) on the viewing cube.

An example of viewpoint space method would be to discretize the viewpoint space and precompute a view for each sample viewpoint. One could then render views very quickly with a simple look-up scheme. The viewer-centered representation which we have introduced in section 2.1 of the previous chapter is typically a viewpoint space approach since each possible view should be represented.

Viewpoint-space can be limited. For example, the viewer can be constrained to lie at eye level, defining a 2D viewpoint space (the plane $z = h_{eye}$) in 3D for perspective projection. Similarly, the distance to a point can be fixed, inducing a spherical viewpoint-space for perspective projection.

It is important to note that even if perspective projection is used, there is a strong difference between viewpoint space methods and object-space methods. In a viewpoint space, the properties of points are defined by their view. An orthographic viewpoint-space could be substituted in the method.

Shadow computation methods are hard to classify: the problem can be seen as the intersection of scene objects with shadow volume, but it can also be seen as the classification of viewpoint lying on the objects according to their view of the source. Some of our choices can be perceived arbitrary.

In 2D, viewpoint-space has 2 dimensions for perspective projection and has 1 dimension if orthographic projection is considered.

Viewpoint space methods will be treated in chapter 7.

1.4 Line-space

Visibility can intuitively be defined in terms of lines: two point A and B are mutually visible if no object intersects line (AB) between them. It is thus natural to describe visibility problems in line space.

For example, one can precompute the list of objects which intersect each line of a discretization of line-space to speed-up ray-casting queries.

In 2D, lines have 2 dimensions: for example its direction θ and distance to the origin ρ. In 3D however, lines have 4 dimensions. They can for example be parameterized by their direction (θ, φ) and by the intersection (u, v) on an orthogonal plane (Fig. 3.2(a)). They can also be parameterized by their intersection with two planes (Fig. 3.2(b)). These two parameterizations have some singularities (at the pole for the first one, and for lines parallel to the two planes in the second). Lines in 3D space can not be parameterized without a singularity. In section 3 of chapter 8 we will study a way to cope with this, embedding lines in a 5 dimensional space.

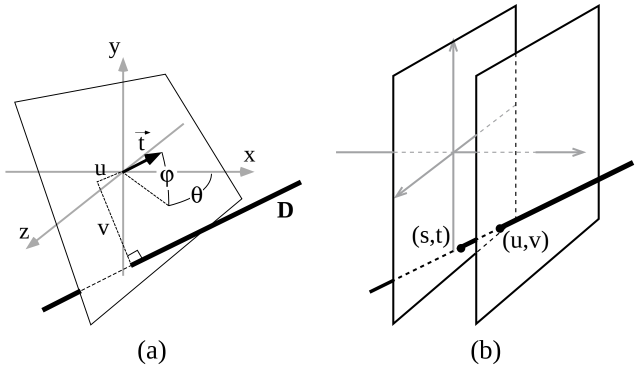


Figure 3.2: Line parameterisation. (a) Using two angles and the intersection on an orthogonal plane. (b) Using the intersection with two planes.

The set of lines going through a point describe the view from this point, as in the ray-tracing technique (see Fig. 2.5). In 2D the set of lines going through a point has one dimension: for example their angle. In 3D, 2 parameters are necessary to describe a line going through a point, for example two angles.

Many visibility queries are expressed in terms of rays and not lines. The ray-shooting query computes the first object seen from a point in a given direction. Mathematically, a ray is a half line. Ray-space has 5 dimensions (3 for the origin and two for the direction).

The mutual visibility query can be better expressed in terms of segments. A and B are mutually visible only if segment $[AB]$ intersects no object. Segment space has 6 dimensions: 3 for each endpoint.

The information expressed in terms of rays or segments is very redundant: many colinear rays “see” the same object, many colinear segments are intersected by the same object. We will see that the notion of *maximal free segments* handles this. Maximal free segments are segments of maximal length which do not touch the objects of the scene in their interior. Intuitively these are segments which touch objects only at their extremities.

We have decided to group the methods which deal with these spaces in chapter 8. The interested reader will find some important notions about line space reviewed in appendix 11.

1.5 Discussion

Some of the methods we survey do not perform all their computations in a single space. An intermediate data-structure can be used, and then projected in the space in which the final result is required.

Even though each method is easier to describe in a given space, it can often be described in a different space. Expressing a problem or a method in different spaces is particularly interesting because it allows different insights and can yield alternative methods. We particularly invite the reader to transpose visibility questions to line space or ray space. We will show throughout this survey that visibility has a very natural interpretation in line space.

However this is not an incitation to actually perform complex calculations in 4D line space. We just suggest a different way to understand problems and develop methods, even if calculations are eventually performed in image or object space.

2 Visual events, singularities

We now introduce a notion which is central to most of the algorithms, and which expresses “how” and “where” visibility changes. We then present the mathematical framework which formalizes this notion, the theory of singularities. The reader may be surprised by the space devoted in this survey to singularity theory compared to its use in the literature. We however believe that singularity theory permits a better insight on visibility problems, and allows one to generalize some results on polygonal scenes to smooth objects.

2.1 Visual events

Consider the example represented in Fig. 3.3. A polygonal scene is represented, and the views from three eyepoints are shown on the right. As the eyepoint moves downwards, pyramid P becomes completely hidden by polygon Q. The limit eyepoint is eyepoint 2, for which vertex V projects exactly on edge E. There is a topological change in visibility: it is called a *visual event* or a *visibility event*.

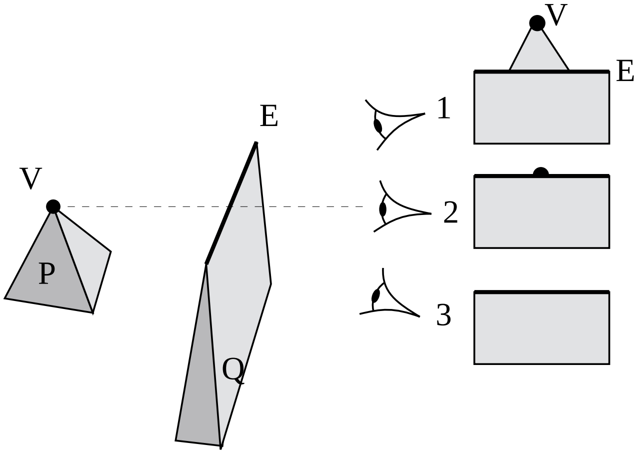


Figure 3.3: EV visual event. The views from the three eyepoints are represented on the right. As the eyepoint moves downwards, vertex V becomes hidden. Viewpoint 2 is the limit eyepoint, it lies on a *visual event*.

Visual events are fundamental to understand many visibility problems and techniques. For example when an observer moves through a scene, objects appear and disappear at such events (Fig. 3.3). If pyramid P emits light, then eyepoint 1 is in penumbra while eyepoint 3 is in umbra: the visual event is a shadow boundary. If a viewpoint is sought from which pyramid P is visible, then the visual event is a limit of the possible solutions.

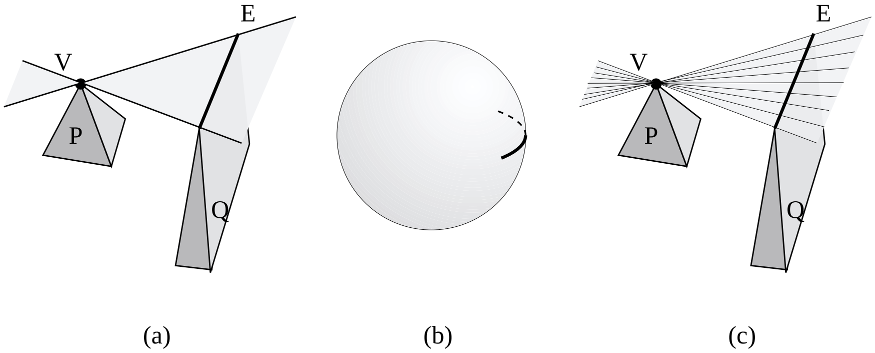


Figure 3.4: Locus an EV visual event. (a) In object space or perspective viewpoint space it is a wedge. (b) In orthographic viewpoint space it is an arc of a great circle. (c) In line space it is the 1D set of lines going through V and E

Fig. 3.4 shows the locus of this visual event in the spaces we have presented in the previous section. In object space or in perspective viewpoint space, it is the wedge defined by vertex V and edge E. We say that V and E are the *generators* of the event. In orthographic viewpoint space it is an arc of a great circle of the viewing sphere. Finally, in line-space it is the set of lines going through V and E. These *critical lines* have one degree of freedom since they can be parameterized by their intercept on E, we say that it is a 1D set of lines.

The EV events generated by a vertex V are caused by the edges which are visible from V. The set of events generated by V thus describe the view from V. Reciprocally, a line drawing of a view from an arbitrary point P can be seen as the set of EV events which would be generated if an imaginary vertex was place at P.

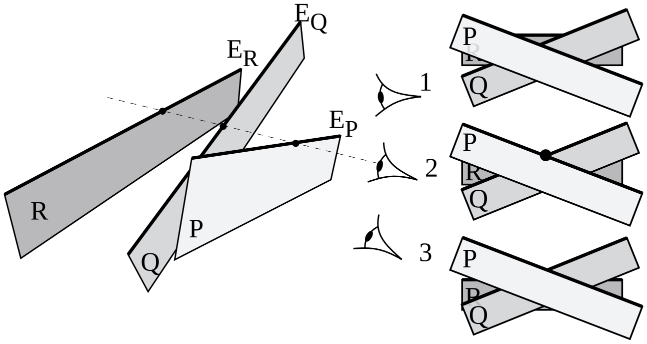
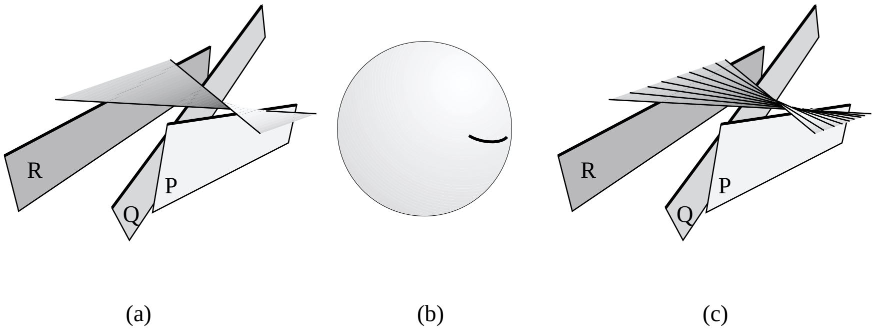


Figure 3.5: A EEE visual event. The views from the three eyepoints are represented on the right. As the eyepoint moves downwards, polygon R becomes hidden by the conjunction of polygon P and Q. From the limit viewpoint 2, the three edges have a visual intersection.

There is also a slightly more complex kind of visual event in polygonal scenes. It involves the interaction of 3 edges which project on the same point (Fig. 3.5). When the eyepoint moves downwards, polygon P becomes hidden by the conjunction of Q and R. From the limit eyepoint 2, edges E_P, E_Q and E_R are aligned.



(a) (b) (c)

Figure 3.6: Locus of a EEE visual event. (a) In object-space or perspective viewpoint space it is a ruled quadrics. (b) In orthographic viewpoint space it is a quadric on the viewing sphere. (c) In line space it is the set of lines stabbing the three edges.

The locus of such events in line space is the set of lines going through the three edges (we also say that they *stab* the three edges) as shown on Fig. 3.6(c). In object space or perspective viewpoint space, this defines a ruled quadric often called *swath* (Fig. 3.6(a)). (It is in fact doubly ruled: the three edges define one family of lines, the stabber defining the second.) In orthographic viewpoint space it is a quadric on the viewing sphere (see Fig. 3.6(b)).

Finally, a simpler class of visual events are caused by a viewpoint lying in the plane of faces of the scene. The face becomes visible or hidden at such an event.

Visual events are simpler in 2D: they are simply the *bitangents* and *inflexion points*of the scene.

A deeper understanding of visual events and their generalisation to smooth objects requires a strong formalism: it is provided by the singularity theory.

2.2 Singularity theory

The singularity theory studies the emergence of discrete structures from smooth continuous ones. The branch we are interested in has been developed mainly by Whitney [Whi55], Thom [Tho56, Tho72] and Arnold [Arn69]. It permits the study of sudden events (called *catastrophes*) in systems governed by smooth continuous laws. An introduction to singularity theory for visibility can be found in the masters thesis by PetitJean [Pet92] and an educational comics has been written by Ian Stewart [Ste82]. See also the book by Koenderink [Koe90] or his papers with van Doorn [Kv76, KvD82, Kø84, Koe87].

We are interested in the singularities of smooth mappings. For example a view projection is a smooth mapping which associate each point of 3D space to a point on a projection plane. First of all, singularity theory permits the description the structure of the visible parts of a smooth object.

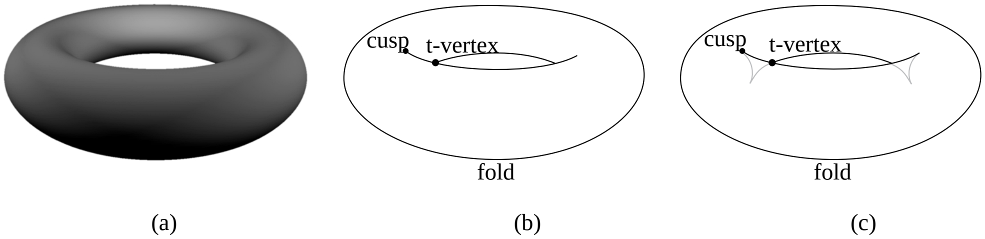


Figure 3.7: View of a torus. (a) Shaded view. (b) Line drawing with singularities indicated (b) Opaque and transparent contour.

Consider the example of a smooth 3D object such as the torus represented in Fig. 3.7(a). Its projection on a viewing plane is continuous nearly everywhere. However, some abrupt changes appear at the so called silhouette. Consider the number of point of the surface of the object projecting on a given point on the projection plane (counting the backfacing points). On the exterior of the silhouette no point is projected. In the interior two points (or more) project on the same point. These two regions are separated by the silhouette of the object at which the number of projected point changes abruptly.

This abrupt change in the smooth mapping is called a *singularity* or *catastrophe* or *bifurcation*. The singularity corresponding to the silhouette was named *fold* (or also *occluding contour* or *limb*). The fold is usually used to make a line drawing of the object as in Fig. 3.7(b). It corresponds to the set of points which are tangent to the viewing direction[1].

The fold is the only stable curve singularity for generic surfaces: if we move the viewpoint, there will always be a similar fold.

The projection in Fig. 3.7 also exhibits two point singularities: a *t-vertex* and a *cusp*. T-vertices results from the intersection of two folds. Fig. 3.7(c) shows that a fourth fold branch is hidden behind the surface. Cusps represent the visual end of folds. In fact, a cusp corresponds to a point where the fold has an inflexion in 3D space. A second tangent fold is hidden behind the surface as illustrated in Fig. 3.7(c).

These are the only three stable singularities: all other singularities disappear after a small perturbation of the viewpoint (if the object is generic, which is not the case of polyhedral objects). These stable singularities describe the limits of the visible parts of the object. Malik [Mal87] has established a catalogue of the features of line drawings of curved objects.

Singularity theory also permits the description of how the line drawing changes as the viewpoint is moved. Consider the example represented in Fig. 3.8. As the viewpoint moves downwards, the back sphere becomes hidden by the front one. From viewpoint (b) where this visual event occurs, the folds of the two spheres are superimposed and tangent. This unstable singularity is called a *tangent crossing*. It is very similar to the *EV* visual event shown in Fig. 3.3. It is unstable in the sense that any small change in the viewpoint will make it disappear. The viewpoint is not *generic*, it is *accidental*.

[1] What is the relationship between the view of a torus and the occurrence of a sudden catastrophe? Imagine the projection plane is the command space of a physical system with two parameters x and y. The torus is the response surface: for a pair of parameters (x,y) the depth z represents the state of the system. Note that for a pair of parameters, there may be many possible states, depending on the history of the system. When the command parameters vary smoothly, the corresponding state varies smoothly on the surface of the torus. However, when a fold is met, there is an abrupt change in the state of the system, this is a *catastrophe*. See *e.g.* [Ste82].

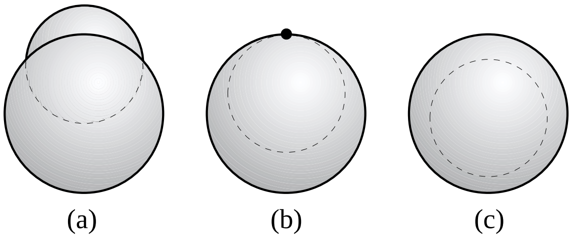


Figure 3.8: Tangent crossing singularity. As the viewpoint moves downwards, the back sphere becomes hidden by the frontmost one. At viewpoint (b) a singularity occurs (highlighted with a point): the two spheres are visually tangent.

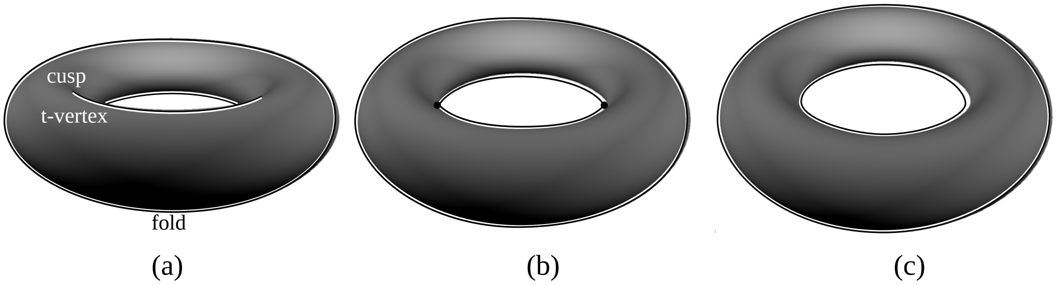


Figure 3.9: Disappearance of a cusp at a swallowtail singularity at viewpoint (b). (in fact two swallowtails occur because of the symmetry of the torus)

Another unstable singularity is shown in Fig. 3.9. As the viewpoints moves upward, the t-vertex and the cusp disappear. In Fig. 3.9(a) the points of the plane below the cusp result from the projection of 4 points of the torus, while in Fig. 3.9(c) all points result from the projection of 2 or 0 points. This unstable singularity is called *swallowtail*.

Unstable singularities are the events at which the organisation of the view of a smooth object (or scene) is changed. These singularities are related to the differential properties of the surface. For example swallowtails occur only in hyperbolic regions of the surface, that is, regions where the surface is locally nor concave nor convex.

Singularity theory originally does not consider opaqueness. Objects are assumed transparent. As we have seen, at cusps and t-vertices, some fold branches are hidden. Moreover a singularity like a tangent crossing is considered even if some objects lie between the two sphere causing occlusion. The visible singularity are only a subset but all the changes observed in views of opaque objects can be described by singularity theory. Some catalogues now exist which describe singularities of opaque objects [2]. See Fig. 3.10.

The catalogue of singularities for views of smooth objects has been proposed by Kergosien [Ker81] and Rieger [Rie87, Rie90] who has also proposed a classification for piecewise smooth objects [Rie87] [3].

3 2D *versus* 3D Visibility

We enumerate here some points which make that the difference between 2D and 3D visibility can not be summarized by a simple increment of one to the dimension of the problem.

This can be more easily envisioned in line space. Recall that the atomic queries in visibility are expressed in line-space (first point seen along a ray, are two points mutually visible?).

[2] Williams [WH96, Wil96] tries to fill in the gap between opaque and transparent singularities. Given the view of an object, he proposes to deduce the invisible singularities from the visible ones. For example at a t-vertex, two folds intersect but only three branches are visible; the fourth one which is occluded can be deduced. See Fig. 3.10.

[3] Those interested in the problems of robustness and degeneracies for geometric computations may also notice that a degenerate configuration can be seen as a singularity of the space of scenes. The exploration of the relations between singularities and degeneracies could help formalize and systemize the treatment of the latter. See also section 2 of chapter 9.

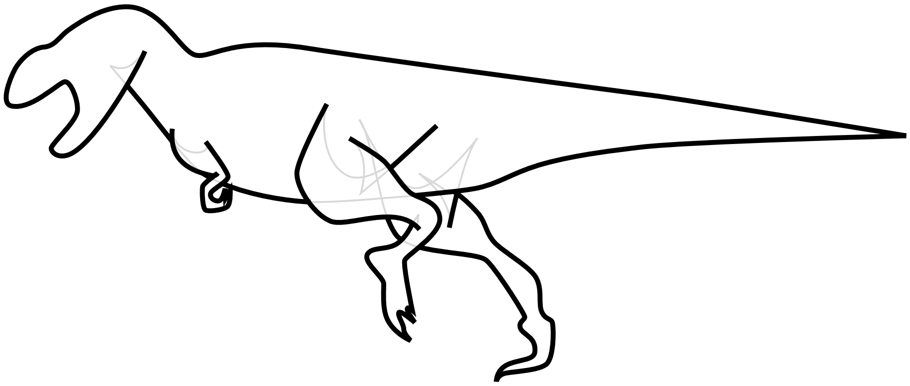


Figure 3.10: Opaque (bold lines) and semi-transparent (grey) singularities. After [Wil96].

First of all, the increase in dimension of line-space is two, not one (in 2D line-space is 2D, while in 3D it is 4D). This makes things much more intricate and hard to apprehend.

A line is a hyperplane in 2D, which is no more the case in 3D. Thus the separability property is lost: a 3D line does not separate two half-space as in 2D.

A 4D parameterization of 3D lines is not possible without singularities (the one presented in Fig. 3.2(a) has two singularities at the pole, while the one in Fig. 3.2(b) can not represent lines parallel to the two planes). See section 3 of chapter 8 for a partial solution to this problem.

Visual events are simple in 2D: bitangent lines or tangent to inflection points. In 3D their locus are surfaces which are rarely planar (EEE or visual events for curved objects).

All these arguments make the sentence “the generalization to 3D is straightforward” a doubtful statement in any visibility paper.

CHAPTER 4

The classics of hidden part removal

> Il convient encore de noter que c'est parce que quelque chose des objets extérieurs pénétre en nous que nous voyons les formes et que nous pensons
>
> ÉPICURE, *Doctrines et Maximes*

We first briefly review the classical algorithms to solve the hidden surface removal problem. It is important to have these techniques in mind for a wider insight of visibility techniques. We will however remain brief, since it is beyond the scope of this survey to discuss all the technical details and variations of these algorithms. For a longer survey see [SSS74, Gra92], and for a longer and more educational introduction see [FvDFH90, Rog97].

The view computation problem is often reduced to the case where the viewpoint lies on the *z* axis at infinity, and *x* and *y* are the coordinates of the image plane; *y* is the vertical axis of the image. This can be done using a perspective transform matrix (see [FvDFH90, Rog97]). The objects closer to the viewpoint can thus be said to lie "above" (because of the *z* axis) as well as "in front" of the others. Most of the methods treat polygonal scenes.

Two categories of approaches have been distinguished by Sutherland *et al*. *Image-precision* algorithms solve the problem for a discrete (rasterized) image, visibility being sampled only at pixels; while *object-precision* algorithm solve the exact problem. The output of the latter category is often a *visibility map*, which is the planar map describing the view. The order in which we present the methods is not chronological and has been chosen for easier comparison.

Solutions to hidden surface removal have other applications that the strict determination of the objects visible from the viewpoint. As evoked earlier, hard shadows can be computed using a view from a point light source. Inversely, the amount of light arriving at a point in penumbra corresponds to the visible part of the source from this point as shown in Fig. 2.2(b). Interest for the application of exact view computation has thus recently been revived.

1 Hidden-Line Removal

The first visibility techniques have were developed for *hidden line removal* in the sixties. These algorithms provide information only on the visibility of edges. Nothing is known on the interior of visible faces, preventing shading of the objects.

1.1 Robert

Robert [Rob63] developed the first solution to the hidden line problem. He tests all the edges of the scene polygons for occlusion. He then computes the intersection of the wedge defined by the viewpoint and the edge and all objects in the scene using a parametric approach.

1.2 Appel

Appel [App67] has developed the notion of *quantitative invisibility* which is the number of objects which occlude a given point. This is the notion which we used to present singularity theory: the number of points of the object which project on a given point in the image. Visible points are those with 0 quantitative invisibility. The quantitative invisibility of an edge of a view changes only when it crosses the projection of another edge (it corresponds to a *t-vertex*). Appel thus computes the quantitative invisibility number of a vertex, and updates the quantitative invisibility at each visual edge-edge intersection.

Markosian *et al.* [MKT$^+$97] have used this algorithm to render the silhouette of objects in a non-photorealistic manner. When the viewpoint is moved, they use a probabilistic approach to detect new silhouettes which could appear because an unstable singularity is crossed.

1.3 Curved objects

Curved objects are harder to handle because their silhouette (or *fold*) first has to be computed (see section 2.2 of chapter 3). Elber and Cohen [EC90] compute the silhouette using adaptive subdivision of parametric surfaces. The surface is recursively subdivided as long as it may contain parts of the silhouette. An algorithm similar to Appel's method is then used. Snyder [Sny92] proposes the use of interval arithmetic for robust silhouette computation.

2 Exact area-subdivision

2.1 Weiler-Atherton

Weiler and Atherton [WA77] developed the first object-precision method to compute a visibility map. Objects are preferably sorted according to their depth (but cycles do not have to be handled). The frontmost polygons are then used to clip the polygons behind them.

This method can also be very simply used for hard shadow generation, as shown by Atherton *et al.* [AWG78]. A view is computed from the point light source, and the clipped polygons are added to the scene database as lit polygon parts.

The problem with Weiler and Atherton's method, as for most of the object-precision methods, is that it requires robust geometric calculations. It is thus prone to numerical precision and degeneracy problems.

2.2 Application to form factors

Nishita and Nakamae [NN85] and Baum *et al.* [BRW89] compute an accurate form factor between a polygon and a point (the portion of light leaving the polygon which arrives at the point) using Weiler and Atherton's clipping. Once the source polygon is clipped, an analytical formula can be used. Using Stoke's theorem, the integral over the polygon is computed by an integration over the contour of the visible part. The jacobian of the lighting function can be computed in a similar manner [Arv94].

Vedel [Ved93] has proposed an approximation for the case of curved objects.

2.3 Mulmuley

Mulmuley [Mul89] has proposed an improvement of exact area-subdivision methods. He inserts polygons in a *randomized* order (as in quick-sort) and maintains the visibility map. Since visibility maps can have complex boundaries (concave, with holes), he uses a trapezoidal decomposition [dBvKOS97]. Each trapezoid corresponds to a part of one (possibly temporary) visible face.

Each trapezoid of the map maintains a list of *conflict* polygons, that is, polygons which have not yet been projected and which are above the face of the trapezoid. As a face is chosen for projection, all trapezoids with which it is in conflict are updated. If a face is below the temporary visible scene, no computation has to be performed.

The complexity of this algorithm is very good, since the probability of a feature (vertex, part of edge) to induce computation is inversely proportional to its quantitative invisibility (the number of objects above it). It should be easy to implement and robust due to its randomized nature. However, no implementation has been reported to our knowledge.

2.4 Curved objects

Krishnan and Manocha [KM94] propose an adaptation of Weiler and Atherton's method for curved objects modeled with NURBS surfaces. They perform their computation in the parameter space of the surface. The silhouette corresponds to the points where the normal is orthogonal to the view-line, which defines a polynomial system. They use an algebraic marching method to solve it. These silhouettes are approximated by piecewise-linear curves and then projected on the parts of the surface below, which gives a partition of the surface where the quantitative invisibility is constant.

3 Adaptive subdivision

The method developed by Warnock [War69] can be seen as an approximation of Weiler and Atherton's exact method, even though it was developed earlier. It recursively subdivides the image until each region (called a *window*) is declared homogeneous. A window is declared homogeneous if one face completely covers it and is in front of all other faces. Faces are classified against a window as intersecting or disjoint or surrounding (covering). This classification is passed to the subwindows during the recursion. The recursion is also stopped when pixel-size is reached.

The classical method considers quadtree subdivision. Variations however exist which use the vertices of the scene to guide the subdivision and which stop the recursion when only one edge covers the window.

Marks *et al.* [MWCF90] presents an analysis of the cost of adaptive subdivision and proposes a heuristic to switch between adaptive methods and brute-force z-buffer.

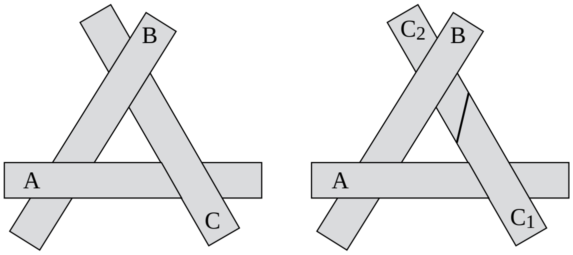
4 Depth order and the painter's algorithm

The painter's algorithm is a class of methods which consist in simply drawing the objects of the scene from back to front. This way, visible objects overwrite the hidden ones. This is similar to a painter who first draws a background then paints the foreground onto it. However, ordering objects according to their occlusion is not straightforward. Cycles may appear, as illustrated in Fig. 4.1(a).

The inverse order (Front to Back) can also be used, but a flag has to be indicate whether a pixel has been written or not. This order allows shading computations only for the visible pixels.

4.1 Newell Newell and Sancha

In the method by Newell, Newell and Sancha [NNS72] polygons are first sorted according to their minimum z value. However this order may not be the occlusion order. A bubble sort like scheme is thus applied. Polygons with overlapping z intervals are first compared in the image for xy overlap. If it is the case, their plane equation is used to test which occlude which. Cycles in occlusion are tested, in which case one of the polygons is split as shown in Fig. 4.1(b).



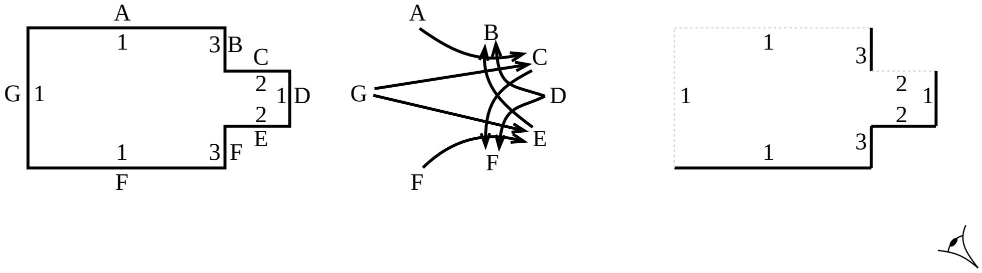
(a) (b)

Figure 4.1: (a) Classic example of a cycle in depth order. (b) Newell, Newell and Sancha split one of the polygons to break the cycle.

For new theoretical results on the problem of depth order, see the thesis by de Berg [Ber93].

4.2 Priority list preprocessing

Schumacker [SBGS69] developed the concept of *a priori* depth order. An object is preprocessed and an order may be found which is valid from any viewpoint (if the backfacing faces are removed). See the example of Fig. 4.2.



(a) (b) (c)

Figure 4.2: *A priori* depth order. (a) Lower number indicate higher priorities. (b) Graph of possible occlusions from any viewpoint. An arrow means that a face can occlude another one from a viewpoint. (c) Example of a view. Backfacing polygons are eliminated and other faces are drawn in the *a priori* order (faces with higher numbers are drawn first).

These objects are then organised in clusters which are themselves depth-ordered. This technique is fundamental for flight simulators where real-time display is crucial and where cluttered scenes are rare. Moreover, antialiasing is easier with list-priority methods because the coverage of a pixel can be maintained more consistently. The survey by Yan [Yan85] states that in 1985, all simulators were using depth order. It is only very recent that z-buffer has started to be used for flight simulators (see section below).

However, few objects can be *a priori* ordered, and the design of a suitable database had to be performed mainly by hand. Nevertheless, this work has led to the development of the BSP tree which we will present in section 1.4 of chapter 5

4.3 Layer ordering for image-based rendering

Recently, the organisation of scenes into layers for image-based rendering has revived the interest in depth-ordering *à la* Newell *et al.* Snyder and Lengyel [SL98] proposed the merging of layers which form an occlusion cycle, while Decoret *al.* [DSSD99] try to group layers which cannot have occlusion relations to obtain better parallax effects.

5 The z-buffer

5.1 Z-buffer

The z-buffer was developed by Catmull [Cat74, Cat75]. It is now the most widespread view computation method.

A depth (or z-value) is stored for each pixel of the image. As each object is scan-converted (or rasterized), the depth of each pixel it covers in the image is computed and compared against the corresponding current z-value. The pixel is drawn only if it is closer to the viewpoint.

Z-buffer was developed to handle curved surfaces, which are recursively subdivided until a sub-patch covers only one pixel. See also [CLR80] for improvements.

The z-buffer is simple, general and robust. The availability of cheap and fast memory has permitted very efficient hardware implementations at low costs, allowing today's low-end computer to render thousands of shaded polygons in real-time. However, due to the rasterized nature of the produced image, aliasing artifacts occur.

5.2 A-buffer

The *A-buffer* (antialiased averaged area accumulation buffer) is a high quality antialiased version of the z-buffer. A similar rasterization scheme is used. However, if a pixel is not completely covered by an object (typically at edges) a different treatment is performed. The list of object fragments which project on these non-simple pixels is stored instead of a color value (see Fig. 4.3). A pixel can be first classified non simple because an edge projects on it, then simple because a closer object completely covers it. Once all objects have been projected, sub-pixel visibility is evaluated for non-simple pixels. 4*8 subpixels are usually used. Another advantage of the A-buffer is its treatment of transparency; Subpixel fragments can be sorted in front-to-back order for correct transparency computations.

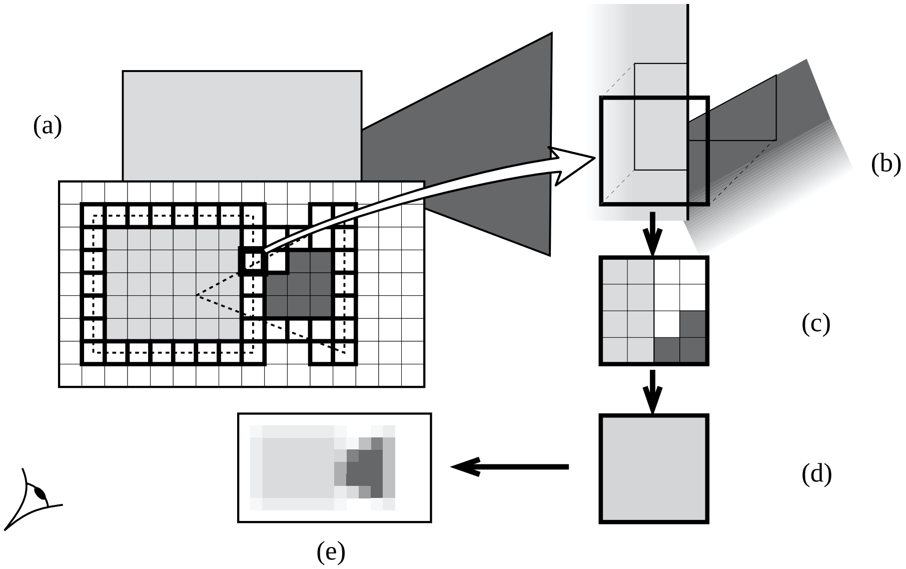


Figure 4.3: A buffer. (a) The objects are scan-converted. The projection of the objects is dashed and non-simple pixels are represented in bold. (b) Close-up of a non-simple pixel with the depth sorted fragments (*i.e.*, the polygons clipped to the pixel boundary). (c) The pixel is subsampled. (d) The resulting color is the average of the subsamples. (e) Resulting antialiased image.

The A-buffer can be credited to Carpenter [Car84], and Fiume *et al.* [FFR83]. It is a simplification of the "ultimate" algorithm by Catmull [Cat78] which used exact sub-pixel visibility (with a Weiler-Atherton clipping) instead of sub-sampling. A comprehensive introduction to the A-buffer and a discussion of implementation is given in the book by Watt and Watt [WW92].

The A-buffer is, with ray-tracing, the most popular high-quality rendering techniques. It is for example implemented in the commercial products Alias Wavefront Maya and Pixar Renderman [CCC87]. Similar techniques are apparently present in the hardware of some recent flight simulator systems [Mue95].

Most of the image space methods we present in chapter 6 are based on the z-buffer. A-buffer-like schemes could be explored when aliasing is too undesirable.

6 Scan-line

6.1 Scan-line rendering

Scan-line approaches produce rasterized images and consider one line of the image at a time. Their memory requirements are low, which explains why they have long been very popular. Wylie and his coauthors [WREE67] proposed the first scan-line algorithms, and Bouknight [Bou70] and Watkins [Wat70] then proposed very similar methods.

The objects are sorted according to *y*. For each scan-line, the objects are then sorted according to *x*. Then for each *span* (*x* interval on which the same objects project) the depths of the polygons are compared. See [WC91] for a discussion of efficient implementation. Another approach is to use a z-buffer for the current scan-line. The A-buffer [Car84] was in fact originally developed in a scan-line system.

Crocker [Cro84] has improved this method to take better advantage of coherence.

Scan-line algorithms have been extended to handle curved objects. Some methods [Cla79, LC79, LCWB80] use a subdivision scheme similar to Catmull's algorithm presented in the previous section while others [Bli78, Whi78, SZ89] actually compute the intersection of the surface with the current scan-line. See also [Rog97] page 417.

Sechrest and Greenberg [SG82] have extended the scanline method to compute *object precision* (exact) views. They place scan-lines at each vertex or edge-edge intersection in the image.

Tanaka and Takahashi [TT90] have proposed an antialiased version of the scan-line method where the image is scanned both in *x* and *y*. An adaptive scan is used in-between two *y* scan-lines. They have applied this scheme to soft shadow computation [TT97] (see also section 1.4 of chapter 8).

6.2 Shadows

The first shadowing methods were incorporated in a scan-line process as suggested by Appel [App68]. For each span (segment where the same polygon is visible) of the scan-line, its shadowing has to be computed. The wedge defined by the span and a point light-source is intersected with the other polygons of the scene to determine the shadowed part of the span.

In section 1.1 of chapter 6 we will see an improvement to this method. Other shadowing techniques for scan-line rendering will be covered in section 4.1 of chapter 5.

7 Ray-casting

The computation of visible objects using ray-casting was pioneered by Appel [App68], the Mathematical Application Group Inc. [MAG68] and Goldstein and Nagel [GN71] in the late sixties. The object visible at one pixel is determined by casting a ray through the scene. The ray is intersected with all objects. The closest intersection gives the visible object. Shadow rays are used to shade the objects. As for the z-buffer, Sutherland *et al.* [SSS74] considered this approach brute force and thought it was not scalable. They are now the two most popular methods.

As evoked in section 1.5 of chapter 2 Whitted [Whi80] and Kay [KG79] have extended ray-casting to *ray-tracing* which treats transparency and reflection by recursively sending secondary rays from the visible points.

Ray tracing can handle any type of geometry (as soon as an intersection can be computed). Various methods have been developed to compute ray-surface intersections, *e.g.*, [Kaj82, Han89].

Ray-tracing is the most versatile rendering technique since it can also render any shading effect. Antialiasing can be performed with subsampling: many rays are sent through a pixel (see *e.g.* [DW85, Mit87]).

Ray-casting and ray-tracing send rays from the eye to the scene, which is the opposite of actual physical light propagation. However, this corresponds to the theory of scientists such as Aristote who think that "visual rays" go from the eye to the visible objects.

As observed by Hofmann [Hof92] and illustrated in Fig. 4.4 ideas similar to ray-casting were exposed by Dürer [Dür38] while he was presenting perspective.

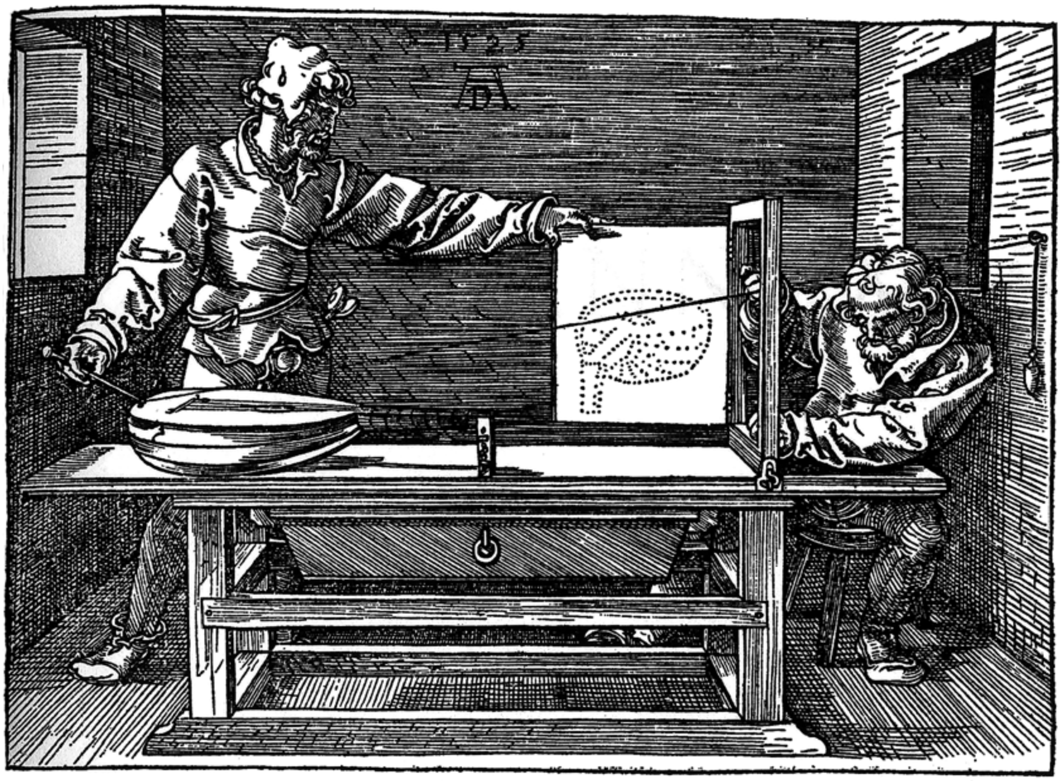


Figure 4.4: Drawing by Dürer in 1538 to illustrate his setting to compute perspective. It can be thought of as an ancestor of ray-casting. The artist's assistant is holding a stick linked to a string fixed at an eyebolt in the wall which represents the viewpoint. He points to part of the object. The position of the string in the frames is marked by the artist using the intersection of two strings fixed to the frame. He then rotates the painting and draws the point.

8 Sweep of the visibility map

Most of the algorithms developed in computational geometry to solve the hidden part removal problem are based on a sweep of the visibility map for polygonal scenes. The idea is illustrated in Fig. 4.5. The view is swept by a vertical (not necessarily straight) line, and computations are performed only at discrete steps often called events. A list of active edges (those crossing the sweep line) is maintained and updated at each events. Possible events are the appearance the vertex of a new polygon, or a *t-vertex*, that is, the visual intersection of an active edge and another edge (possibly not active).

The problem then reduces to the efficient detection of these events and the maintenance of the active edges. As evoked in the introduction this often involves some ray shooting queries (to detect which face becomes visible at a t-vertex for example). More complex queries are required to detect some t-vertices.

The literature on this subject is vast and well surveyed in the paper by Dorward [Dor94]. See also the thesis by de Berg [Ber93]. Other recent results on the subject include [Mul91, Pel96] (see section 1.5 of chapter 5).

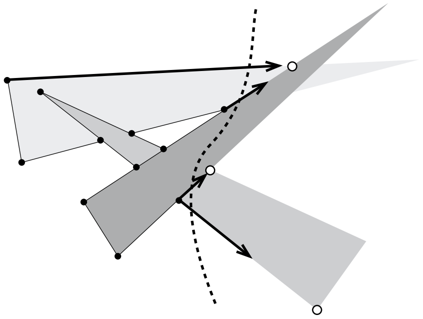


Figure 4.5: Sweep of a visibility map. Active edges are in bold. Already processed events are black points, while white points indicate the event queue.

CHAPTER 5

Object-Space

Ombres sans nombre
nombres sans ombre
à l'infini
au pas cadencé
Nombres des ombres
ombre des nombres
à l'infini
au pas commencé

Jacques PRÉVERT, *Fatras*

OBJECT-SPACE methods exhibit the widest range of approaches. We first introduce methods which optimize visibility computation by using a well-behaved subdivision of space. We then present two important data-structures based on the object-space locus of visual events, the 2D visibility graph (section 2) and visual hull (section 3). We then survey the large class of methods which characterize visibility using pyramid-like shapes. We review methods using beams for visibility with respect to a point in section 4. We then present the extensions of these methods to compute limits of umbra and penumbra in section 5, while section 6 discusses methods using shafts with respect to volumes. Finally section 7 surveys methods developed for visibility in architectural environments where visibility information is propagated through sequences of openings.

1 Space partitioning

If all objects are convex, simple, well structured and aligned, visibility computations are much easier. This is why some methods attempt to fit the scene into simple enclosing volumes or regular spatial-subdivisions. Computations are simpler, occlusion cycles can no longer occur and depth ordering is easy.

1.1 Ray-tracing acceleration using a hierarchy of bounding volumes

Intersecting a ray with all objects is very costly. Whitted [Whi80] enclosed objects in *bounding volumes* for which the intersection can be efficiently computed (spheres in his paper). If the ray does not intersect the bounding volume, it cannot intersect the object.

Rubin and Whitted [RW80] then extended this idea with hierarchies of bounding volumes, enclosing bounding volumes in a hierarchy of successive bounding volumes. The trade-off between how the bounding volumes fits the object and the cost of the intersection has been studied by Weghorst *et al.* [WHG84] using a probabilistic approach based on surface ratios (see also section 4 of chapter 8). Kay and Kajiya [KK86] built tight-fitting bounding volumes which approximate the convex hull of the object by the intersection of parallel slabs.

The drawback of standard bounding volume methods, is that all objects intersecting the rays have to be tested. Kay and Kajiya [KK86] thus propose an efficient method for a traversal of the hierarchy which first tests the closest bounding volumes and terminates when an intersection is found which is closer than the remaining bounding volumes.

Many other methods were proposed to improve bounding volume methods for ray-tracing, see *e.g.* [Bou85, AK89, FvDFH90, Rog97, WW92]. See also [Smi99] for efficiency issues.

1.2 Ray-tracing acceleration using a spatial subdivision

The alternative to bounding volumes for ray-tracing is the use of a structured spatial subdivision. Objects of the scene are classified against *voxels* (boxes), and shooting a ray consists in traversing the voxels of the subdivision and performing intersections only for the objects inside the encountered voxels. An object can lie inside many voxels, so this has to be taken into account.

The trade-off here is between the simplicity of the subdivision traversal, the size of the structure and the number of objects per voxel.

Regular grids have been proposed by Fujimoto *et al.* [FTI86] and Amanatides and Woo [AW87]. The drawback of regular grids is that regions of high object density are “sampled” at the same rate as regions with many objects, resulting in a high cost for the latter because one voxel may contain many objects. However the traversal of the grid is very fast, similar to the rasterization of a line on a bitmap image. To avoid the time spent in traversing empty regions of the grid, the distance to the closest object can be stored at each voxel (see *e.g.* [CS94, SK97]).

Glassner [Gla84] introduced the use of octrees which result in smaller voxels in regions of high object density. Unfortunately the traversal of the structure becomes more costly because of the cost induced by the hierarchy of the octree. See [ES94] for a comparison between octrees and regular grids.

Recursive grids [JW89, KS97] are similar to octrees, except that the branching factor may be higher, which reduces the depth of the hierarchy (see Fig. 5.1(a)). The size of the voxel in a grid or sub-grid should be proportional to the cubic root of the number of objects to obtain a uniform density.

Snyder and Bar [SB87] use tight fitting regular grids for complex tessellated objects which they insert in a bounding box hierarchy.

Finally Cazals *et al.* [CDP95, CP97] propose the Hierarchy of Uniform Grids, where grids are not nested. Objects are sorted according to their size. Objects which are close and have the same size are clustered, and a grid is used for each cluster and inserted in a higher level grid (see Fig. 5.1(b)). An in-depth analysis of the performance of spatial subdivision methods is presented. Recursive grids and the hierarchy of uniform grid seem to be the best trade-off at the moment (see also [KWCH97, Woo97] for a discussion on this subject).

1.3 Volumetric visibility

The methods in the previous sections still require an intersection calculations for each object inside a voxel. In the context of radiosity lighting simulation, Sillion [Sil95] approximates visibility inside a voxel by an attenuation factor (transparency or *transmittance*) as is done for volume rendering. A multiresolution extension was presented [SD95] and will be discussed in section 1.2 of chapter 9.

The transmittance is evaluated using the area of the objects inside a voxel. These voxels (or *clusters*) are organised in a hierarchy. Choosing the level of the hierarchy used to compute the attenuation along a ray allows a trade-off between accuracy and time. The problem of refinement criteria will be discussed in section 1.1 of chapter 9.

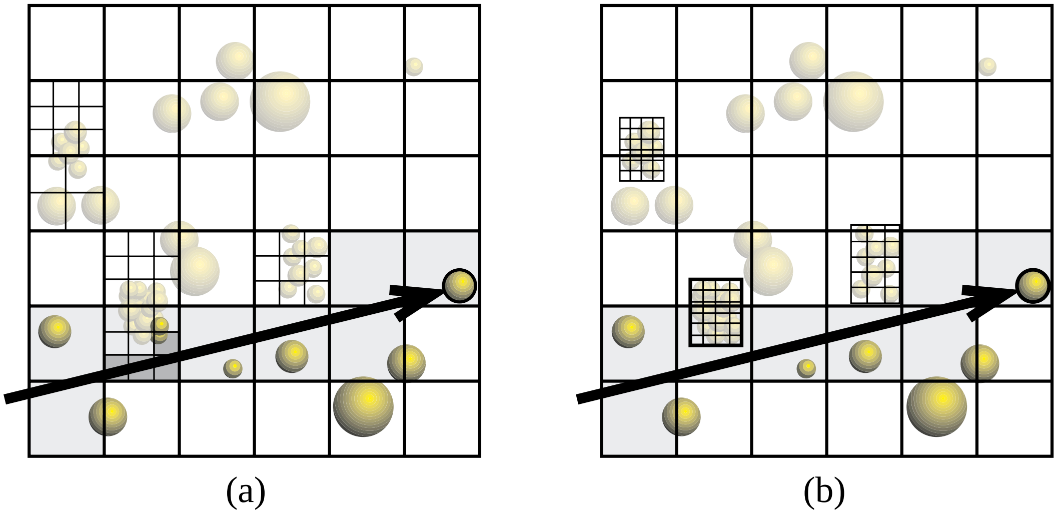


Figure 5.1: A 2D analogy of ray-tracing acceleration. An intersection test is performed for objects which are in bold type. (a) Recursive grid. (b) Hierarchy of uniform grids. Note that fewer intersections are computed with the latter because the grids fit more tightly to the geometry.

Christensen *et al.* [CLSS97] propose another application of volumetric visibility for radiosity.

Chamberlain *et al* [CDL$^+$96] perform real-time rendering by replacing distant geometry by semi-transparent regular voxels averaging the color and occlusion of their content. Neyret [Ney96, Ney98] presents similar ideas to model and render complex scenes with hierarchical volumetric representations called *texels*.

1.4 BSP trees

We have seen in section 4.2 of chapter 4 that an *a priori* depth order can be found for some objects. Unfortunately, this is quite rare. Fuchs and his co authors [FKN80, FAG83] have developed the *BSP* tree (Binary Space Partitioning tree) to overcome this limitation.

The principle is simple: if the scene can be separated by a plane, the objects lying on the same side of the plane as the viewer are closer than the others in a depth order. BSP trees recursively subdivide the scene along planes, resulting in a binary tree where each node corresponds to a splitting plane. The computation of a depth order is then a straightforward tree traversal: at each node the order in which the subtrees have to be drawn is determined by the side of the plane of the viewer. Unfortunately, since a scene is rarely separable by a plane, objects have to be split. Standard BSP approaches perform subdivision along the polygons of the scene. See Fig. 5.2 for an example[1].

It has been shown [PY90] that the split in BSP trees can cause the number of sub-polygons to be as high as $O(n^3)$ for a scene composed of n entry polygons. However, the choice of the order of the polygons with which subdivision is performed is very important. Paterson and Yao [PY90] give a method which builds a BSP tree with size $O(n^2)$. Unfortunately, it requires $O(n^3)$ time. However these bounds do not say much on the practical behaviour of BSPs.

See *e.g.* [NR95] for the treatment of curved objects.

Agarwal *et al.* [AGMV97, AEG98] do not perform subdivision along polygons. They build *cylindrical* BSP trees, by performing the subdivision along vertical planes going through edges of the scene (in a way similar to the method presented in the next section). They give algorithms which build a quadratic size BSP in roughly quadratic time.

Chen and Wang [CW96] have proposed the *feudal priority* algorithm which limits the number of splits compared to BSP. They first treat polygons which are back or front-facing from any other polygon, and then chose the polygons which cause the smallest number of splits.

[1] BSP trees have also been applied as a modeling representation tool and powerful *Constructive Solid Geometry* operations have been adapted by Naylor *et al.* [NAT90].

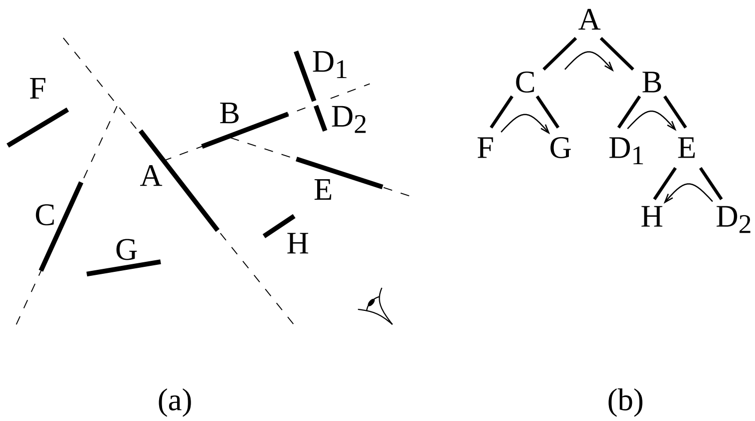


Figure 5.2: 2D BSP tree. (a) The scene is recursively subdivided along the polygons. Note that polygon D has to be split. (b) Corresponding binary tree. The traversal order for the viewpoint in (a) is depicted using arrows. The order is thus, from back to front: $FCGAD_1BHED_2$

Naylor [Nay92] also uses a BSP tree to encode the image to perform occlusion-culling; nodes of the object-space BSP tree projecting on a covered node of the image BSP are discarded in a manner similar to the hierarchical z-buffer which we will present in section 3 of the next chapter.

BSP trees are for example in the game *Quake* for the hidden-surface removal of the static part if the model [Abr96] (moving objects are treated using a z-buffer).

1.5 Cylindrical decomposition

Mulmuley [Mul91] has devised an efficient preprocessing algorithm to perform object-precision view computations using a sweep of the view map as presented in section 8 of chapter 4. However this work is theoretical and is unlikely to be implemented. He builds a cylindrical partition of 3D space which is similar to the BSPs that Agarwall *et al.* [AGMV97, AEG98] have later described. Nonetheless, he does not use whole planes. Each cell of his partition is bounded by parts of the input polygons and by vertical walls going through edges or vertices of the scene. His paper also contains an interesting discussion of sweep algorithms.

2 Path planning using the visibility graph

2.1 Path planning

Nilsson [Nil69] developed the first path planning algorithms. Consider a 2D polygonal scene. The *visibility graph* is defined as follows: The nodes are the vertices of the scene, and an arc joins two vertices A and B if they are mutually visible, *i.e.* if the segment $[AB]$ intersects no obstacle. As noted in the introduction, it is possible to go in straight line from A to B only if B is visible from A. The start and goal points are added to the set of initial vertices, and so are the corresponding arcs (see Fig. 5.3). Only arcs which are tangent to a pair of polygons are necessary.

It can be easily shown that the shortest path between the start point and the goal goes through arcs of the visibility graph. The rest of the method is thus a classical graph problem. See also [LPW79].

This method can be extended to non-polygonal scenes by considering bitangents and portions of curved objects.

The method unfortunately does not generalize simply to 3D where the problem has been shown to be NP-complete by Canny [Can88].

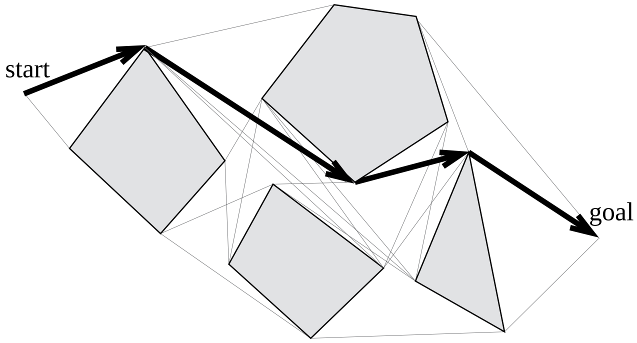


Figure 5.3: Path planning using the visibility graph.

2.2 Visibility graph construction

The 2D visibility graph has size which is between linear and quadratic in the number of polygon edges. The construction of visibility graphs is a rich subject of research in computational geometry. Optimal $O(n^2)$ algorithms have been proposed [EG86] as well as *output-sensitive* approaches (their running time depends on the size of the output, *i.e.* the size of the visibility graph) [OW88, GM91].

The *2D visibility complex* which we will review in section 1.2 of chapter 8 is also a powerful tool to build visibility graphs.

In 3D, the term “visibility graph” often refers to the abstract graph where each object is a node, and where arcs join mutually visible objects. This is however not the direct equivalent of the 2D visibility graph.

2.3 Extensions to non-holonomic visibility

In this section we present some motion planning works which are hard to classify since they deal with extensions of visibility to curved lines of sight. They have been developed by Vendittelli *et al.* [VLN96] to plan the motion of a car-like robot. Car trajectories have a minimum radius of curvature, which constraints their motion. They are submitted to *non-holonomic* constraints: the tangent of the trajectory must be colinear to the velocity. Dubins [Dub57] and Reeds and Shepp [RS90] have shown that minimal-length trajectories of bounded curvature are composed of arcs of circles of minimum radius and line segments.

For example if a car lies at the origin of the plane and is oriented horizontally, the shortest path to the points of the upper quadrant are represented in Fig. 5.4(a). The rightmost paths are composed of a small arc of circle forward followed by a line segment. To go to the points on the left, a backward circle arc is first necessary, then a forward arc, then a line segment.

Now consider an obstacle such as the line segment represented in Fig. 5.4(a). It forbids certain paths. The points which cannot be reached are said to be in shadow, by analogy to the case where optimal paths are simple line segments[2].

The shape of such a shadow can be much more complex than in the line-visibility case, as illustrated in Fig. 5.4(b).

This analogy between visibility and reachability is further exploited in the paper by Nissoux *et al.* [NSL99] where they plan the motion of robots with arbitrary numbers of degrees of freedom.

3 The Visual Hull

The reconstruction of objects from silhouettes (see section 2.2 of chapter 2) is very popular because it is robust and simple. Remember that only exterior silhouettes are considered, folds caused by self occlusion of the object are not considered because they are harder to extract from images. Not all objects can be reconstructed with

[2] What we describe here are in fact shadows in a Riemannian geometry. Our curved lines of sight are in fact *geodesics*, *i.e.*c the shortest path from one point to another.

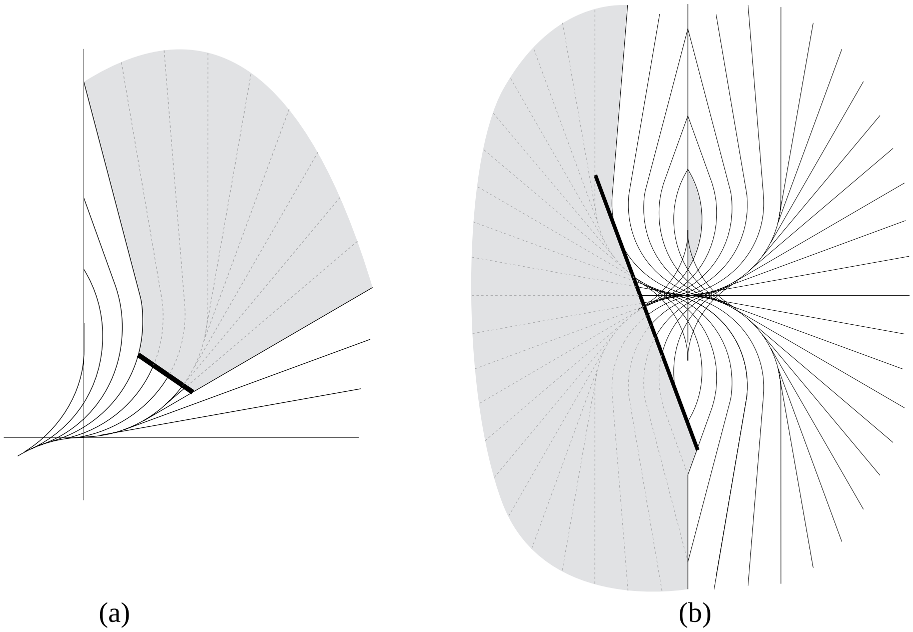


Figure 5.4: Shadow for non-holonomic path-planning (adapted from [VLN96]). (a) Simple (yet curved) shadow. (b) Complex shadows. Some parts of the convex blocker do not lie on the shadow boundary. The small disconnected shadow is caused by the impossibility to perform an initial backward circle arc.

this method; The cavity of a bowl can not be reconstructed because it is not present on an external silhouette. The best reconstruction of a bowl one can expect is a "full" version of the initial object.

However the reconstructed object is not necessarily the convex hull of the object: the hole of a torus can be reconstructed because it is present on the exterior silhouette of some images.

Laurentini [Lau94, Lau95, Lau97, Lau99] has introduced the *visual hull* concept to study this problem. A point P of space is inside the visual hull of an object A, if from any viewpoint P projects inside the projection of A. To give a line-space formulation, each line going through a point P of the visual hull intersects object A. The visual hull is the smallest object which can be reconstructed from silhouettes. See Fig. 5.5 for an example.

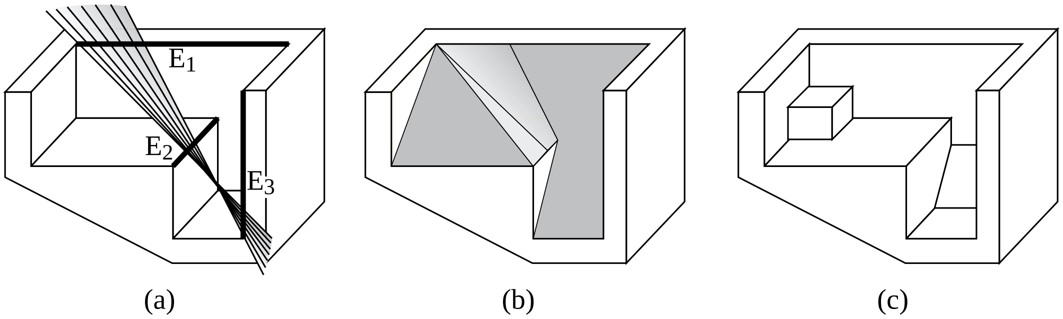


Figure 5.5: Visual hull (adapted from [Lau94]). (a) Initial object. A EEE event is shown. (b) Visual hull of the object (the viewer is not allowed inside the convex hull of the object). It is delimited by polygons and a portion of the ruled quadric of the $E_1E_2E_3$ event. (c) A different object with the same visual hull. The two objects can not be distinguished from their exterior silhouette and have the same occlusion properties.

The exact definition of the visual hull in fact depends on the viewing region authorized. The visual hull is different if the viewer is allowed to go inside the convex hull of the object. (Half lines have to be considered instead of lines in our line-space definition)

The visual hull is delimited by visual events. The visual hull of a polyhedron is thus not necessarily a polyhedron, as shown in Fig. 5.5 where a EEE event is involved.

Laurentini has proposed a construction algorithms in 2D [Lau94] and for objects of revolution in 3D [Lau99]. Petitjean [Pet98] has developed an efficient construction algorithm for 2D visual hulls using the visibility graph.

The visual hull also represents the maximal solid with the same occlusion properties as the initial object. This concept thus completely applies to the simplification of occluders for occlusion culling. The simplified occluder does not need to lie inside the initial occluder, but inside its visual hull. See the work by Law and Tan [LT99] on occluder simplification.

4 Shadows volumes and beams

In this section we present the rich category of methods which perform visibility computation using pyramids or cones. The apex can be defined by the viewpoint or by a point light source. It can be seen as the volume occupied by the set of rays emanating from the apex and going through a particular object. The intersection of such a volume with the scene accounts for the occlusion effects.

4.1 Shadow volumes

Shadow volumes have been developed by Crow [Cro77] to compute hard shadows. They are pyramids defined by a point light source and a blocker polygon. They are then used in a scan-line renderer as illustrated in Fig. 5.6.

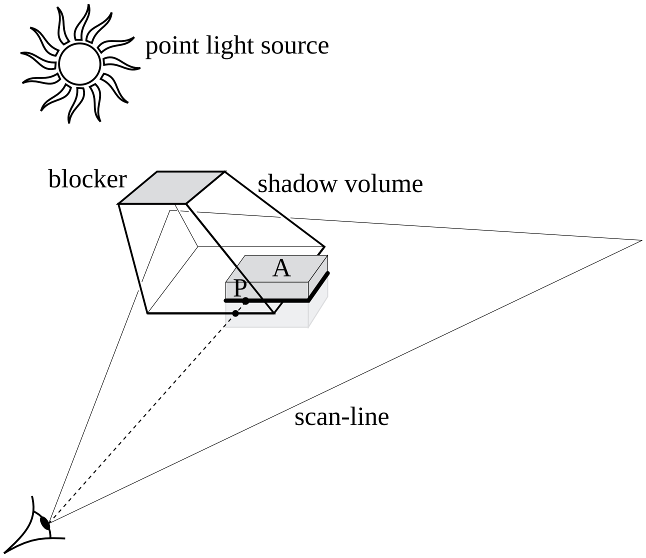


Figure 5.6: Shadow volume. As object *A* is scan converted on the current scan-line, the shadowing of each pixel is computed by counting the number of back-facing and front-facing shadow volume polygons on the line joining it to the viewpoint. For point P, there is one front-facing intersection, it is thus in shadow.

The wedges delimiting shadow volumes are in fact visual events generated by the point light source and the edges of the blockers. In the case of a polyhedron light source, only silhouette edges (with respect to the source) need to be considered to build the shadow volume polygons.

Bergeron [Ber86] has proposed a more general version of Crow's shadow volumes. His method has long been very popular for production rendering.

Shadow volumes have also been used with ray-tracing [EK89]. Brotman and Badler [BB84] have presented a z-buffer based use of shadow volumes. They first render the scene in a z-buffer, then they build the shadow volumes and scan convert them. Instead of displaying them, for each pixel they keep the number of frontfacing and backfacing shadow volume polygons. This method is hybrid object-space and image space, the advantage over the shadow map is that only one sampling is performed. They also sample an area light source with points and add the contributions computed using their method to obtain soft shadow effects. An implementation using current graphics hardware is described in [MBGN98] section 9.4.2. A hardware implementation has also been developed on *pixel-plane* architecture [FGH$^+$85], except that shadow volumes are simply described as plane-intersections.

Shadow volumes can also be used inversely as light-volumes to simulate the scattering of light in dusty air (*e.g.*, [NMN87, Hai91]).

Albrecht Dürer [Dür38] describes similar constructions, as shown in Fig. 5.7

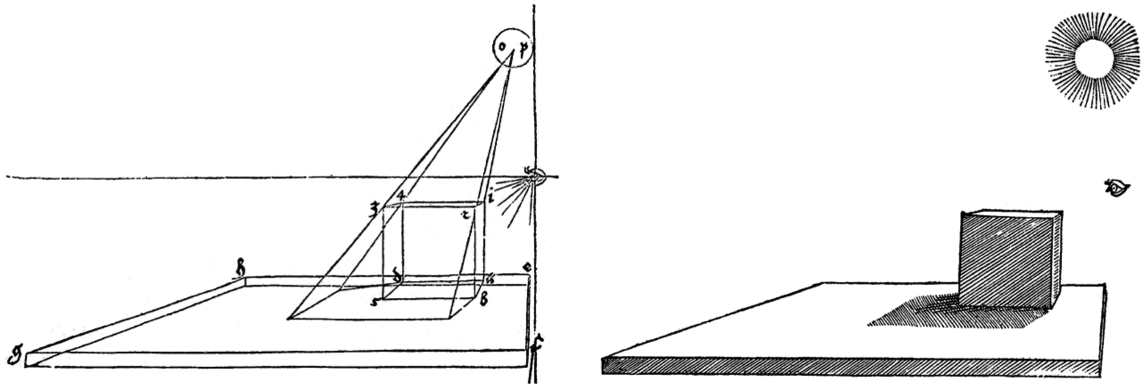


Figure 5.7: Construction of the shadow of a cube by Dürer.

4.2 Shadow volume BSP

Chin and Feiner [CF89] compute hard shadows using BSP trees. Their method can be compared to Atherton *et al.*'s technique presented in section 2.1 of chapter 4 where the same algorithm is used to compute the view and to compute the illuminated parts of the scene. Two BSP are however used: one for depth ordering, and one called *shadow BSP tree* to classify the lit and unlit regions of space.

The polygons are traversed from front to back from the light source (using the first BSP) to build a shadow BSP tree. The shadow BSP tree is split along the planes of the shadow volumes. As a polygon is considered, it is first classified against the current shadow BSP tree (Fig. 5.8(a)). It is split into lit and unlit parts. Then the edges of the lit part are used to generate new splitting planes for the shadow BSP tree (Fig. 5.8 (b)).

The scene augmented with shadowing information can then be rendered using the standard BSP.

Chrysanthou and Slater [CS95] propose a method which avoids the use of the scene BSP to build the shadow BSP, resulting in fewer splits.

Campbell and Fussel [CF90] were the first to subdivide a radiosity mesh along shadow boundaries using BSPs. A good discussion and some improvements can be found in Campbell's thesis [Cam91].

4.3 Beam-tracing and bundles of rays

Heckbert and Hanrahan [HH84] developed *beam tracing*. It can be seen as a hybrid method between Weiler and Atherton's algorithm [WA77], Whitted's ray-tracing [Whi80] and shadow volumes.

Beams are traced from the viewpoint into the scene. One initial beam is cast and clipped against the scene polygons using Weiler and Atherton's exact method, thus defining smaller beams intersecting only one polygon (see Fig. 5.9(a)). If the a polygon is a mirror, a reflection beam is recursively generated. Its apex is the symmetric to the viewpoint with respect to the light source (Fig. 5.9(b)). It is clipped against the scene, and the computation proceeds.

Shadow beams are sent from the light source in a preprocess step similar to Atherton *et al*'s shadowing [AWG78]. Refraction can be approximated by sending refraction beams. Unfortunately, since refraction is not linear, this computation is not exact.

Dadoon *et al.* [DKW85] propose an efficient version optimized using BSP trees.

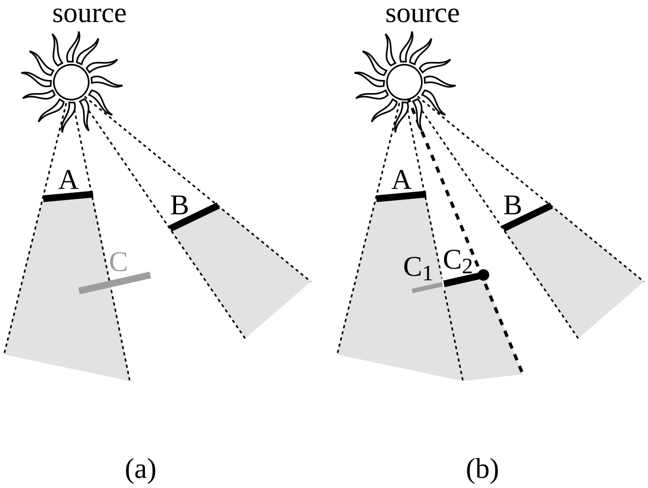


Figure 5.8: 2D equivalent of shadow BSP. The splitting planes of the shadow BSP are represented with dashed lines. (a) Polygon C is tested against the current shadow BSP. (b) It is split into a part in shadow C_1 and a lit part C_2. The boundary of the lit part generates a new splitting plane.

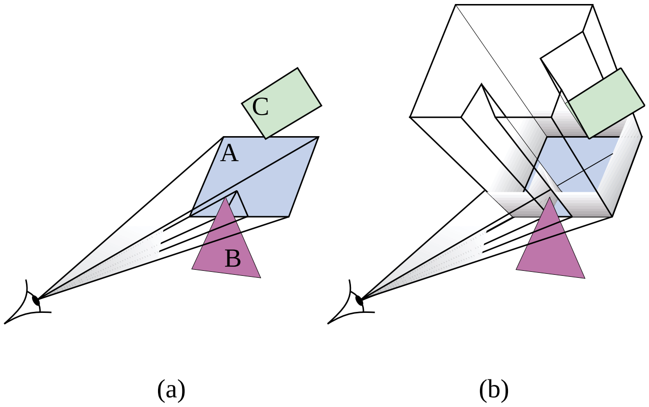


Figure 5.9: Beam tracing. (a) A beam is traced from the eye to the scene polygons. It is clipped against the other polygons. (b) Since polygon A is a mirror, a reflected beam is recursively traced and clipped.

Amanatides [Ama84] and Kirk [Kir87] use cones instead of beams. *Cone-tracing* allows antialiasing as well as depth-of-field and soft shadow effects. The practical use of this method is however questionable because secondary cones are hard to handle and because object-cone intersections are difficult to perform. Shinya *et al.* [STN87] have formalized these concepts under the name of *pencil tracing*.

Beam tracing has been used for efficient specular sound propagation by Funkhouser and his co-author. [FCE$^+$98]. A bidirectional version has also been proposed where beams are propagated both from the sound source and from the receiver [FMC99].They moreover *amortize* the cost of beam propagation as listeners and sources move smoothly.

Speer [SDB85] has tried to take advantage of the coherence of bundles of rays by building cylinders in free-space around a ray. If subsequent rays are within the cylinder, they will intersect the same object. Unfortunately his method did not procure the expected speed-up because the construction of the cylinders was more costly than a brute-force computation.

Beams defined by rectangular windows of the image can allow high-quality antialiasing with general scenes. Ghazanfarpour and Hasenfratz [GH98, Has98] classify non-simple pixels in a manner similar to the A-buffer or to the ZZ-buffer, but they take shadows, reflection and refraction into account.

Teller and Alex [TA98] propose the use of beam-casting (without reflection) in a real-time context. Beams are adaptively subdivided according to a time budget, permitting a trade-off between time and image quality.

Finally Watt [Wat90] traces beams from the light source to simulate caustic effects which can for example be caused by the refraction of light in water.

4.4 Occlusion culling from a point

Sometimes, nearby large objects occlude most of the scene. This is the case in a city where nearby facades hide most of the buildings. Coorg and Teller [CT96, CT97b] quickly reject the objects hidden by some convex polygonal occluders. The scene is organised into an octree. A Shadow volume is generated for each occluder, and the cells of the octree are recursively classified against it as occluded, visible or partially visible, as illustrated in Fig. 5.10.

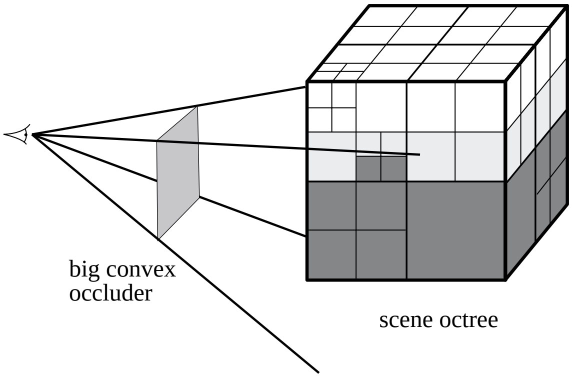


Figure 5.10: Occlusion culling with large occluders. The cells of the scene octree are classified against the shadow volumes. In dark grey we show the hidden cells, while those partially occluded are in light grey.

The occlusion by a conjunction of occluders in not taken into account, as opposed to the hierarchical z-buffer method exposed in section 3 of chapter 6. However we will see in section 4.2 of chapter 7 that they treat frame-to-frame coherence very efficiently.

Similar approaches have been developed by Hudson *et al.* [HMC+97]. Bittner *et al.* [BHS98] use shadow volume BSP tree to take into account the occlusion caused by multiple occluders.

Woo and Amanatides [WA90] propose a similar scheme to speed-up hard shadow computation in ray-tracing. They partition the scene in a regular grid and classify each voxel against shadow volumes as completely lit, completely in umbra or complicated. Shadow rays are then sent only from complicated voxels.

Indoor architectural scenes present the dual characteristic feature to occlusion by large blockers: one can see outside a room only through doors or windows. These opening are named *portals*. Luebke and George [LG95] following ideas by Jones [Jon71] and Clark [Cla76] use the portals to reject invisible objects in adjacent rooms. The geometry of the current room is completely rendered, then the geometry of adjacent rooms is tested against the screen bounding box of the portals as shown in Fig. 5.11. They also apply their technique to the geometry reflected by mirrors.

This technique was also used for a walk through a virtual colon for the inspection of acquired medical data [HMK+97] and has been implemented in a 3D game engine [BEW+98].

4.5 Best-next-view

Best-next-view methods are used in model reconstruction to infer the position of the next view from the data already acquired. The goal is to maximize the visibility of parts of the scene which were occluded in the previous view. They are delimited by the *volume of occlusion* as represented in Fig. 5.12. These volumes are in fact the *shadow volumes* where the camera is considered as a light source.

Reed and Allen [RA96] construct a BSP model of the object as well as the boundaries of the occlusion volume. They then attempt to maximize the visibility of the latter. This usually results roughly in a 90 ° rotation of the camera since the new viewpoint is likely to be perpendicular to the view volume.

Similar approaches have been developed by Maver and Bajcsy [MB93] and Banta *et al.* [BZW+95].

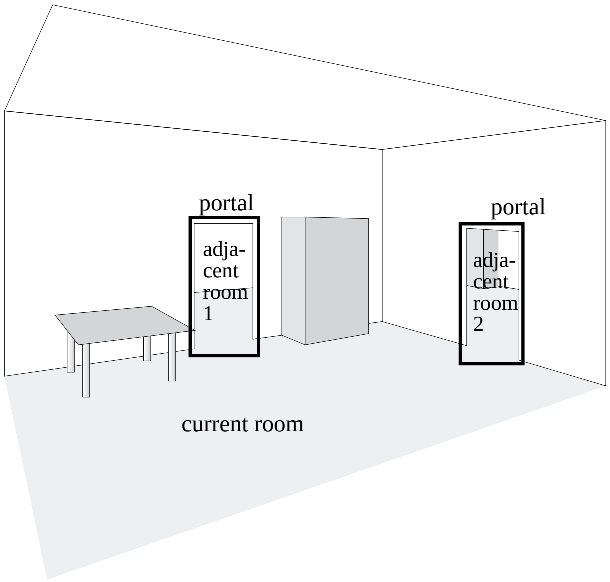


Figure 5.11: Occlusion culling using image-space portals. The geometry of the adjacent rooms is tested against the screen bounding boxes of the portals

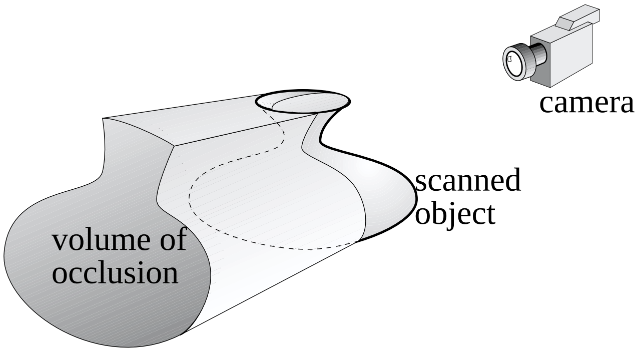


Figure 5.12: Acquisition of the model of a 3D object using a range image. The volume of occlusion is the unknown part of space.

This problem is very similar to the problem of gaps in image-based view warping (see section 1.7 of chapter 2 and Fig. 2.7 page 12). When a view is reprojected, the regions of indeterminate visibility lie on the boundary of the volumes of occlusion.

5 Area light sources

5.1 Limits of umbra and penumbra

Nishita and Nakamae [NN85, NON85, NN83] have computed the regions of umbra and penumbra caused by convex blockers. They show that the umbra from a polygonal light source of a convex object is the intersection of the umbra volumes from the vertices of the source (see Fig. 5.13). The penumbra is the convex hull of the union of the umbra volumes. They use Crow's shadow volumes to compute these regions.

The umbra is bounded by portions of EV events generated by one vertex of the source and one edge of the blocker, while the penumbra is bounded EV events generated by edges and vertices of both the source and the blocker.

Their method fails to compute the exact umbra caused by multiple blockers, since it is no longer the inter-

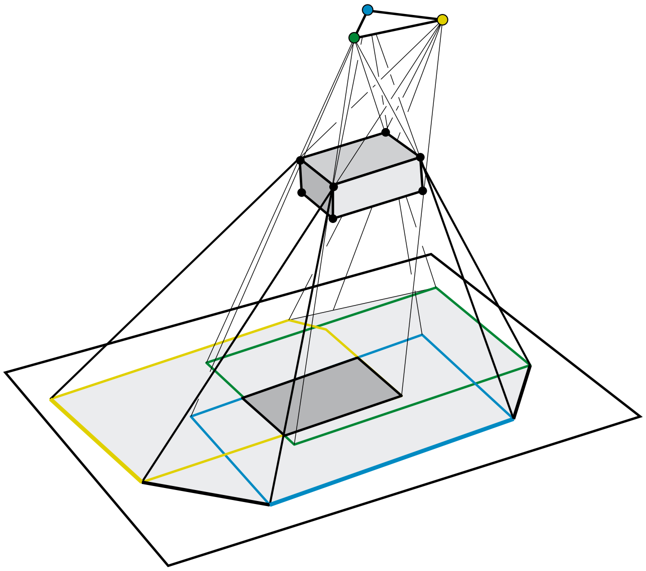


Figure 5.13: Umbra (dark grey) and penumbra (light grey) of a convex blocker (adapted from [NN85]).

section of their umbras. The penumbra boundary is however valid, but some parts of the umbra are incorrectly classified as penumbra. This is not a problem in their method because a shadow computation is performed in the penumbra region (using an exact hidden line removal method). The umbra of a concave object is bounded by EV visual events and also by EEE events (for example in Fig. 3.5 page 27 if polygon R is a source, the EEE event exhibited is an umbra boundary). Penumbra regions are bounded only by EV events.

Drawings by da Vinci exhibit the first description of the limits of umbra and penumbra (Fig. 5.14).

5.2 BSP shadow volumes for area light sources

Chin and Feiner [CF92] have extended their BSP method to handle area light sources. They build two shadow BSP, one for the umbra and one for the penumbra.

As in Nishita and Nakamae's case, their algorithm does not compute the exact umbra volume due to the occlusion by multiple blockers.

5.3 Discontinuity meshing

Heckbert [Hec92b, Hec92a] has introduced the notion of discontinuity meshing for radiosity computations. At a visual event, a C^2 discontinuity occurs in the illumination function (see [Arv94] for the computation of illumination gradients). Heckbert uses EV discontinuity surfaces with one generator on the source.

Other authors [LTG93, LTG92, Stu94, Cam91, CF91a, GH94] have used similar techniques. See Fig. 5.15 for an example. Hardt and Teller [HT96] also consider discontinuities which are caused by indirect lighting. Other discontinuity meshing techniques will be treated in section 2.3 of chapter 7 and 2.1 of chapter 8.

However, discontinuity meshing approaches have not yet been widely adopted because they are prone to robustness problems and also because the irregular meshes induced are hard to handle.

5.4 Linear time construction of umbra volumes

Yoo *et al.* [YKSC98] perform the same umbra/penumbra classification as Nishita and Nakamae, but they avoid the construction and intersection/union of all the shadow volumes from the vertices of the source.

They note that only EV events on separating and supporting planes have to be considered. Their algorithm walks along the chain of edges and vertices simultaneously on the source and on the blocker as illustrated in Fig. 5.16.

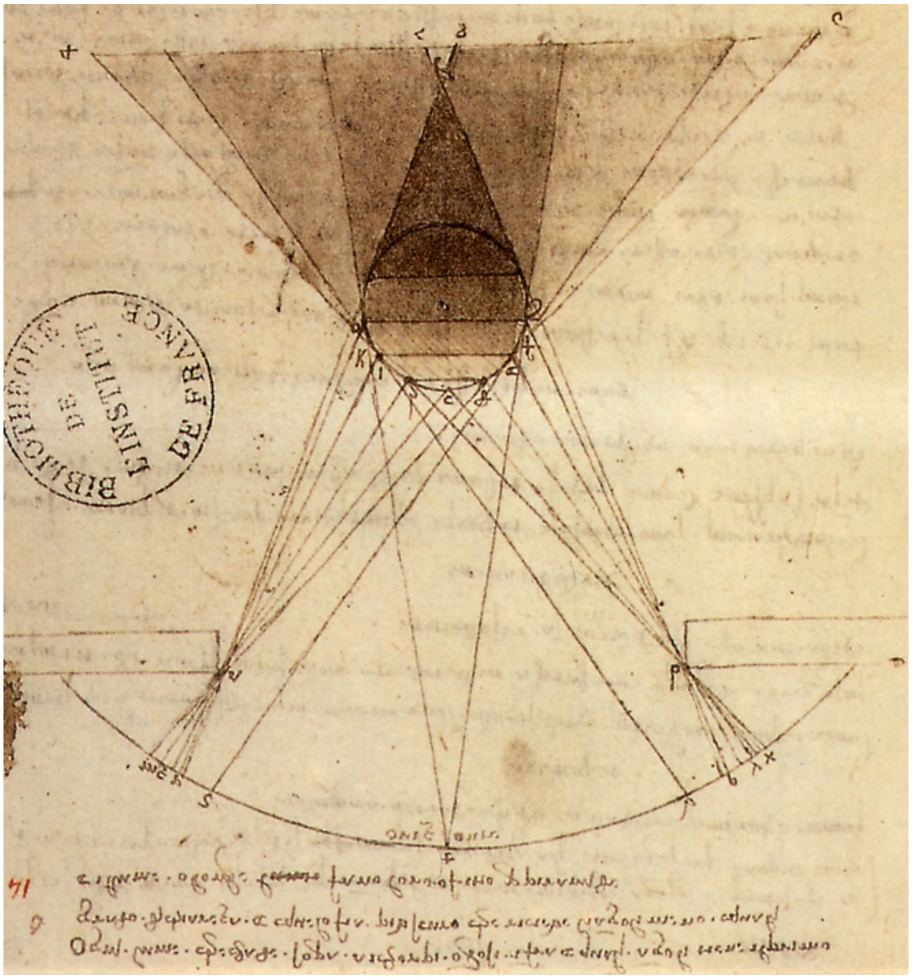


Figure 5.14: Penumbra by Leonardo da Vinci (Manuscript). Light is coming from the lower window, and the sphere causes soft shadows.

This can be interpreted in line space as a walk along the chain of 1 dimensional sets of lines defined by visual events.

Related methods can be found in [Cam91, TTK96].

5.5 Viewpoint constraints

As we have seen, viewpoint optimisation is often performed for the monitoring of robotics tasks. In this setting, the visibility of a particular feature of object has to be enforced. This is very similar to the computation of shadows considering that the feature is an extended light source.

Cowan and Kovesi [CK88] use an approach similar to Nishita and Nakamae. They compute the penumbra region caused by a convex blocker as the intersection of the half spaces defined by the separating planes of the feature and blockers (*i.e.* planes tangent to both objects such that each object lies on a different side of the plane). The union of the penumbra of all the blockers is taken and constraints related to the sensor are then included: resolution of the image, focus, depth of field and view angle. The admissible region is the intersection of these constraints.

Briggs and Donald [BD98] propose a 2D method which uses the intersection of half-planes defined by bitangents. They also reject viewpoints from which the observation can be ambiguous because of similarities in the workspace or in the object to be manipulated.

Tarabanis and Tsai [TTK96] compute occlusion free viewpoints for a general polyhedral scene and a general

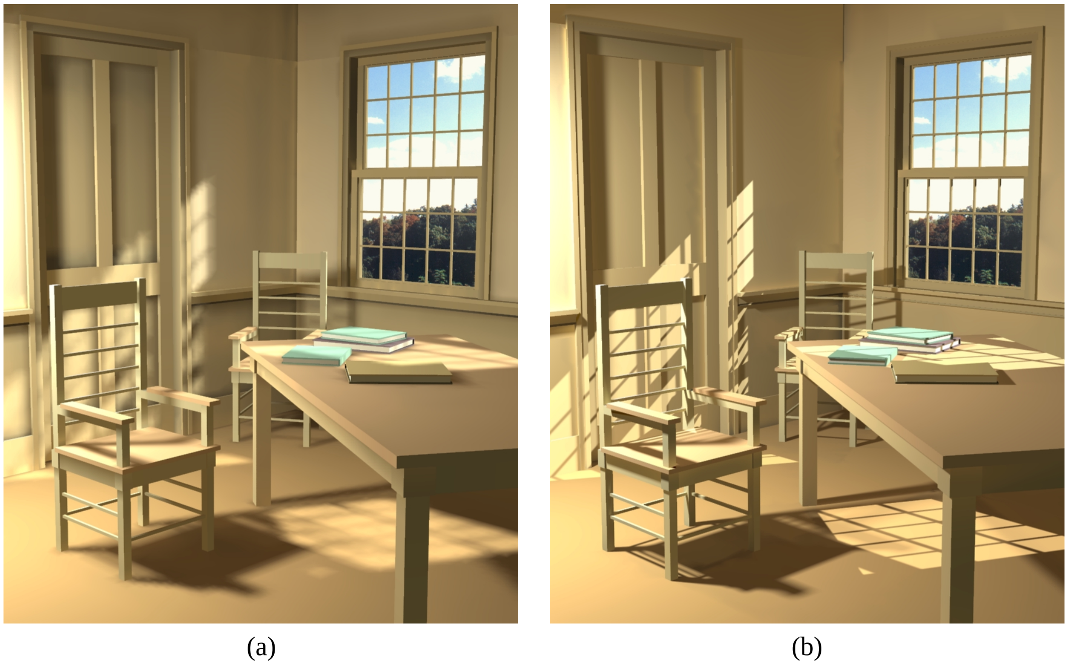

(a) (b)

Figure 5.15: Global illumination simulation. (a) Without discontinuity meshing. Note the jagged shadows. (b) Using discontinuity meshing, shadows are finer (images courtesy of Dani Lischinski, Program of Computer Graphics, Cornell University).

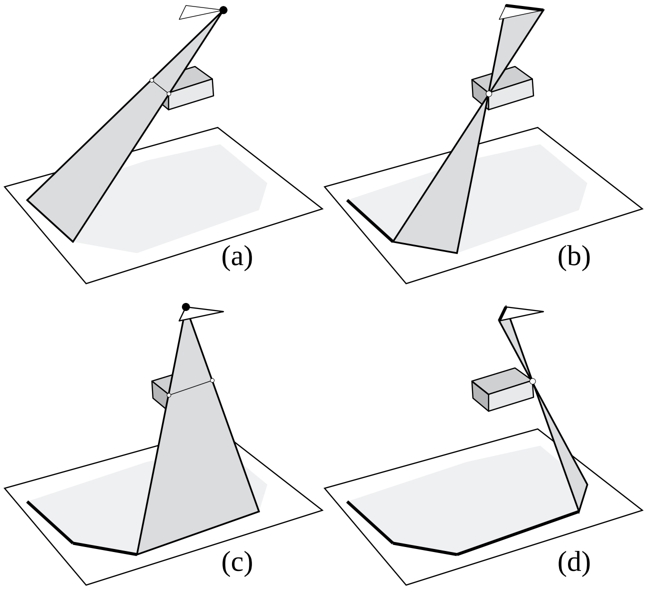

(a) (b) (c) (d)

Figure 5.16: Linear time construction of a penumbra volume.

polygonal feature. They enumerate possible *EV* wedges and compute their intersection.

Kim *et al.* [KYCS98] also present an efficient algorithm which computes the complete visibility region of a convex object.

5.6 Light from shadows

Poulin *et al.* [PF92, PRJ97] have developed inverse techniques which allow a user to sketch the positions of shadows. The position of the light source is then automatically deduced.

The principle of shadow volumes is reversed: A point *P* lies in shadow if the point light source is in a shadow volume emanating from point *P*. The sketches of the user thus define constraints under the form of an intersection of shadow volumes (see Fig. 5.17).

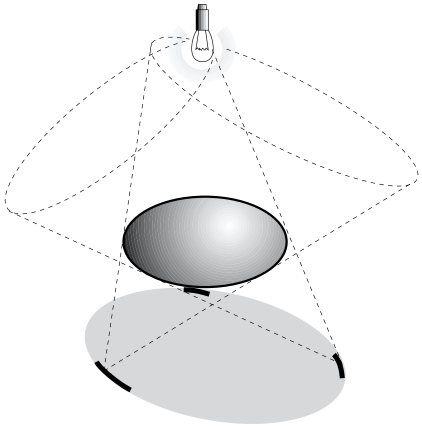


Figure 5.17: Sketching shadows. The user specifies the shadows of the ellipsoid on the floor with the thick strokes. This generates constraint cones (dashed). The position of the light source is then deduced (adapted from [PRJ97]).

Their method can also handle soft shadows, and additional constraints such as the position of highlights.

6 Shafts

Shaft method are based on the fact that occlusion between two objects can be caused only by objects inside their convex hull. Shafts can be considered as finite beams for which the apex is not a point. They can also be seen as the volume of space defined by the set of rays between two objects.

6.1 Shaft culling

Haines and Wallace [HW91] have developed shaft culling in a global illumination context to speed up form factor computation using ray-casting. They define a shaft between two objects (or patches of the scene) as the convex hull of their bounding box (see Fig. 5.18).

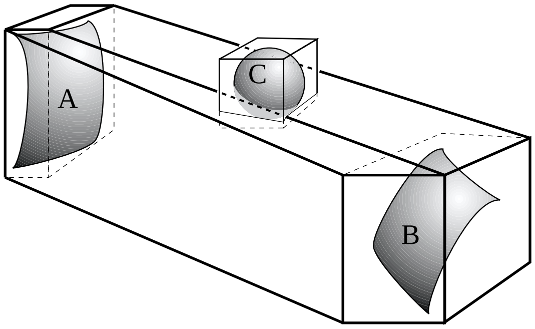


Figure 5.18: Shaft culling. The shaft between *A* and *B* is defined as the convex hull of the union of their bounding boxes. Object *C* intersects the shaft, it may thus cause occlusion between *A* and *B*.

They have developed an efficient construction of approximate shafts which takes advantage of the axis aligned bounding boxes. The test of an object against a shaft is also optimized for bounding boxes.

Similar methods have been independently devised by Zhang [Zha91] and Campbell [Cam91].

Marks *et al* [MWCF90], Campbell [Cam91] and Drettakis and Sillion [DS97] have derived hierarchical versions of shaft culling. The hierarchy of shafts is implicitly defined by a hierarchy on the objects. This hierarchy of shaft can also be seen as a hierarchy in line-space [DS97]. Brière and Poulin [BP96] also use a hierarchy of shafts or tubes to accelerate incremental updates in ray tracing.

6.2 Use of a dual space

Zao and Dobkin [ZD93] use shaft culling between pairs of triangles. They speed up the computation by the use of a multidimensional dual space. They decompose the shaft between a pair of triangles into tetrahedra and derive the conditions for another triangle to intersect a tetrahedron. These conditions are linear inequalities depending on the coordinates of the triangle.

They use multidimensional spaces depending on the coordinates of the triangles to speed up these tests. The queries in these spaces are optimized using binary trees (kd-trees in practice).

6.3 Occlusion culling from a volume

Cohen-Or and his co-authors [COFHZ98, COZ98] compute *potentially visible sets* from viewing cells. That is, the part of the scene where the viewer is allowed (the viewing space in short) is subdivided into cells from which the set of objects which may be visible is computed. This method can thus be seen as a viewpoint space method, but the core of the computation is based on the shaft philosophy.

Their method detects if a convex occluder occludes an object from a given cell. If convex polygonal objects are considered, it is sufficient to test if all rays between pairs of vertices are blocked by the occluder. The test is early terminated as soon as a non-blocked ray is found. It is in fact sufficient to test only silhouette rays (a ray between two point is a silhouette ray if each point is on the silhouette as seen from the other).

The drawback of this method is that it can not treat the occlusion caused by many blockers. The amount of storage required by the potentially visible set information is also a critical issue, as well as the cost of ray-casting.

7 Visibility propagation through portals

As already introduced, architectural scenes are organized into rooms, and inter-room visibility occurs only along openings named *portals*. This makes them particularly suitable for visibility preprocessing. Airey [Air90] and Teller [Tel92b, TS91] decompose a building into cells (roughly representing rooms) and precompute *Potentially Visible Sets* for each set. These are superset of objects visible from the cell which will then typically be sent to a z-buffer in a walkthrough application (see below).

7.1 Visibility computation

We describe here the methods proposed by Teller [Tel92b]. An adjacency graph is built connecting cells sharing a portal. Visibility is then propagated from a cell to neighbouring cells through portal sequences in a depth-first manner. Consider the situation illustrated in Fig. 5.19(a). Cell B is visible from cell A through the sequence of portals $p_1 p_2$. Cell C is neighbour of B in the adjacency graph, its visibility from A is thus tested. A sightline stabbing the portals p_1, p_2 and p_3 is searched (see Fig. 5.19(b)). A *stab-tree* is built which encodes the sequences of portals.

If the scene is projected on a floorplan, this stabbing problem reduces to find a stabber for a set of segments and can be solved using linear programming (see [Tel92b, TS91]).

If rectangular axis-aligned portals are considered in 3D, Teller [Tel92b] shows that the problem can be solved by projecting it in 2D along the three axis directions.

If arbitrary oriented portals are computed, he proposes to compute a conservative approximation to the visible region [Tel92b, TH93]. As each portal is added to the sequence, the EV events bounding the visibility

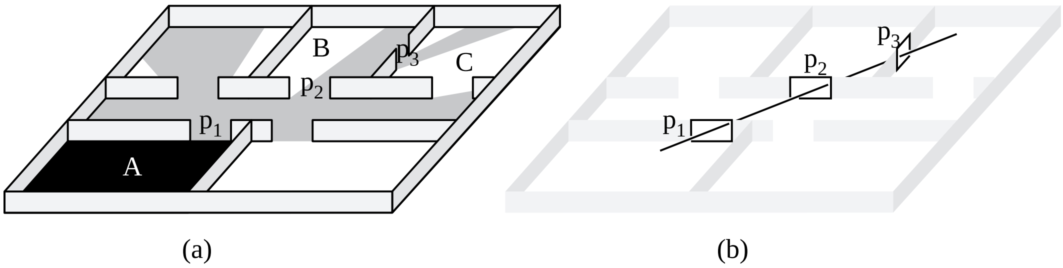


Figure 5.19: Visibility computations in architectural environments. (a) In grey: part of the scene visible from the black cell. (b) A stabbing line (or sightline) through a sequence of portals.

region are updated. These EV events correspond to separating planes between the portals. For each edge of the sequence of portals, only the extremal event is considered. The process is illustrated Fig. 5.20. It is a conservative approximation because EEE boundaries are not considered.

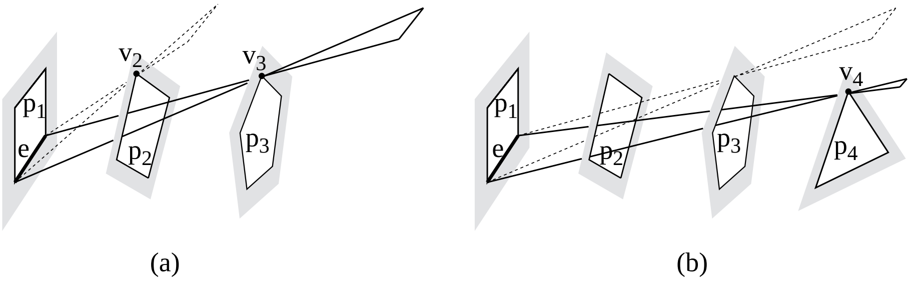


Figure 5.20: Conservative visibility propagation through arbitrary portals. (a) The separating plane considered for e is generated by v_3 because it lies below the one generated by v_2. (b) As a new portal is added to the sequence, the separating plane is updated with the same criterion.

If the visibility region is found to be empty, the new cell is not visible from the current cell. Otherwise, objects inside the cell are tested for visibility against the boundary of the visibility region as in a shaft method.

Airey [Air90] also proposes an approximate scheme where visibility between portals is approximated by casting a certain number of rays (see section 4 of chapter 8 for the approaches involving sampling with rays). See also the work by Yagel and Ray [YR96] who describe similar ideas in 2D.

The portal sequence can be seen as a sort of infinite shaft. We will also study it as the set of lines going through the portals in section 3.3 of chapter 8.

7.2 Applications

The primary focus of these potentially visible sets methods was the use in walkthrough systems. Examples can be found in both Airey [ARB90] and Teller's thesis [TS91, Tel92b]. Teller also uses an online visibility computation which restricts the visible region to the current viewpoint. The stab-tree is used to speed up a beam-like computation.

Funkhouser *et al.* [FS93] have extended Teller's system to use other rendering acceleration techniques such as mesh simplification in a real time context to obtain a constant framerate. He and his co-authors [FST92, Fun96c] have also used the information provided by the potentially visible sets to efficiently load from the disk or from the network only the parts of the geometry which may become visible in the subsequent frames. It can also be used in a distributed virtual environment context to limit the network bandwidth to messages between clients who can see each other [Fun95].

These computations have also been applied to speed-up radiosity computations by limiting the calculation of light interactions between mutually visible objects [TH93, ARB90]. It also permits lighting simulations for scenes which cannot fit into memory [TFFH94, Fun96b].

CHAPTER 6

Image-Space

L'art de peindre n'est que l'art d'exprimer l'invisible par le visible

Eugène FROMENTIN

MOST OF the image-space methods we present are based on a discretisation of an image. They often take advantage of the specialised hardware present in most of today's computers, which makes them simple to implement and very robust. Sampling rate and aliasing are however often the critical issues. We first present some methods which detect occlusions using projections on a sphere or on planes. Section 1 deals with the use of the z-buffer hardware to speed-up visibility computation. We then survey extensions of the z-buffer to perform occlusion-culling. Section 4 presents the use of a z-buffer orthogonal to the view for occlusion-culling for terrain-like scenes. Section 5 presents epipolar geometry and its use to perform view-warping without depth comparison. Section 6 discusses the computation of soft shadow using convolution, while section 7 deals with shadow-coherence in image-space.

1 Projection methods

1.1 Shadow projection on a sphere

Bouknight and Kelly [BK70] propose an optimization to compute shadows during a scan-line process as presented in section 6 of chapter 4. Their method avoids the need to intersect the wedge defined by the current span and the light source with all polygons of the scene.

As a preprocess, the polygons of the scene are projected onto a sphere centered at the point light source. A polygon can cast shadows on another polygon only if their projections overlap. They use bounding-box tests to speed-up the process.

Slater [Sla92] proposes a similar scheme to optimize the classification of polygons in shadow volume BSPs. He uses a discretized version of a cube centered on the source. Each *tile* (pixel) of the cube stores the polygon which project on it. This speeds up the determination of overlapping polygons on the cube. This shadow tiling is very similar to the light-buffer and to the hemicube which we will present in section 2.

1.2 Area light sources

Chrysanthou and Slater [CS97] have extended this technique to handle area light sources. In the methods presented above, the size of the sphere or cube does not matter. This is not the case of the extended method: a cube is taken which encloses the scene.

For each polygon, the projection used for point light sources becomes the intersection of its *penumbra volume* with the cube. The polygons with which it interacts are those which project on the same tiles.

1.3 Extended projections

The extended projection method proposed in chapter 5 of [Dur99] can be seen as an extension of the latter technique to perform offline occlusion culling from a volumetric cell (it can also be seen as an extension of Greene's hierarchical z-buffer surveyed in section 3). The occluders and occludees are projected onto a projection plane using *extended projection operators*. The extended projection of an occluder is the intersection of its views from all the viewpoints inside the cell. The extended projection of an occludee is the union of its views (similar to the penumbra used by Chrysanthou *et al.*).

If the extended projection of an occludee is in the cumulative extended projection of some occluders (and if it lies behind them), then it is ensured that it is hidden from any point inside the cell. This method handles *occluder fusion*.

2 Advanced z-buffer techniques

The versatility and robustness of the z-buffer together with efficient hardware implementations have inspired many visibility computation and acceleration schemes[1]. The use of the frame-buffer as a computational model has been formalized by Fournier and Fussel [FF88].

2.1 Shadow maps

As evoked in section 1.2 of chapter 2, hard shadow computation can be seen as the computation of the points which are visible from a point-light source. It is no surprise then that the z-buffer was used in this context.

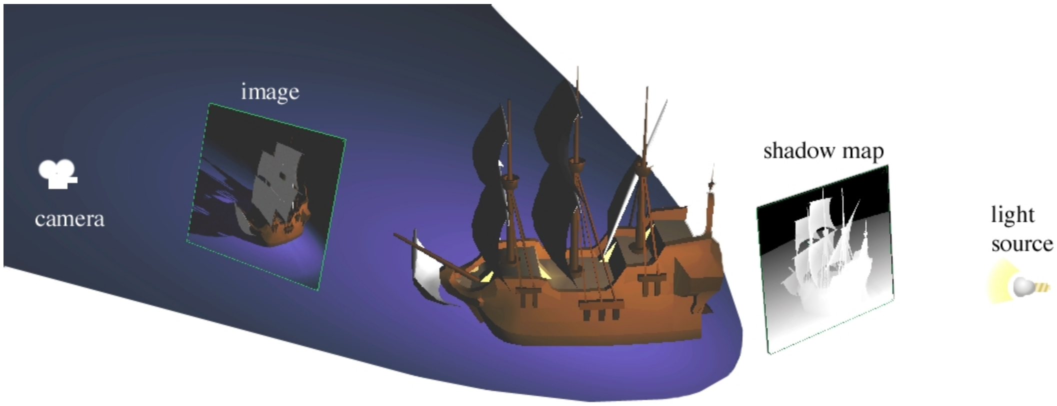


Figure 6.1: Shadow map principle. A shadow map is computed from the point of view of the light source (z-values are represented as grey levels). Then each point in the final image is tested for shadow occlusion by projecting it back in the shadow map (gallion model courtesy of Viewpoint Datalab).

A two pass method is used. An image is first computed from the source using a z-buffer. The *z* values of the closest points are stored in a depth map called *shadow map*. Then, as the final image is rendered, deciding

[1] Unexpected applications of the z-buffer have also been proposed such as 3D motion planning [LRDG90], Voronoi diagram computation [Hae90, ICK$^+$99] or collision detection [MOK95].

if a point is in shadow or not consists in projecting it back to the shadow map and comparing its distance to the stored z value (similarly to shadow rays, using the depth map as a query data-structure). The shadow map process is illustrated in Fig 6.1. Shadow maps were developed by Williams [Wil78] and have the advantage of being able to treat any geometry which can be handled by a z-buffer. Discussions of improvements can be found in [Gra92, Woo92].

The main drawback of shadow masks is aliasing. Standard filtering can not be applied, because averaging depth values makes no sense in this context. This problem was addressed by Reeves *et al.* [RSC87]. Averaging the depth values of the neighbouring pixels in the shadow map before performing the depth comparison would make no sense. They thus first compare the depth value with that of the neighbouring pixels, then they compute the average of the binary results. Had-oc soft shadows are obtained with this filtering, but the size of the penumbra is arbitrary and constant. See also section 6 for soft computation using an image-space shadow-map.

Soft shadow effects can be also achieved by sampling an extended light source with point light sources and averaging the contributions [HA90, HH97, Kel97]. See also [Zat93] for a use of shadow maps for high quality shadows in radiosity lighting simulation.

Shadow maps now seem to predominate in production. Ray tracing and shadow rays are used only when the artifacts caused by shadow maps are too noticeable. A hardware implementation of shadow maps is now available on some machines which allow the comparison of a texture value with a texture coordinate [SKvW $^+$92][2].

Zhang [Zha98a] has proposed an inverse scheme in which the pixels of the shadow map are projected in the image. His approach consists in warping the view from the light source into the final view using the view warping technique presented in section 1.7 of chapter 2. This is similar in spirit to Atherton and Weiler's method presented in section 2.1 of chapter 4: the view from the source is added to the scene database.

2.2 Ray-tracing optimization using item buffers

A z-buffer can be used to speed up ray-tracing computations. Weghorst *et al.* [WHG84] use a z-buffer from the viewpoint to speed up the computation of primary rays. An identifier of the objects is stored for each pixel (for example each object is assigned a unique color) in a so called *item buffer*. Then for each pixel, the primary ray is intersected only with the corresponding object. See also [Sun92].

Haines and Greenberg [HG86] propose a similar scheme for shadow rays. They place a *light buffer* centered on each point light source. It consists of 6 item buffers forming a cube (Fig. 6.2(a)). The objects of the scene are projected onto this buffer, but no depth test is performed, all objects projecting on a pixel are stored. Object lists are sorted according to their distance to the point light source. Shadow rays are then intersected only with the corresponding objects, starting with the closest to the source.

Poulin and Amanatides [PA91] have extended the light-buffer to linear light sources. This latter method is a first step towards line-space acceleration techniques that we present in section 1.4 of chapter 8, since it precomputes all objects intersected by the rays emanating from the light source.

Salesin and Stolfi [SS89, SS90] have extended the item buffer concept for ray-tracing acceleration. Their *ZZ-buffer* performs anti-aliasing through the use of an A-buffer like scheme. They detect completely covered pixels, avoiding the need for a subsampling of that pixel. They also sort the objects projecting on a non - simple pixel by their depth intervals. The ray-object intersection can thus be terminated earlier as soon as an intersection is found.

ZZ buffers can be used for primary rays and shadow rays. Depth of field and penumbra effects can also be obtained with a slightly modified ZZ-buffer.

In a commercial products such as Maya from Alias Wavefront [May99], an A-buffer and a ray-tracer are combined. The A-buffer is used to determine the visible objects, and ray-tracing is used only for pixels where high quality refraction or reflection is required, or if the shadow maps cause too many artifacts.

[2] A shadow map is computed from the point light source and copied into texture memory. The texture coordinate matrix is set to the perspective matrix from the light source. The initial u,v,w texture coordinate of a vertex are set to its 3D coordinates. After transformation, w represents the distance to the light source. It is compared against the texture value at u,v, which encodes the depth of the closest object. The key feature is the possibility to draw a pixel only if the value of w is smaller than the texture value at u,v.See [MBGN98] section 9.4.3. for implementation details.

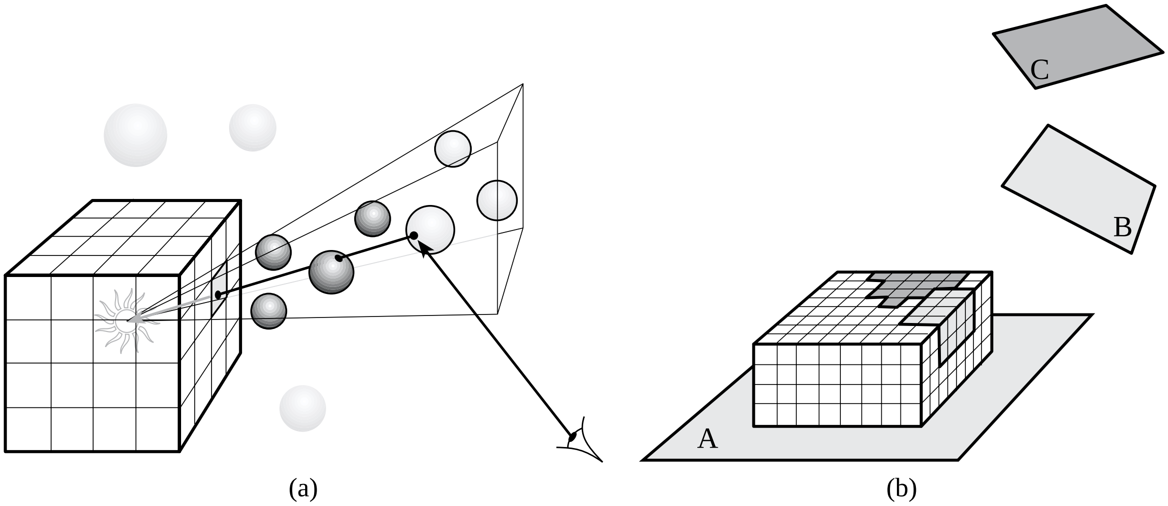


Figure 6.2: (a) Light buffer. (b) Form factor computation using the hemicube. Five z-buffers are placed around the center of patch *A*. All form factors between *A* and the other patches are evaluated simultaneously, and occlusion of *C* by *B* is taken into account.

2.3 The hemicube

Recall that *form factors* are used in radiosity lighting simulations to model the proportion of light leaving a patch which arrives at another. The first method developed to estimate visibility for form factor computations was the *hemicube* which uses five item-buffer images from the center of a patch as shown in Fig. 6.2(b). The form factor between one patch and all the others is evaluated simultaneously by counting the number of pixels covered by each patch.

The hemicube was introduced by Cohen *et al.* [CG85] and has long been the standard method for radiosity computations. However, as for all item buffer methods, sampling and aliasing problems are its main drawbacks. In section 2.2 of chapter 4 and section 4 of chapter 8 we present some solutions to these problems.

Sillion and Puech [SP89] have proposed an alternative to the hemicube which uses only one plane parallel the patch (the plane is however not uniformly sampled: A Warnock subdivision scheme is used.

Pietrek [Pie93] describe an anti-aliased version of the hemicube using a heuristic based on the variation between a pixel and its neighbours. See also [Mey90, BRW89]. Alonso and Holzschuch [AH97] present similar ideas as well as a deep discussion of the efficient access to the graphics hardware resources.

2.4 Sound occlusion and non-binary visibility

The wavelengths involved in sound propagation make it unrealistic to neglect diffraction phenomena. Simple binary visibility computed using ray-object intersection is far from accurate.

Tsingos and Gascuel [TG97a] use *Fresnel ellipsoids* and the graphics hardware to compute semi-quantitative visibility values between a sound source and a microphone. Sound does not propagate through lines; Fresnel ellipsoids describe the region of space in which most of the sound propagation occurs. Their size depends on the sound frequency considered. Sound attenuation can be modeled as the amount of occluders present in the Fresnel ellipsoid. They use the graphics hardware to compute a view from the microphone in the direction of the source, and count the number of occluded pixels.

They also use such a view to compute diffraction patterns on an extended receiver such as a plane [TG97b]. One view is computed from the source, and then for each point on the receiver, and integral is computed using the z values of the view. The contribution of each pixel to diffraction is then evaluated (see Fig. 6.3 for an example).

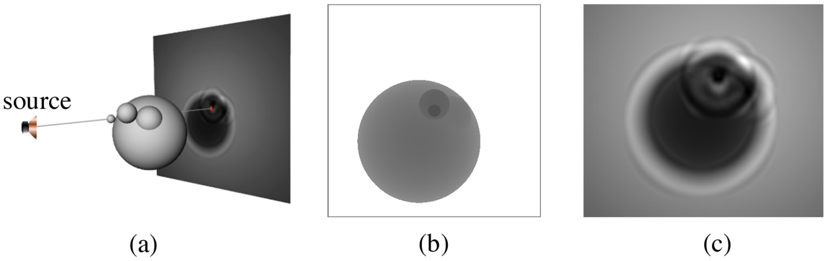


Figure 6.3: Non binary visibility for sound propagation. The diffraction by the spheres of the sound emitted by the source causes the diffraction pattern on the plane. (a) Geometry of the scene. (b) z-buffer from the source. (c) Close up of the diffraction pattern of the plane. (Courtesy of Nicolas Tsingos, iMAGIS-GRAVIR).

3 Hierarchical z-buffer

The z-buffer is simple and robust, however it has linear cost in the number of objects. With the ever increasing size of scenes to display, *occlusion culling* techniques have been developed to avoid the cost incurred by objects which are not visible.

Greene *et al.* [GKM93, Gre96] propose a hierarchical version of the z-buffer to quickly reject parts of the scene which are hidden. The scene is partitioned to an octree, and cells of the octree are rendered from front to back (the reverse of the original *painter algorithm*, see *e.g.* [FvDFH90, Rog97] or section 4 of chapter 4) to be able to detect the occlusion of back objects by frontmost ones. Before it is rendered, each cell of the octree is tested for occlusion against the current z values. If the cell is occluded, it is rejected, otherwise its children are treated recursively.

The z-buffer is organised in a pyramid to avoid to test all the pixels of the cell projection. Fig. 6.4 shows the principle of the hierarchical z-buffer.

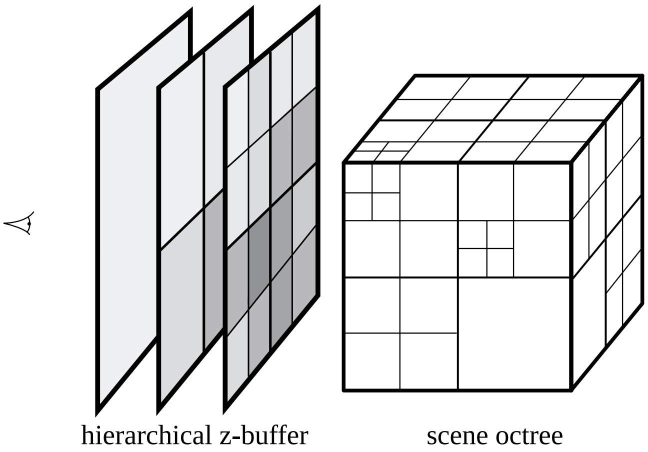


Figure 6.4: Hierarchical z-buffer.

The hierarchical z-buffer however requires many z-value queries to test the projection of cells and the maintenance of the z-pyramid; this can not be performed efficiently on today's graphics hardware. Zhang *et al.* [ZMHH97, Zha98b] have presented a two pass version of the hierarchical z-buffer which they have successfully implemented using available graphics hardware. They first render a subset of close and big objects called occluders, then read the frame buffer and build a so-called *hierarchical occlusion map* against which they test the bounding boxes of the objects of the scene. This method has been integrated in a massive model rendering system system [ACW$^+$99] in combination with geometric simplification and image-based acceleration techniques.

The strength of these methods is that they consider general occluders and handle *occluder fusion*, *i.e.* the occlusion by a combination of different objects.

The library Open GL Optimizer from Silicon Graphics proposes a form of screen space occlusion culling which seems similar to that described by Zhang *et al*. Some authors [BMT98] also propose a modification to the current graphics hardware to have access to z-test information for efficient occlusion culling.

4 Occluder shadow footprints

Many 3D scenes have in fact only two and a half dimensions. Such a scene is called a *terrain*, *i.e.*, a function $z = f(x,y)$. Wonka and Schmalstieg [WS99] exploit this characteristic to compute occlusions with respect to a point using a z-buffer with a top view of a scene.

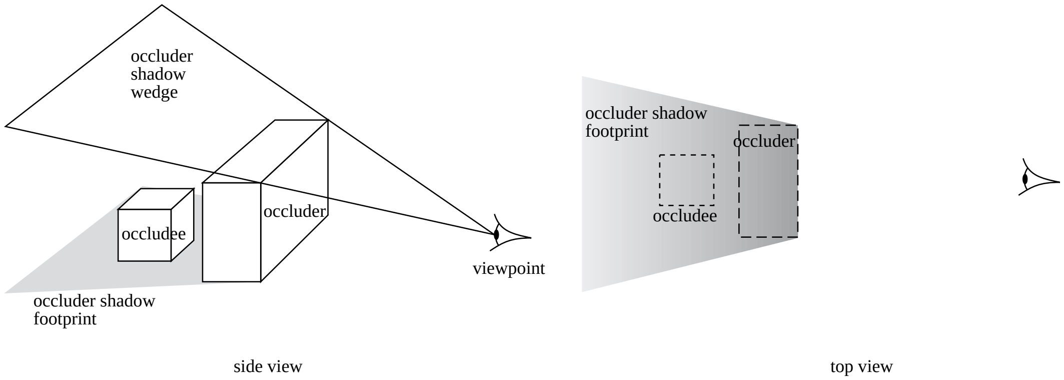


Figure 6.5: Occluder shadow footprints. A projection from above is used to detect occlusion. Objects are hidden if they are below the occluder shadows. The footprints (with height) of the occluded regions are rasterized using a z-buffer. Depth is represented as grey levels. Note the gradient in the footprint due to the slope of the wedge.

Consider the situation depicted in Fig. 6.5 (side view). They call the part of the scene hidden by the occluder from the viewpoint the *occluder shadow* (as if the viewpoint were a light source). This occluder shadow is delimited by wedges. The projection of such a wedge on the floor is called the footprint, and an occludee is hidden by the occluder if it lies on the shadow footprint and if it is below the edge.

The z-buffer is used to scan-convert and store the height of the shadow footprints, using an orthographic top view (see Fig. 6.5). An object is hidden if its projection from above is on a shadow footprint and if it is *below* the shadow wedges *i.e*, if it is occluded by the footprints in the top view.

5 Epipolar rendering

Epipolar geometry has been developed in computer vision for stereo matching (see *e.g.* [Fau93]). Assume that the geometry of two cameras is known. Consider a point A in the first image (see Fig. 6.6). The possible point of the 3D scene must lie on the line L_A going through A and viewpoint 1. The projection of the corresponding point of the scene on the second image is constrained by the epipolar geometry: it must be on line L'_A which is the projection of L_A on image 2. The search for a correspondence can thus be restricted from a 2D search over the entire image to a 1D search on the *epipolar line*.

Mc Millan and Bishop [MB95] have taken advantage of the epipolar geometry for view warping. Consider the warping from image 2 to image 1 (image 2 is the initial image, and we want to obtain image 1 by reprojecting the points of image 2). We want to decide which point(s) is reprojected on A. These are necessarily points on the epipolar line L'_A. However, many points may project on A; only the closest has to be displayed. This can be achieved without actual depth comparison, by warping the points of the epipolar line L'_A in the order shown by the thick arrow, that is, from the farthest to the closest. If more than one point projects on A, the closest will overwrite the others. See also section 1.5 of chapter 8 for a line-space use of epipolar geometry.

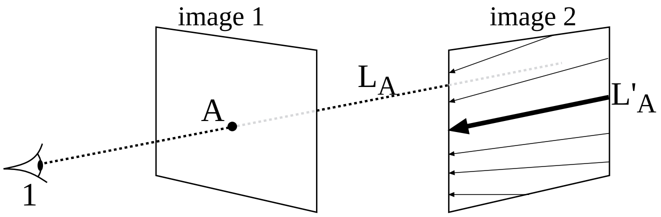


Figure 6.6: Epipolar geometry. L_A is the set of all points of the scene possibly projecting on A. L'_A is the projection on image 2. For a warping from image 2 to image 1, points of image 2 have to be reprojected to image 1 in the order depicted by the arrows for correct occlusion.

6 Soft shadows using convolution

Soler and Sillion [SS98a, Sol98] have developed efficient soft shadow computations based on the use of convolutions. Some of the ideas are also present in a paper by Max [Max91]. A simplification could be to see their method as a “wise” blurring of shadow maps depending on the shape of the light source.

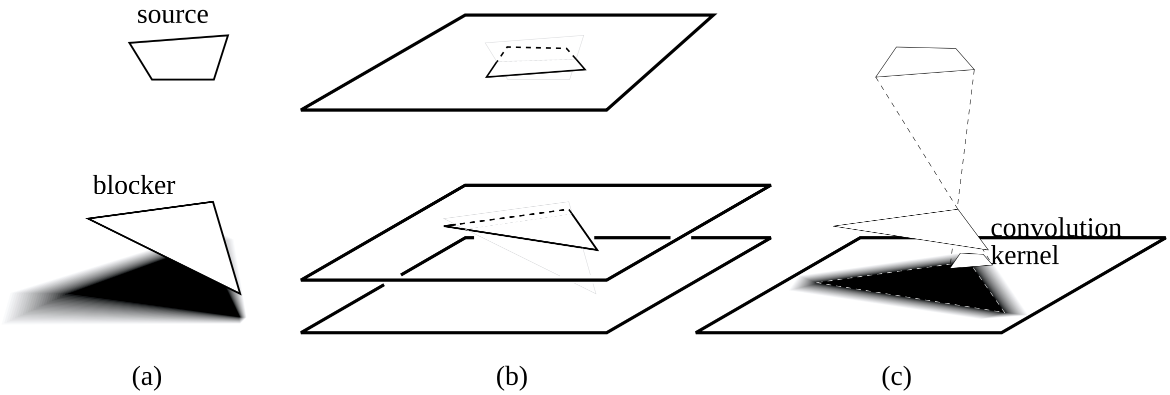


Figure 6.7: Soft shadows computation using convolution. (a) Geometry of the scene. (b) Projection on a parallel approximate geometry. (c) The shadow is the convolution of the projection of the blockers with the inverse image of the source.

Consider an extended light source, a receiver and some blockers as shown in Fig. 6.7(a). This geometry is first projected onto three parallel planes (Fig. 6.7(b)). The shadow computation for this approximate geometry is equivalent to a convolution: the projection of the blocker(s) is convolved with the inverse projection of the light source (see Fig. 6.7(c)). The shadow map obtained is then projected onto the receiver (this is not necessary in our figures since the receiver is parallel to the approximate geometry).

In the general case, the shadows obtained are not exact: the relative sizes of umbra and penumbra are not correct. They are however not constant if the receiver is not parallel to the approximate geometry. The results are very convincing (see Fig. 6.8).

For higher quality, the blockers can be grouped according to their distance to the source. A convolution is performed for each group of blockers. The results then have to be combined; Unfortunately the correlation between the occlusions of blockers belonging to different groups is lost (see also [Gra92] for a discussion of correlation problems for visibility and antialiasing).

This method has also been used in a global simulation system based on radiosity [SS98b].

7 Shadow coherence in image-space

Haines and Greenberg [HG86] propose a simple scheme to accelerate shadow computation in ray-tracing. Their *shadow cache* simply stores a pointer to the object which caused a shadow on the previous pixel. Because of

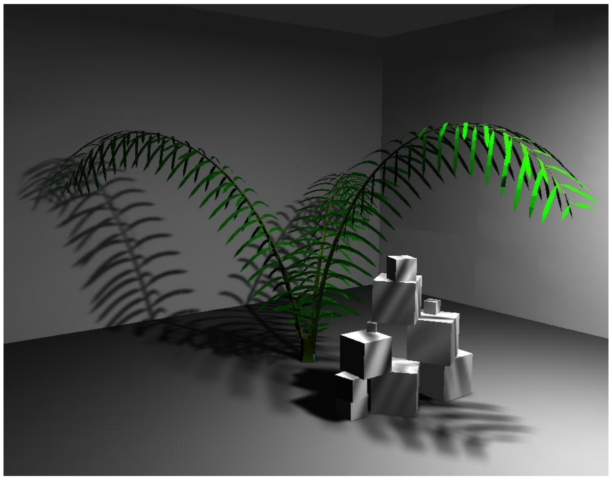


Figure 6.8: Soft shadows computed using convolutions (image courtesy of Cyril Soler, iMAGIS-GRAVIR)

coherence, it is very likely that this object will continue to cast a shadow on the following pixels.

Pearce and Jevans [PJ91] extend this idea to secondary shadow rays. Because of reflection and refraction, many shadow rays can be cast for each pixel. They thus store a tree of pointers to shadowing objects corresponding to the secondary ray-tree.

Worley [Wor97] pushes the idea a bit further for efficient soft shadow computation. He first computes simple hard shadows using one shadow-ray per pixel. He notes that pixels where shadow status changes are certainly in penumbra, and so are their neighbours. He thus “spreads” soft shadows, using more shadow rays for these pixels. The spreading operation stops when pixels in umbra or completely lit are encountered.

Hart *et al* [HDG99] perform a similar image-space floodfill to compute a blocker map: for each pixel, the objects casting shadows on the visible point are stored. They are determined using a low number of rays per pixel, but due to the image-space flood-fill the probability to miss blockers is very low. They then use an analytic clipping of the source by the blockers to compute the illumination of each pixel.

CHAPTER 7

Viewpoint-Space

> On ne voit bien qu'avec le cœur. L'essentiel est invisible pour les yeux.
>
> Antoine de Saint-Exupery, *Le Petit Prince*

VIEWPOINT-SPACE methods characterize viewpoints with respect to some visibility property. We first present the aspect graph which partitions viewpoint space according to the qualitative aspect of views. It is a fundamental visibility data-structure since it encodes all possible views of a scene. Section 2 presents some methods which are very similar to the aspect graph. Section 3 deals with the optimization of a viewpoint or set of viewpoints to satisfy some visibility criterion. Finally section 4 presents two methods which use visual events to determine the viewpoints at which visibility changes occur.

1 Aspect graph

As we have seen in section 2 of chapter 2 and Fig. 2.8 page 14, model-based object recognition requires a viewer-centered representation which encodes all the possible views of an object. This has led Koenderink and Van Doorn [Kv76, Kv79] to develop the *visual potential* of an object which is now more widely known as the *aspect graph* (other terminology are also used in the literature such as *view graph*, *characteristic views*, *principal views*, *viewing data*, *view classes* or *stable views*).

Aspect graph approaches consist in partitioning viewpoint space into cells where the view of an object are qualitatively invariant. The aspect graph is defined as follows:

- Each node represents a *general view* or *aspect* as seen form a connected cell of viewpoint space.

- Each arc represents a *visual event*, that is, a transition between two neighbouring general views.

The aspect graph is the dual graph of the partition of viewpoint space into cells of constant aspect. This partition is often named *viewing data* or *viewpoint space partition*. The terminology aspect graph and viewpoint space partition are often used interchangeably although they refer to dual concepts.

Even though all authors agree on the general definition, the actual meaning of *general view* and *visual event* varies. First approximate approaches have considered the set of visible features as defining a view. However for exact approaches the *image structure graph* has rapidly imposed itself. It is the graph formed by the occluding contour or visible edges of the object. This graph may be labeled with the features of the object.

It is important to understand that the definition of the aspect graph is very general and that any definition of the viewing space and aspect can be exchanged. This makes the aspect graph concept a very versatile tool as we will see in section 2.

Aspect graphs have inspired a vast amount of work and it is beyond the scope of this survey to review all the literature in this field. We refer the reader to the survey by Eggert *et al.* [EBD92] or to the articles we cite and the references therein. Approaches have usually been classified according to the viewpoint space used (perspective or orthographic) and by the class of objects considered. We will follow the latter, reviewing the methods devoted to polyhedra before those related to smooth objects. But first of all, we survey the approximate method which use a discretization of viewpoint space.

1.1 Approximate aspect graph

Early aspect graph approaches have used a quasi uniform tessellation of the viewing sphere for orthographic projection. It can be obtained through the subdivision of an initial icosahedron as shown by Fig. 7.1. Sample views are computed from the vertices of this tessellation (the typical number of sample views is 2000). They are then compared, and similar views are merged. Very often, the definition of the aspect is the set of visible features (face, edge, vertex) and not their adjacencies as it is usually the case for exact aspect graphs This approach is very popular because of its simplicity and robustness, which explains that it has been followed by many researchers *e.g.* [Goa83, FD84, HK85]. We will see that most of the recognition systems using aspect graphs which have been implemented use approximate aspect graphs.

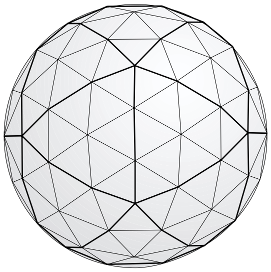


Figure 7.1: Quasi uniform subdivision of the viewing sphere starting with an icosahedron.

We will see in section 3.2 that this quasi uniform sampling scheme has also been applied for viewpoint optimization problems.

A similar approach has been developed for perspective viewpoint space using voxels [WF90].

The drawback of approximate approaches is that the sampling density is hard to set, and approximate approach may miss some important views, which has led some researchers to develop exact methods.

1.2 Convex polyhedra

In the case of convex polyhedra, the only visual events are caused by viewpoints tangent to faces. See Fig. 7.2 where the viewpoint partition and aspect graph of a cube are represented. For orthographic projection, the directions of faces generate 8 regions on the viewing sphere, while for perspective viewpoint space, the 6 faces of the cube induce 26 regions.

The computation of the visual events only is not sufficient. Their *arrangement* must be computed, that is, the decomposition of viewpoint space into cells, which implies the computation of the intersections between the events to obtain the segments of events which form the boundaries of the cells. Recall that the arrangement of n lines (or well-behaved curves) in 2D has $O(n^2)$ cells. In 3D the arrangement of n planes has complexity $O(n^3)$ in size [dBvKOS97, O'R94, Ede87, BY98].

The first algorithms to build the aspect graph of 3D objects have dealt with convex polyhedra under orthographic [PD86] and perspective [SB90, Wat88] projection.

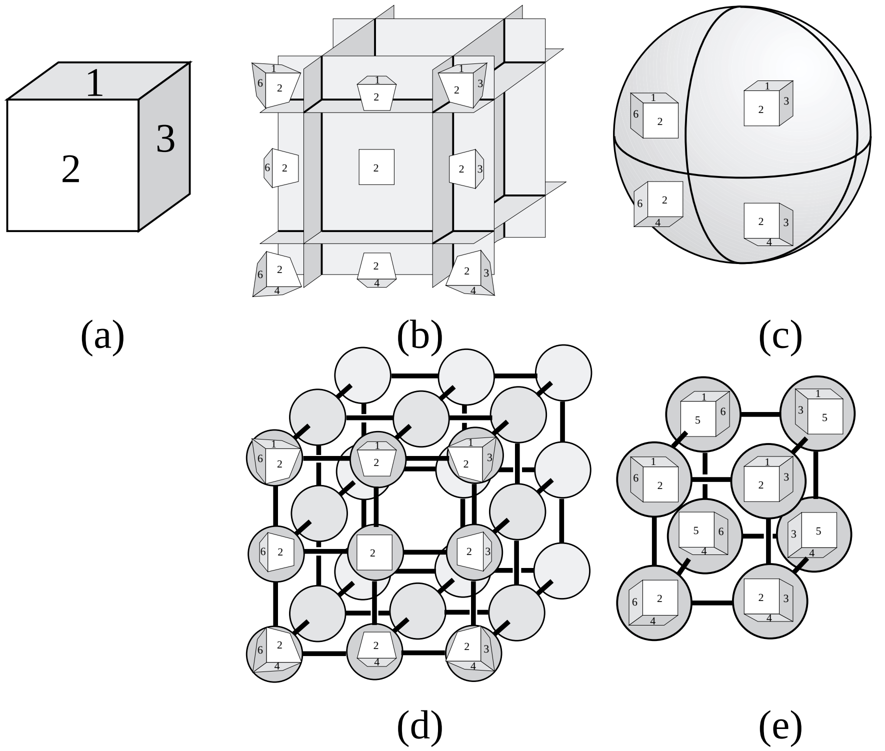


Figure 7.2: Aspect graph of a convex cube. (a) Initial cube with numbered faces. (b) and (c) Partition of the viewpoint space for perspective and orthographic projection with some representative aspects. (d) and (e) Corresponding aspect graphs. Some aspects are present in perspective projection but not in orthographic projection, for example when only two faces are visible. Note also that the cells of the perspective viewpoint space partition have infinite extent.

1.3 General polyhedra

General polyhedra are more involved because they generate edge-vertex and triple-edge events that we have presented in chapter 3. Since the number of triple-edge events can be as high as $O(n^3)$, the size of the aspect graph of a general polygon is $O(n^6)$ for orthographic projection (since the viewing sphere is two dimensional), and $O(n^9)$ for perspective projection for which viewpoint space is three-dimensional. However these bounds may be very pessimistic. Unfortunately the lack of available data impede a realistic analysis of the actual complexity. Note also that we do not count here the size of the representative views of aspects, which can be $O(n^2)$ each, inducing a size $O(n^8)$ for the orthographic case and $O(n^{11})$ for the perspective case.

The cells of the aspect graph of a general polyhedron are not necessary convex. Partly because of the EEE events, but also because of the EV events. This is different from the 2D case where all cells are convex because in 2D visual events are line segments.

We detail here the algorithms proposed by Gigus and his co-authors [GM90, GCS91] to build the aspect graph of general polyhedra under orthographic projection.

In the first method [GM90], potential visual events are considered for each face, edge-vertex pair and triple of edges. At this step, occlusion is not taken into account: objects lying between the generators of the events are considered transparent. These potential events are projected on the viewing sphere, and the arrangement is built using a plane sweep.

However, some boundaries of the resulting partition may correspond to false visual event because of occlusion. For example, an object may lie between the edge and vertex of an EV event as shown in Fig. 7.3. Each segment of cell boundary (that is, each portion of visual event) has to be tested for occlusion. False segment are discarded, and the cells are merged.

Gigus Canny and Seidel [GCS91] propose to cope with the problem of false events before the arrangement is constructed. They compute the intersection of all the event with the object in object space as shown in Fig.

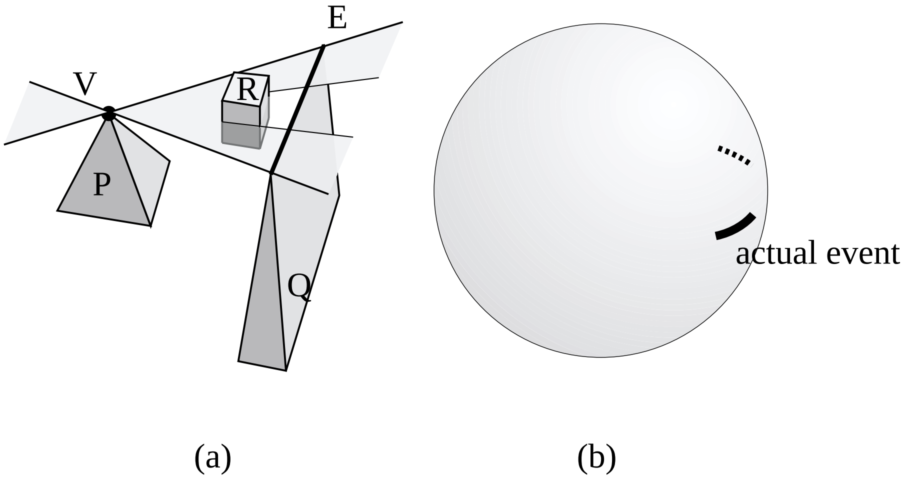


Figure 7.3: False event ("transparent" event). Object R occludes vertex V from edge E, thus only a portion of the potential visual event corresponds to an actual visual event. (a) In object space. (b) In orthographic viewpoint space.

7.3(a), and only the unoccluded portion is used for the construction of the arrangement.

They also propose to store and compute the representative view efficiently. They store only one aspect for an arbitrary seed cell. Then all other views can be retrieved by walking along the aspect graph and updating this initial view at each visual event.

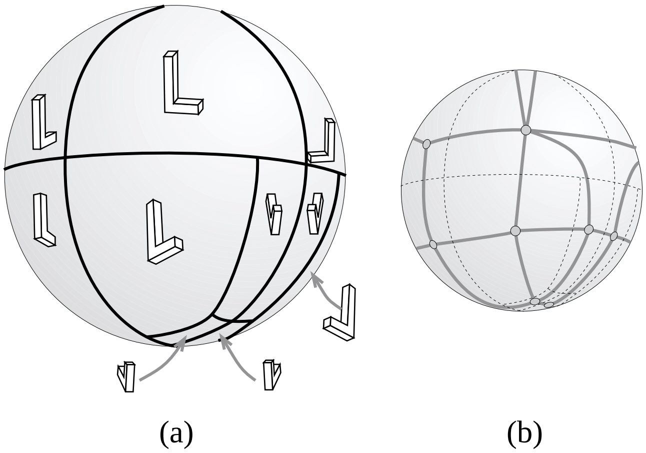


Figure 7.4: Aspect graph of a L-shaped polyhedron under orthographic projection (adapted from [GM90]). (a) Partition of the viewing sphere and representative views. (b) Aspect graph.

These algorithms have however not been implemented to our knowledge. Fig. 7.4 shows the partition of the viewing sphere and the aspect graph of a L-shaped polyhedron under orthographic transform.

Similar construction algorithms have been proposed by Stewman and Bowyer [SB88] and Stewman [Ste91] who also deals with perspective projection.

We will see in section 1.1 of chapter 8 that Plantinga and Dyer [PD90] have proposed a method to build the aspect graph of general polyhedra which uses an intermediate line space data-structure to compute the visual events.

1.4 Curved objects

Methods to deal with curved objects were not developed till later. Seales and Dyer [SD92] have proposed the use of a polygonal approximation of curved objects with polyhedra, and have restricted the visual events to those involving the silhouette edges. For example, an edge-vertex event EV will be considered only if E is a silhouette edge from V (as this is the case in Fig. 3.3 page 26). This is one example of the versatility of the aspect graph definition: here the definition of the aspect depends only on the silhouette.

Kriegman and Ponce [KP90] and Eggert and Bowyer [EB90] have developed methods to compute aspect graphs of solids of revolution under orthographic projection, while Eggert [Egg91] also deals with perspective viewpoint space. Objects of revolution are easier to handle because of their rotational symmetry. The problem reduces to a great circle on the viewing sphere or to one plane going through the axis of rotation in perspective viewpoint space. The rest of the viewing data can then be deduced by rotational symmetry. Eggert *et al.* [EB90, Egg91] report an implementation of their method.

The case of general curved object requires the use of the catalogue of singularities as proposed by Callahan and Weiss [CW85]; they however developed no algorithm.

Petitjean and his co-authors [PPK92, Pet92] have presented an algorithm to compute the aspect graph of smooth objects bounded by arbitrary smooth algebraic surface under orthographic projection. They use the catalogue of singularities of Kergosien [Ker81]. There approach is similar to that of Gigus and Malik [GM90]. They first trace the visual events of the "transparent" object (occlusion is not taken into account) to build a partition of the viewing sphere. They then have to discard the false (also called occluded) events and merge the corresponding cells. Occlusion is tested using ray-casting at the center of the boundary. To trace the visual event, they derive their equation using a computer algebra system and powerful numerical techniques. The degree of the involved algebraic systems is very large, reaching millions for an object described by an equation of degree 10. This algorithm has nevertheless been implemented and an example of result is shown in Fig. 7.5.

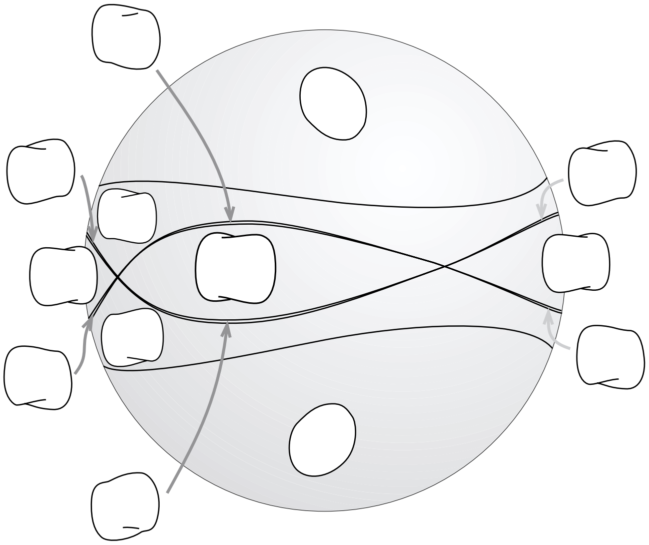


Figure 7.5: Partition of orthographic viewpoint space for a dimple object with representative aspects. (adapted from [PPK92]).

Similar methods have been developed by Sripradisvarakul and Jain [SJ89], Ponce and Kriegman [PK90] while Rieger [Rie92, Rie93] proposes the use of symbolic computation and cylindrical algebraic decomposition [Col75] (for a good introduction to algebraic decomposition see the book by Latombe [Lat91] p. 226).

Chen and Freeman [CF91b] have proposed a method to handle quadric surfaces under perspective projection. They use a sequence of growing concentric spheres centered on the object. They trace the visual events on each sphere and compute for which radius the aspects change.

Finally PetitJean has studied the enumerative properties of aspect graphs of smooth and piecewise smooth objects [Pet95, Pet96]. In particular, he gives bounds on the number of topologically distinct views of an object using involved mathematical tools.

1.5 Use of the aspect graph

The motivation of aspect graph research was model-based object recognition. The aspect graph provides informations on all the possible views of an object. The use of this information to recognise an object and its pose are however far from straightforward, one reason being the huge number of views. Once the view of an object has been acquired from a camera and its features extracted, those features can not be compared to all possible views of all objects in a database: indexing schemes are required. A popular criterion is the number of visible features (face, edge, vertex) [ESB95].

The aspect graph is then often used to build offline a *strategy tree* [HH89]or an *interpretation tree* [Mun95]. At each node of an interpretation tree corresponds a choice of correspondence, which then recursively leads to a restricted set of possible interpretation. For example if at a node of the tree we suppose that a feature of the image corresponds to a given feature *A* of a model, this may exclude the possibility of another feature *B* to be present because feature *A* and *B* are never visible together.

The information of the viewing space partition can then be used during pose estimation to restrict the possible set of viewpoint [Ike87, ESB95]. If the observation is ambiguous, Hutchinson and Kak [HK89] and Gremban and Ikeuchi [GI87] also use the information encoded in the aspect graph to derive a new relevant viewpoint from which the object and pose can be discriminated.

Dickinson *et al.* [DPR92] have used the aspect for object composed of elementary objects which they call *geons*. They use an aspect graph for each geon and then use structural information on the assembly of geons to recognise the object.

However the aspect graph has not yet really imposed itself for object recognition. The reasons seem to be the difficulty of robust implementation of exact methods, huge size of the data-structure and the lack of obvious and efficient indexing scheme. One major drawback of the exact aspect graphs is that they capture all the possible views, whatever their likelihood or significance. The need of a notion "importance" or *scale* of the features is critical, which we will discuss in section 1 of chapter 9.

For a good discussion of the pros and cons of the aspect graph, see the report by Faugeras *et al.* [FMA $^+$92].

Applications of the aspect graph for rapid view computation have also been proposed since all possible views have been precomputed [PDS90, Pla93]. However, the only implementation reported restricted the viewpoint movement to a rotation around one axis.

More recently Gu and his coauthors [GGH$^+$99] have developed a data-structure which they call a *silhouette tree* which is in fact an aspect graph for which the aspect is defined only by the exterior silhouette. It is built using a sampling and merging approach on the viewing sphere. It is used to obtain images with a very fine silhouette even if a very simplified version of the object is rendered.

Pellegrini [Pel99] has also used a decomposition of the space of direction similar to the aspect graph to compute the form factor between two unoccluded triangles. The sphere S^2 is decomposed into regions where the projection of the two triangles has the same topology. This allows an efficient integration because no discontinuity of the integration kernel occur in these regions.

A somehow related issue is the choice of a good viewpoint for the view of a 3D graph. Visual intersections should be avoided. These in fact correspond to EV or EEE events. Some authors [BGRT95, HW98, EHW97] thus propose some methods which avoid points of the viewing sphere where such events project.

2 Other viewpoint-space partitioning methods

The following methods exhibit a typical aspect graph philosophy even though they use a different terminology. They subdivide the space of viewpoints into cells where a view is qualitatively invariant.

2.1 Robot Localisation

Deducing the position of a mobile robot from a view is exactly the same problem as determining the pose of an object. The differences being that a range sensor is usually used and that the problem is mostly two dimensional since mobile robots are usually naturally constrained on a plane.

Methods have thus been proposed which subdivide the plane into cells where the set of visible walls is constant [GMR95, SON96, TA96]. See Fig. 7.6. Visual events occur when the viewpoint is aligned with a wall segments or along a line going through two vertices. Indexing is usually done using the number of visible walls.

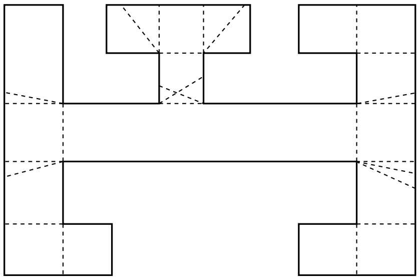


Figure 7.6: Robot self-localization. Partition of a scene into cells of structurally invariant views by visual events (dashed).

Guibas and his co-authors [GMR95] also propose to index the aspects in a multidimensional space. To summarize, they associate to a view with m visible vertices a vector of $2m$ dimensions depending on the coordinates of the vertices. They then use standard multidimensional search methods [dBvKOS97].

2.2 Visibility based pursuit-evasion

The problem of pursuit-evasion presented in section 3 and Fig. 2.14 page 18 can also be solved using an aspect-graph-like structure. Remember that the robot has to "clean" a scene by checking if an intruder is present. "Contaminated" regions are those where the intruder can hide. We present here the solution developed by LaValle *et al.* [LLG$^+$97, GLL$^+$97, GLLL98].

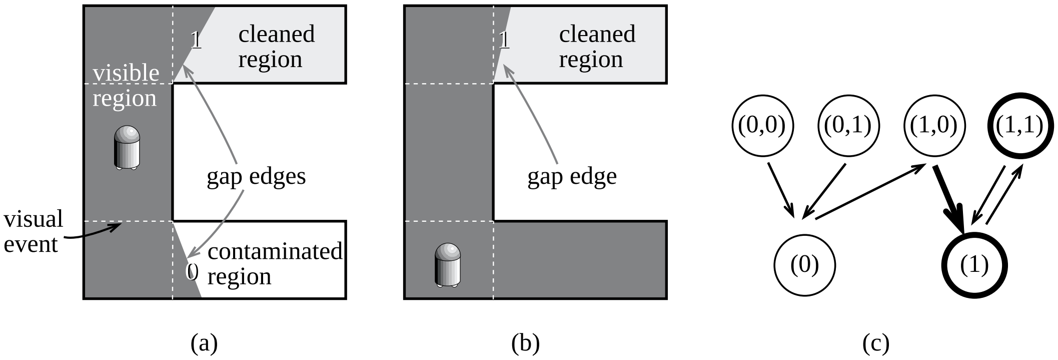


Figure 7.7: Pursuit-Evasion strategy. (a) The contaminated region can be cleaned only if the visual event is crossed. The status of the neighbouring regions is coded on the gap edges. (b) The robot has moved to a second cell, cleaning a region. (c) Part of the graph of possible states (upper node correspond to cell in (a) while lower nodes correspond to the cell in (b)). In thick we represent the goal states and the move from (a) to (b).

Consider the situation in Fig. 7.7(a). The view from the robot is in dark grey. The contaminated region can be cleaned only when the indicated visual event is crossed as in Fig. 7.7(b).

The scene is partitioned by the visibility event with the same partition as for robot localization (see Fig. 7.6). For each cell of the partition, the structure of the view polygon is invariant, and in particular the *gap edges* (edges of the view which are not on the boundary of the scene). The status of the neighbouring regions is coded on these gap edges: 0 indicates a contaminated region while 1 indicates a cleaned one.

The state of the robot is thus coded by its current cell and the status of the corresponding gap edges. In Fig 7.7(a) the robot status is $(1,0)$, while in (b) it is (1). Solving the pursuit problem consists in finding the succession of states of the robot which end at a state where all gap edges are at 1. A graph is created with one node for each state (that means 2^m states for a cell with m edges). Edges of the graph correspond to possible transition. A transition is possible only to neighbouring cells, but not to all corresponding states. Fig. 7.7 represents a portion of this graph.

The solution is then computed using a standard Dijkstra search. See Fig. 2.14 page 18 for an example. Similar methods have also been proposed for curved environments [LH99].

2.3 Discontinuity meshing with backprojections

We now turn to the problem of soft shadow computation in polygonal environments. Recall that the penumbra region corresponds to zones where only a part of an extended light source is visible. Complete discontinuity meshing subdivides the scene polygons into regions where the topology of the visible part of the source is constant. In this regions the illumination varies smoothly, and at the region boundary there is a C^2 discontinuity.

Moreover a data-structure called *backprojection* encodes the topology of the visible part of the source as represented in Fig. 7.8(b) and 7.9(b). Discontinuity meshing is an aspect graph method where the aspect is defined by the visible part of the source, and where viewpoint space is the polygons of the scene.

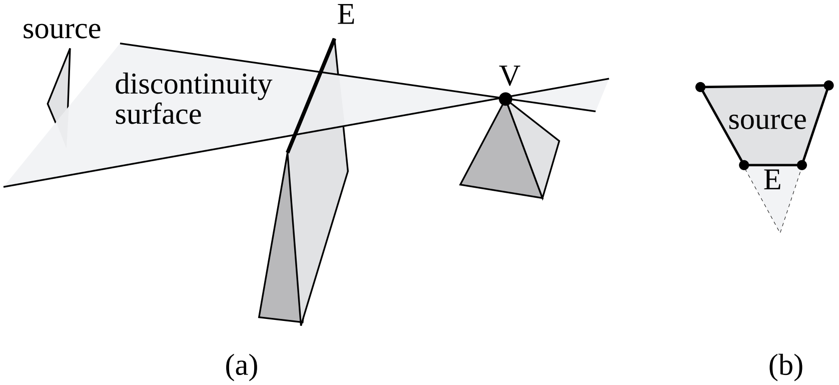


Figure 7.8: Complete discontinuity meshing with backprojections. (a) Example of an EV event intersecting the source. (b) In thick backprojection from V (structure of the visible part of the source)

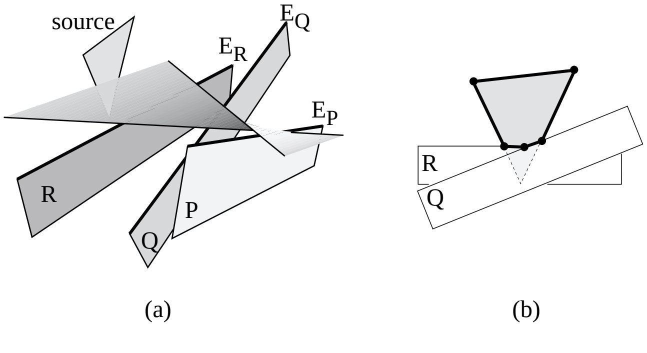


Figure 7.9: Discontinuity meshing. (a) Example of an EEE event intersecting the source. (b) In thick backprojection from a point on E_P (structure of the visible part of the source)

Indeed the method developed and implemented by Drettakis and Fiume [DF94] is the equivalent of Gigus Canny and Seidel's algorithm [GCS91] presented in the previous section. Visual events are the EV and EEE event with one generator on the source or which intersect the source (Fig. 7.8(a) and 7.9(a)). An efficient space subdivision acceleration is used to speed up the enumeration of potential visual events. For each vertex generator V an extended pyramid is build with the light source, and only the generators lying inside this volume are considered. Space subdivision is used to accelerate this test. A similar scheme is used for edges. Space subdivision is also used to speed-up the discontinuity surface-object intersections. See Fig. 7.10 for an example of shadows and discontinuity mesh.

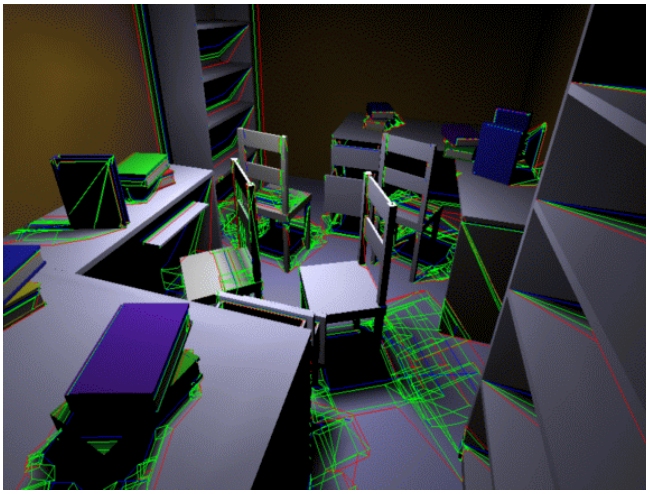


Figure 7.10: Complete discontinuity mesh of a 1000 polygons scene computed with Drettakis and Fiume's algorithm [DF94].

This method has been used for global illumination simulation using radiosity [DS96]. Both the mesh and form-factor problem are alleviated by this approach, since the backprojection allows for efficient point-to-area form factor computation (portion of the light leaving the light source arriving at a point). The experiments exhibited show that both the quality of the induced mesh and the precision of the form-factor computation are crucial for high quality shadow rendering.

2.4 Output-sensitive discontinuity meshing

Stewart and Ghali [SG94] have proposed an output-sensitive method to build a complete discontinuity mesh. They use a similar discontinuity surface-object intersection, but their enumeration of the discontinuity surfaces is different.

It is based on the fact that a vertex V can generate a visual event with an edge E only if E lies on the boundary of the visible part of the source as seen from V (see Fig. 7.8). A similar condition arises for EEE events: the two edges closest to the source must belong to the backprojection of some part of the third edge, and must be adjacent in this backprojection as shown in Fig. 7.9.

They use an update of the backprojections at visual events. They note that a visual event has effect only on the parts of scene which are farther from the source than its generators. They thus use a sweep with planes parallel to the source. Backprojections are propagated along the edges and vertices of the scene, with an update at each edge-visual event intersection.

Backprojection have however to be computed for scratch at each *peak vertex*, that is, for each polyhedron, the vertex which is closest to the source. Standard hidden surface removal is used.

The algorithm can be summarized as follows:

- Sort the vertices of the scene according to the distance to the source.
- At peak vertices compute a backprojection and propagate it to the beginning of the edges below.
- At each edge-visual event intersection update the backprojection.
- For each new backprojection cast (intersect) the generated visual event through the scene.

This algorithm has been implemented [SG94] and extended to handle degenerate configuration [GS96] which cause some C^1 discontinuities in the illumination function.

3 Viewpoint optimization

In this section we present methods which attempt to chose a viewpoint or a set of viewpoints to optimize the visibility of all or some of the features of a scene. The search is here exhaustive, all viewpoints (or a sampling) are tested. The following section will present some methods which alleviate the need to search the whole space of viewpoints. Some related results have already been presented in section 4.5 and 5.5 of chapter 5.

3.1 Art galleries

We present the most classical results on art gallery problems. The classic art gallery theorem is due to Chvátal [Chv75] but he exhibited a complex proof. We here present the proof by Fisk [Fis78] which is much simpler. We are given an art-gallery modeled by a simple (with no holes) 2D polygons.

Theorem: $\lfloor \frac{n}{3} \rfloor$ *stationary guards are always sufficient and occasionally necessary to guard a polygonal art gallery with n vertices.*

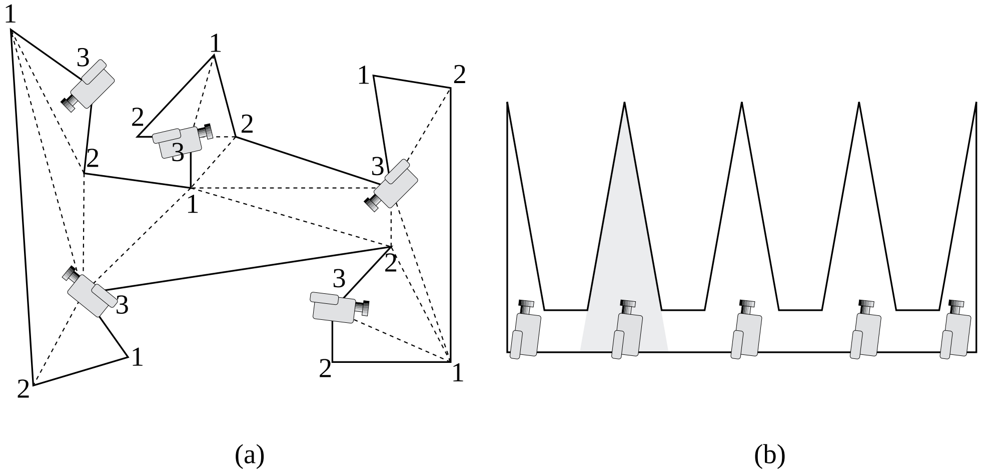


Figure 7.11: Art gallery. (a) The triangulation of a simple polygon is 3-colored with colors 1, 2 and 3. Color 3 is the less frequent color. Placing a guard at each vertex with color 3 permits to guard the polygon with less than $\lfloor \frac{n}{3} \rfloor$ guards. (b) Worst-case scene. To guard the second spike, a camera is needed in the grey region. Similar constraints for all the spikes thus impose the need of at least $\lfloor \frac{n}{3} \rfloor$ guards

The proof relies on the triangulation of the polygon with diagonals (see Fig. 7.11(a)). The vertices of such a triangulation can always be colored with 3 colors such that no two adjacent vertices share the same color (Fig. 7.11(a)). This implies that any triangle has one vertex of each color. Moreover, each vertex can guard its adjacent triangles.

Consider the color which colors the minimum number of vertices. The number of corresponding vertices is lower than $\lfloor \frac{n}{3} \rfloor$, and each triangle has such a vertex. Thus all triangles are guarded by this set of vertices. The lower bound can be shown with a scene like the one presented in Fig. 7.11(b).

Such a set of guards can be found in $O(n)$ time using a linear time triangulation algorithm by Chazelle [dBvKOS97]. The problem of finding the minimum number of guards has however been shown NP-hard by Aggarwal [Aga84] and Lee and Lin [LL86].

For other results see the surveys on the domain [O'R87, She92, Urr98].

3.2 Viewpoint optimization

The methods which have been developed to optimize the placement of sensors or lights are all based on a sampling approach similar to the approximate aspect graph.

We present here the methods developed by Tarbox and Gottschlich [TG95]. Their aim is to optimize the placement of a laser and a camera (as presented in Fig. 2.12 page 16) to be able to inspect an object whose pose and geometry are known. The distance of the camera and laser to the object is fixed, viewpoint space is thus a viewing sphere even if perspective projection is used. The viewing sphere is tessellated starting with an icosahedron (Fig. 7.1 page 66). Sample points are distributed over the object. For each viewpoint, the visibility of each sample point is tested using ray-casting. It is recorded in a two dimensional array called the *viewability matrix* indexed by the viewpoint and sample point. (In fact two matrices are used since the visibility constraints are not the same for the camera and for the laser.)

The viewability matrix can be seen as a structure in segment space: each entry encodes if the segment joining a given viewpoint and a given sample point intersects the object.

The set of viewpoints which can see a given feature is called the *viewpoint set*. For more robustness, especially in case of uncertainties in the pose of the object, the viewpoints of the boundary of a viewpoint set are discarded, that is, the corresponding entry in the viewability matrix is set to 0. For each sample point, a difficulty-to-view is computed which depends on the number of viewpoints from which it is visible.

A set of pairs of positions for the laser and the camera are then searched which resumes to a set-cover problem. The first strategy they propose is greedy. The objective to maximize is the number of visible sample points weighted by their difficulty-to-view. Then each new viewpoint tries to optimize the same function without considering the already seen points until all points are visible from at least one viewpoint.

The second method uses simulated annealing (which is similar to a gradient descend which can "jump" over local minima). An arbitrary number of viewpoints are randomly placed on the viewing sphere, and their positions are then perturbated to maximize the number of visible sample points. If no solution is found for *n*, a new viewpoint is added and the optimization proceeds. This method provides results with fewer viewpoints.

Similar methods have been proposed for sensor placement [MG95, TUWR97], data acquisition for mobile robot on a 2D floorplan [GL99] and image-based representation [HLW96]. See Fig. 7.12 for an example of sensor planning.

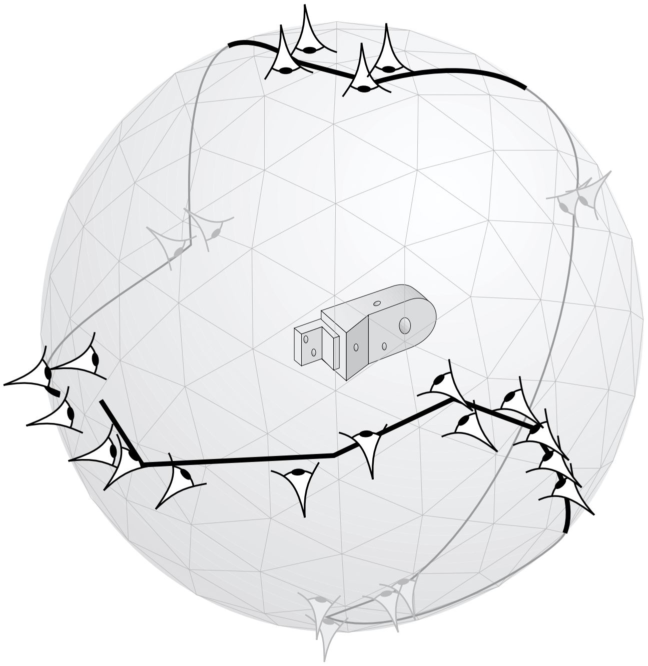


Figure 7.12: Planning of a stereo-sensor to inspect an object (adapted from [TUWR97])

Sturzlinger [Stu99] also proposes a similar method for the image-based representation of scenes. His viewpoint space is a horizontal plane at human height. Both objects and viewpoint space are adaptively subdivided for more efficient results. He then uses simulated annealing to optimize the set of viewpoints.

3.3 Local optimization and target tracking

Yi, Haralick and Shapiro [YHS95] optimize the position of both a camera and a light source. The position of the light should be such that features have maximal contrast in the image observed by the camera. Occlusion is not really handled in their approach since they performed their experiments only on a convex box. However their problem is in spirit very similar to that of viewpoint optimization for visibility constraints, so we include it in this survey because occlusion could be very easily included in their optimization metric.

They use no initial global computation such as the viewability matrix studied in the previous paragraph, but instead perform a local search. They perform a gradient descent successively on the light and camera positions. This method does not necessarily converge to a global maximum for both positions, but they claim that in their experiments the function to optimize is well behaved and convex and that satisfactory results are obtained.

Local optimization has also been proposed [LGBL97, FL98] for the computation of the motion of a mobile robot which has to keep a moving target in view. Assume the motion of the target is only partially predictable (by bound on the velocity for example). A local optimization is performed in the neighbourhood of the pursuer position in a game theoretic fashion: the pursuer has to take into account all the possible movements of the target to decide its position at the next timestep. For a possible pursuer position in free space, all the possible movements of the target are enumerated and the probability of its being visible is computed. The pursuer position with the maximum probability of future visibility is chosen. See Fig. 7.13 for an example of pursuit. The range of the sensor is taken into account.

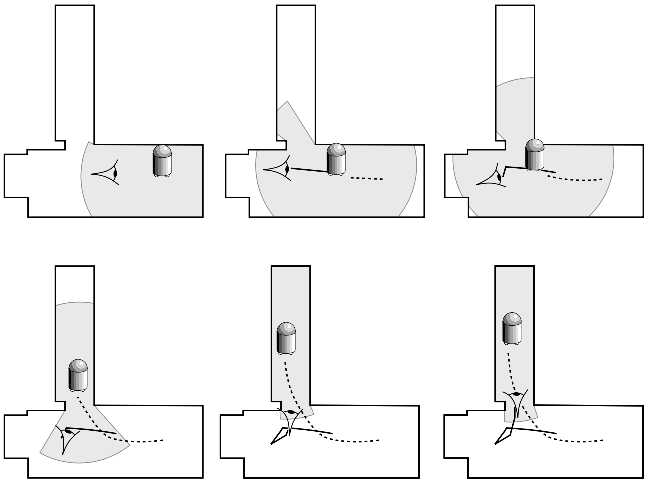


Figure 7.13: Tracking of a mobile target by an observer. The region in which the target is visible is in light grey (adapted from [LGBL97]).

They also propose another strategy for a better prediction [LGBL97]. The aim is here to maximize the escape time of the target. For each possible position of the pursuer, its visibility region is computed (the inverse of a shadow volume). The distance of the target to the boundary of this visibility region defines the minimum distance it has to cover to escape the pursuer (see Fig. 7.14).

The extension of these methods to the prediction of many timesteps is unfortunately exponential.

4 Frame-to-frame coherence

In section 1.5 we have presented applications of the aspect graph to updating a view as the observer continuously moves. The cost induced by the aspect graph has prevented the use of these methods. We now present methods which use the information encoded by visual events to update views, but which consider only a subset of them.

4.1 Coherence constraints

Hubschman and Zucker [HZ81, HZ82] have studied the so-called *frame-to-frame coherence* for static scenes. This approach is based on the fact that if the viewpoint moves continuously, two successive images are usually very similar. They study the occlusions between pairs of convex polyhedra.

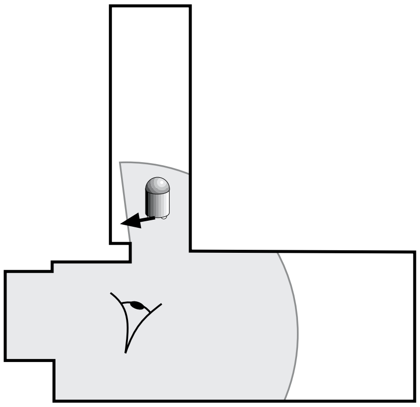


Figure 7.14: Tracking of a mobile target by an observer. The region in light grey is the region in which the target is visible from the observer. The thick arrow is the shortest path for the target to escape.

They note that a polyhedron will start (or stop) occluding another one only if the viewpoint crosses one of their separating planes. This corresponds to *EV* visual events. Moreover this can happen only for silhouette edges.

Each edge stores all the separating planes with all other polyhedra. These planes become active only when the edge is on the silhouette in the current view. As the viewpoint crosses one of the active planes, the occlusion between the two corresponding polyhedra is updated.

This approach however fails to detect occlusions caused by multiple polyhedra (*EEE* events are not considered). Furthermore, a plane is active even if both polyhedra are hidden by a closer one, in which case the new occlusion has no actual effect on the visibility of the scene; Transparent as well as opaque events are considered. These limitations however simplify the approach and make it tractable. Unfortunately, no implementation is reported.

4.2 Occlusion culling with visual events

Coorg and Teller [CT96] have extended their shadow-volume based occlusion culling presented in section 4.4 of chapter 5 to take advantage of frame-to-frame coherence.

The visibility of a cell of the scene subdivision can change only when a visual event is crossed. For each large occluder visibility changes can occur only for the neighbourhood of partially visible parts of the scene (see Fig. 7.15). They thus dynamically maintain the visual events of each occluders and test the viewpoint against them.

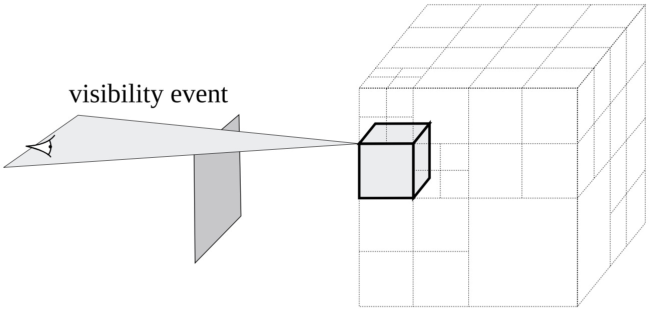


Figure 7.15: Occlusion culling and visual events

They explain that this can be seen as a local linearized version of the aspect graph. Indeed they maintain a superset of the *EV* boundaries of the current cell of the perspective aspect graph of the scene.

CHAPTER 8

Line-Space

Car il ne sera fait que de pure lumière
Puisée au foyer saint des rayons primitifs

Charles BAUDELAIRE, *Les Fleurs du Mal*

LINE-SPACE methods characterize visibility with respect to line-object intersections. The methods we present in section 1 partition lines according to the objects they intersect. Section 2 introduces graphs in line-space, while section 3 discusses Plücker coordinates, a powerful parameterization which allows the characterization of visibility using hyperplanes in 5D. Finally section 4 presents stochastic and probabilistic approaches in line-space.

1 Line-space partition

1.1 The Asp

Plantinga and Dyer [PD87, PD90, Pla88] devised the *asp* as an auxiliary data-structure to compute the aspect graph of polygonal objects. The definition of the *asp* depends on the viewing space considered. We present the *asp* for orthographic projection.

A duality is used which maps oriented lines into a 4 dimensional space. Lines are parameterized as presented in section 1.4 of chapter 3 and Fig. 3.2(a) (page 25) by their direction, denoted by two angles (θ, φ) and the coordinates (u, v) on an orthogonal plane. The *asp* for θ and φ constant is thus an orthographic view of the scene from direction (θ, φ). The *asp* of an object corresponds to the set of lines intersecting this object. See Fig. 8.1(a) and (b).

Occlusion in a view corresponds to subtraction in the *asp*: if object *A* is occluded by object *B*, then the *asp* of *B* has to be subtracted from the *asp* of *A* as shown in Fig. 8.1(c). In fact the intersection of the *asp* of two objects is the set of lines going through them. Thus if object *B* is in front of object *A*, and these lines no longer "see" *A*, they have to be removed from the *asp* of *A*.

The 1 dimensional boundaries of the *asp* correspond to the visual events necessary to build the aspect graph. See Fig. 8.1(c) where an *EV* event is represented. Since it is only a slice of the *asp*, only one line of the event

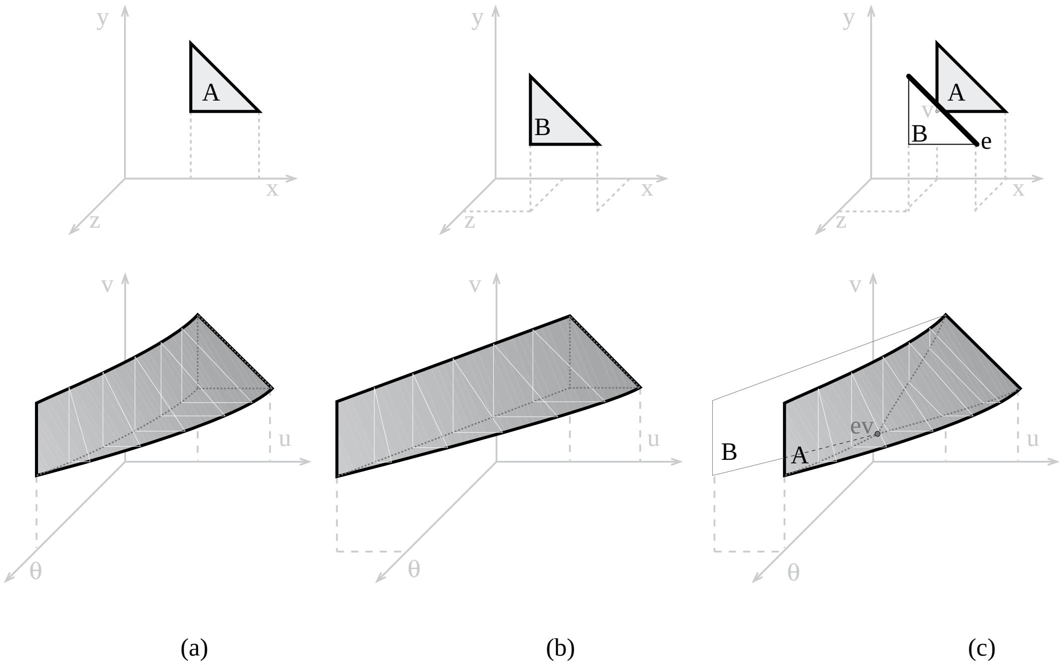


Figure 8.1: Slice of the *asp* for $\varphi = 0$ (adapted from [PD90]). (a) and (b) *Asp* for one triangle. The θ slices in white correspond to orthographic views of the triangle. When $\theta = 90^\circ$ the view of the triangle is a segment. (c) Occlusion corresponds to subtraction in *asp* space. We show the *asp* of triangle *A* which is occluded by *B*. Note the occlusion in the θ slices in white. We also show the outline of the *asp* of *B*. The visual event *EV* is a point in *asp* asp space.

is present under the form of a point. Since occlusion has been taken into account with subtraction, the *asp* contains only the opaque events, transparent events do not have to be detected and discarded as in Gigus and Malik's method [GM90] presented in section 1.3. Unfortunately no full implementation is reported. The size of the *asp* can be as high as $O(n^4)$, but as already noted, this does not give useful information about its practical behaviour with standard scenes.

In the case of perspective projection, the *asp* is defined in the 5 dimensional space of rays. Occlusion is also handled with subtractions. Visual events are thus the 2 dimensional boundaries of the *asp*.

1.2 The 2D Visibility Complex

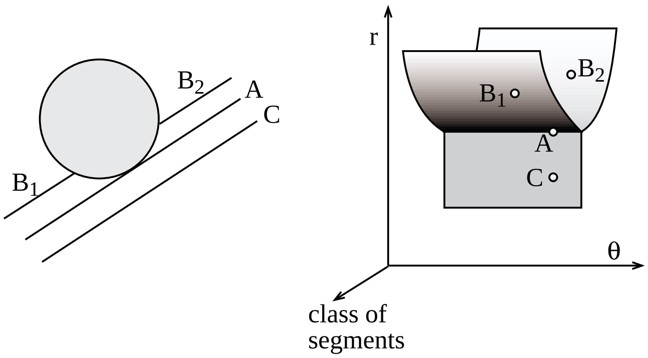
Pocchiola and Vegter [PV96b, PV96a] have developed the 2D *visibility complex* which is a topological structure encoding the visibility of a 2D scene. The idea is in a way similar to the *asp* to group rays which "see" the same objects. See [DP95] for a simple video presentation.

The central concept is that of *maximal free segments*. These are segments of maximal length that do not intersect the interior of the objects of the scene. More intuitively, a maximal free segment has its extremities on the boundary of objects, it may be tangent to objects but does not cross them. A line is divided in many maximal free segment by the objects it intersects. A maximal free segment represents a group of colinear rays which see the same objects. The manifold of 2D maximal free segments is two-dimensional nearly everywhere, except at certain branchings corresponding to tangents of the scene. A tangent segment has neighbours on both sides of the object and below the object (see Fig. 8.2).

The visibility complex is the partition of maximal free segments according to the objects at their extremities. A face of the visibility complex is bounded by chains of segments tangent to one object (see Fig. 8.3).

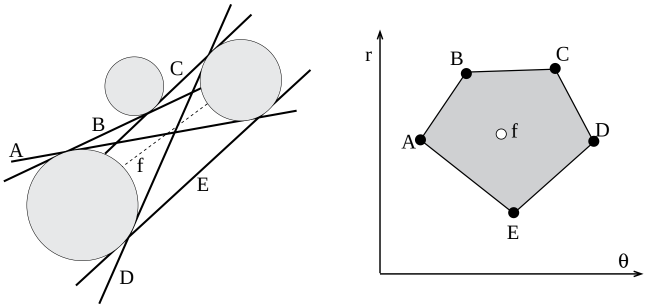
Pocchiola and Vegter [PV96b, PV96a] propose optimal output sensitive construction algorithms for the visibility complex of scenes of smooth objects. Rivière [Riv95, Riv97] has developed an optimal construction algorithm for polygonal scenes.

The visibility complex implicitly encodes the visibility graph (see section 2 of chapter 5) of the scene: its



(a) (b)

Figure 8.2: Topology of maximal free segments. (a) In the scene. (b) In a dual space where lines are mapped into points (the polar parameterization of line is used).



(a) (b)

Figure 8.3: A face of the visibility complex. (a) In the scene. (b) In a dual space.

vertices are the bitangents forming the visibility graph.

The 2D visibility complex has been applied to the 2D equivalent of lighting simulation by Orti *et al.* [ORDP96, DORP96]. The form factor between two objects corresponds to the face of the complex grouping the segments between these two objects. The limits of umbra and penumbra are the vertices (bitangents) of the visibility complex.

1.3 The 3D Visibility Complex

Durand *et al.* [DDP96, DDP97b] have proposed a generalization of the visibility complex for 3D scenes of smooth objects and polygons. The space of maximal free segments is then a 4D manifold embedded in 5D because of the branchings. Faces of the complex are bounded by tangent segments (which have 3 dimensions), bitangent segments (2 dimension), tritangent segments (1D) and finally vertices are segments tangent to four objects. If polygons are considered, the 1-faces are the EV and EEE critical lines.

The visibility complex is similar to the *asp*, but the same structure encodes the information for both perspective and orthographic projection. It moreover provides adjacencies between sets of segments.

Langer and Zucker [LZ97] have developed similar topological concepts (particularly the branchings) to describe the manifold of rays of a 3D scene in a shape-from-shading context.

See also section 4 where the difference between lines and maximal free segments is exploited.

1.4 Ray-classification

Ray classification is due to Arvo and Kirk [AK87]. The 5 dimensional space of rays is subdivided to accelerate ray-tracing computation. A ray is parameterized by its 3D origin and its direction which is encoded on a cube for simpler calculations. Beams in ray-space are defined by an XYZ interval (an axis aligned box) and an interval on the cube of directions (see Fig. 8.4).

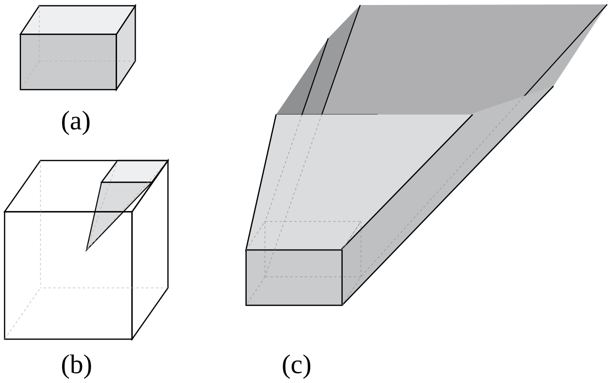


Figure 8.4: Ray classification. (a) interval in origin space. (b) interval in direction space. (c) Corresponding beam of rays.

The objects lying in the beam are computed using a cone approximation of the beam. They are also sorted by depth to the origin box. Each ray belonging to the beam then needs only be intersected with the objects inside the beam. The ray-intervals are lazily and recursively constructed. See Fig. 8.5 for an example of result.

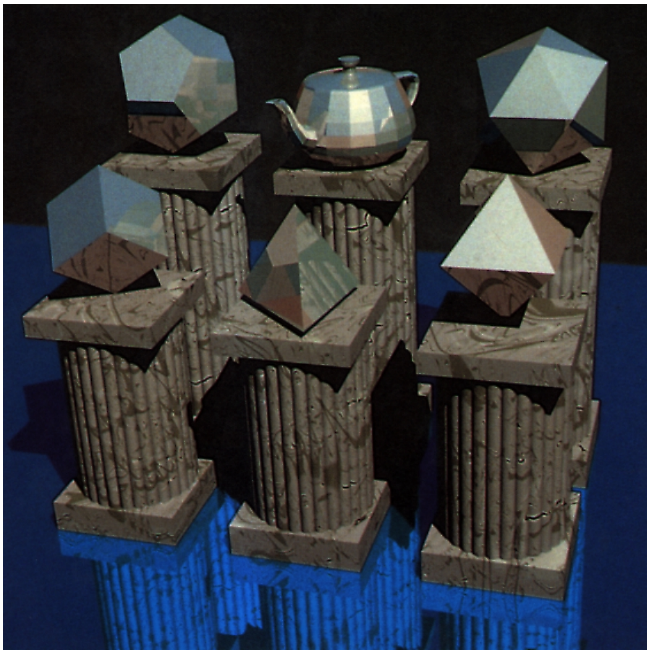


Figure 8.5: Image computed using ray classification (courtesy of Jim Arvo and David Kirk, Apollo Computer Inc.)

Speer [Spe92b] describes similar ideas and Kwon *et al* [KKCS98] improve the memory requirements of ray-classification, basically by using 4D line space instead of 5D ray-space. This method is however still memory intensive, and it is not clear that it is much more efficient that 3D regular grids.

The concept of the light buffer presented in section 2.2 of chapter 6 has been adapted for linear and area light source by Poulin and Amanatides [PA91] and by Tanaka and Takahashi [TT95, TT97]. The rays going through the source are also classified into beams. The latter paper uses an analytical computation of the visible part of the light source using the cross-scanline method reviewed in section 6 of chapter 4.

Lamparter *et al.* [LMW90] discretize the space of rays (using adaptive quadtrees) and rasterize the objects of the scene using a z-buffer like method. Hinkenjann and Müller [HM96] propose a similar scheme to classify segments using a 6 dimensional space (3 for each extremity of a segment).

1.5 Multidimensional image-based approaches

Recently there has been great interest in both computer vision and computer graphics for the study of the description of a scene through the use of a multidimensional function in ray-space. A 3D scene can be completely described by the light traveling through each point of 3D space in each direction. This defines a 5D function named the *plenoptic function* by Adelson and Bergen [AB91].

The plenoptic function describes light transport in a scene, similar data-structures have thus been applied for global illumination simulation [LF96, LW95, GSHG98].

Gortler *et al.* [GGSC96] and Levoy and Hanrahan [LH96] have simplified the plenoptic function by assuming that the viewer is outside the convex hull of the scene and that light is not modified while traveling in free-space. This defines a function in the 4 dimensional space of lines called *lumigraph* or *light-field*. This space is discretized, and a color is kept for each ray. A view can then be extracted very efficiently from any viewpoint by querying rays in the data structure. This data structure is more compact than the storage of one view for each 3D point (which defines a 5D function) for the same reason exposed before: a ray is relevant for all the viewpoints lying on it. There is thus redundancy if light does not vary in free-space.

A two plane parameterization is used both in the light-field [LH96] and lumigraph [GGSC96] approaches (see Fig 3.2(b) page 25). Xu *et al.* [GGC97] have studied the form of some image features in this dual space, obtaining results similar to those obtained in the aspect graph literature [PD90, GCS91]. Camahort *et al.* [CLF98] have studied the (non) uniformity of this parameterization and proposed alternatives based on tessellations of the direction sphere. Their first parameterization is similar to the one depicted in Fig. 3.2(a) using a direction and an orthogonal plane, while the second uses parameterization line using two points on a sphere bounding the scene. See section 4 and the book by Santalo [San76] for the problems of measure and probability on sets of lines. See also the paper by Halle [Hal98] where images from multiple viewpoints (organised on a grid) are rendered simultaneously using epipolar geometry.

Chrysanthou *et al.* [CCOL98] have adapted the lumigraph methods to handle ray occlusion query. They re-introduce a fifth dimension to handle colinear rays, and their scheme can be seen as a discretization of the 3D visibility complex.

Wang *et al.* [WBP98] perform an occlusion culling preprocessing which uses concepts from shaft culling, ray classification and lumigraph. Using a two-plane parameterization of rays leaving a given cell of space, they recursively subdivide the set of rays until each beam can be classified as blocked by a single object or too small to be subdivided.

2 Graphs in line-space

In this section we present some methods which build a graph in line space which encodes the visual events of a scene. As opposed to the previous section, only one and zero dimensional sets of lines are considered.

2.1 The Visibility Skeleton

Durand *et al* [DDP97c, DDP97a] have defined the *visibility skeleton* which can be seen either as a simplification of the 3D visibility complex or as a graph in line space defined by the visual events.

Consider the situation represented in Fig. 8.6(a). A visual event V_1V_2 and the corresponding critical line set are represented. Recall that it is a one dimensional set of lines. It is bounded by two *extremal stabbing lines* V_1V_2 and V_1V_3. Fig. 8.6(b) shows another visual event V_2E_2 which is adjacent to the same extremal stabbing line. This defines a graph structure in line space represented in Fig. 8.6(c). The arcs are the 1D critical line sets and the nodes are the extremal stabbing lines. Other extremal stabbing lines include lines going through one vertex and two edges and lines going through four edges (see Fig. 8.7).

Efficient access to the arcs of this graph is achieved through a two dimensional array indexed by the polygons at the extremity of each visual event. The visibility skeleton is built by detecting the extremal stabbing lines. The adjacent arcs are topologically deduced thanks to a catalogue of adjacencies. This avoids explicit geometric calculations on the visual events.

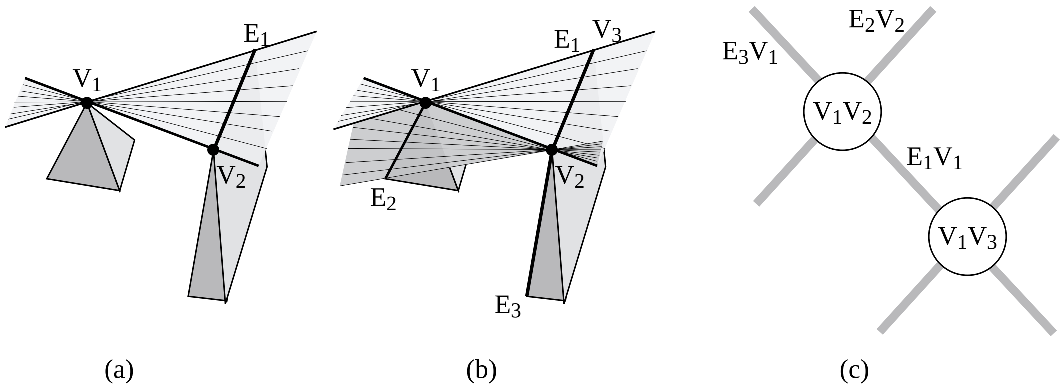


Figure 8.6: (a) An EV critical line set. It is bounded by two extremal stabbing lines V_1V_2 and V_1V_3. (b) Other EV critical line sets are adjacent to V_1V_2. (c) Corresponding graph structure in line space.

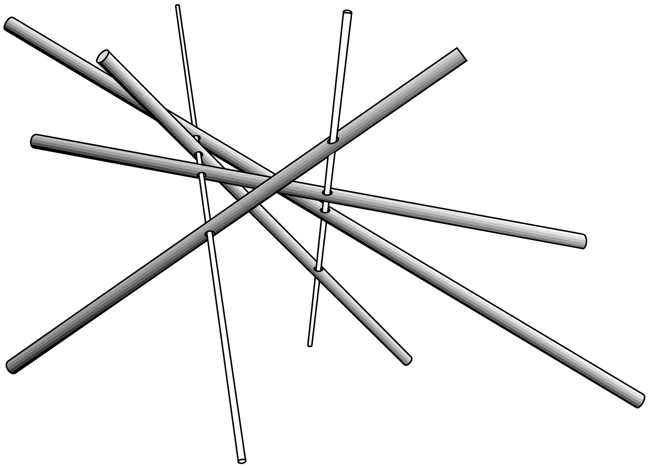


Figure 8.7: Four lines in general position are stabbed by two lines (adapted from [Tel92b])

The visibility skeleton has been implemented and used to perform global illumination simulation [DDP99]. Point-to-area form factors can be evaluated analytically, and the limits of umbra and penumbra can be quickly computed considering any polygon as a light source (as opposed to standard discontinuity meshing where only a small number of primary light sources are considered).

2.2 Skewed projection

McKenna et O'Rourke [MO88] consider a scene which is composed of lines in 3D space. Their aim is to study the class of another line in a sense similar to the previous section if the original lines are the edges of polyhedron, or to compute the mutually visible faces of polyhedra.

They use a *skewed projection* to reduce the problem to 2D computations. Consider a pair of lines L_1 and L_2 as depicted in Fig. 8.8. Consider the segment joining the two closest points of the lines (shown dashed) and the plane P orthogonal to this segment and going through its mid-point. Each point on P defines a unique line going through L_1 and L_2. Consider a third line L_3. It generates EEE critical lines. The intersections of these critical lines with plane P lie on an hyperbola H.

The intersections of the hyperbolae defined by all other lines of the scene allow the computation of the extremal stabbing lines stabbing L_1 and L_2. The computation of course has to be performed in the $O(n^2)$ planes defined by all pairs of lines. A graph similar to the visibility skeleton is proposed (but for sets of lines). No implementation is reported.

The skewed projection duality has also been used by Jaromczyk and Kowaluk [JK88] in a stabbing context

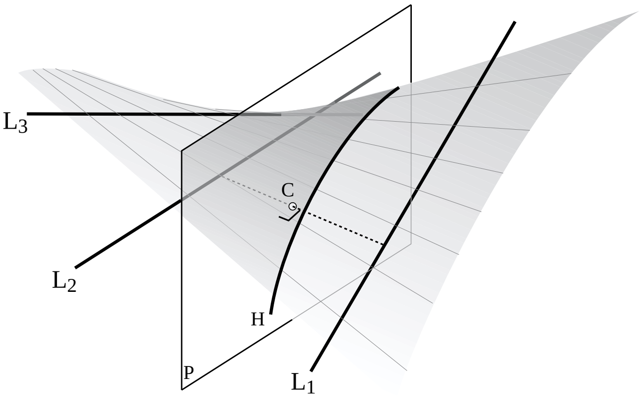


Figure 8.8: Skewed projection.

and by Bern *et al.* [BDEG90] to update a view along a linear path (the projection is used to compute the visual events at which the view has to be updated).

3 Plücker coordinates

3.1 Introduction to Plücker coordinates

Lines in 3D space can not be parameterized continuously. The parameterizations which we have introduced in section 1.4 of chapter 3 both have singularities. In fact there cannot be a smooth parameterization of lines in 4D without singularity. One intuitive way to see this is to note that it is not possible to parameterize the S^2 sphere of directions with two parameters without a singularity. Nevertheless, if S^2 is embedded in 3D, there is a trivial parameterization, *i.e.* x,y,z. However not all triples of coordinates correspond to a point on S^2.

Similarly, oriented lines in space can be parameterized in a 5D space with the so-called *Pl ücker coordinates* [Plü65]. The equations are given in appendix 11, here we just outline the principles. One nice property of Plücker coordinates is that the set of lines which intersect a given line a is a hyperplane in Plücker space (its dual Π_a; The same notation is usually used for the dual of a line and the corresponding hyperplane). It separates Plücker space into oriented lines which turn around ℓ clockwise or counterclockwise (see Fig. 8.9).

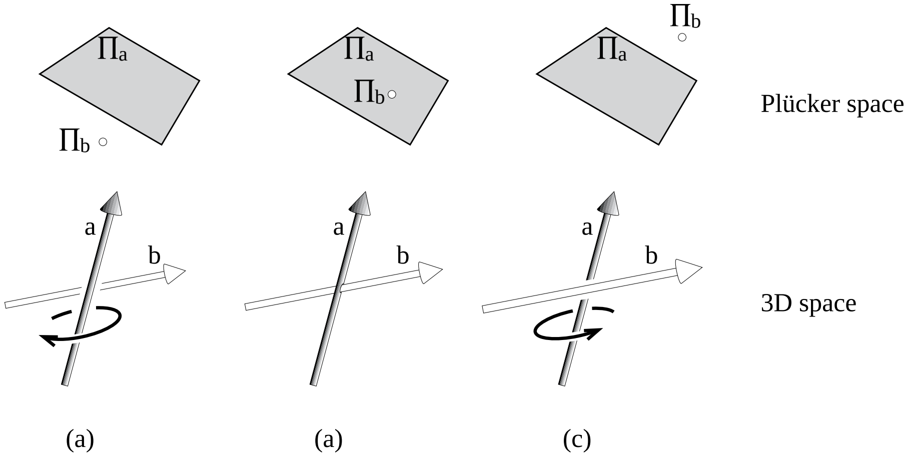


Figure 8.9: In Plücker space the hyperplane corresponding to a line a separates lines which turn clockwise and counterclockwise around a. (The hyperplane is represented as a plane because a five-dimensional space is hard to illustrate, but note that the hyperplane is actually 4D).

As for the embedding of S^2 which we have presented, not all 5-uples of coordinates in Plücker space correspond to a real line. The set of lines in this parameterization lie on a quadric called the *Pl ücker hypersurface* or *Grassman manifold* or *Klein quadric*.

Now consider a triangle in 3D space. All the lines intersecting it have the same orientation with respect to the three lines going through its edges (see Fig. 8.10). This makes stabbing computations very elegant in Plücker space. Linear calculations are performed using the hyperplanes corresponding to the edges of the scene, and the intersection of the result with the Plücker hypersurface is then computed to obtain real lines.

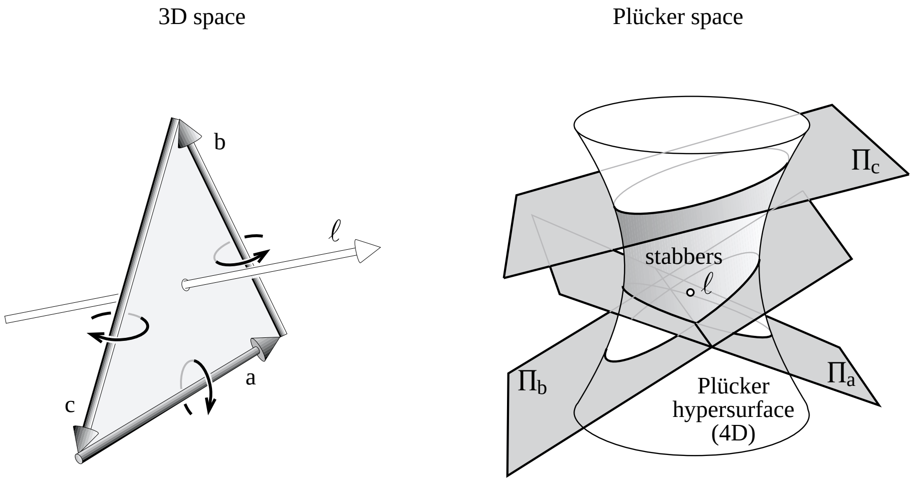


Figure 8.10: Lines stabbing a triangle. In 3D space, if the edges are well oriented, all stabbers rotate around the edges counterclockwise. In Plücker space this corresponds to the intersection of half spaces. To obtain real lines, the intersection with the Plücker hypersurface must be considered. (In fact the hyperplanes are tangent to the Plücker hypersurface)

Let us give a last example of the power of Plücker duality. Consider three lines in 3D space. The lines stabbing each line lie on its (4D) hyperplanes in Plücker space. The intersection of the three hyperplane is a 2D plane in Plücker space which can be computed easily. Once intersected with the Plücker hypersurface, we obtain the *EEE* critical line set as illustrated Fig. 8.11.

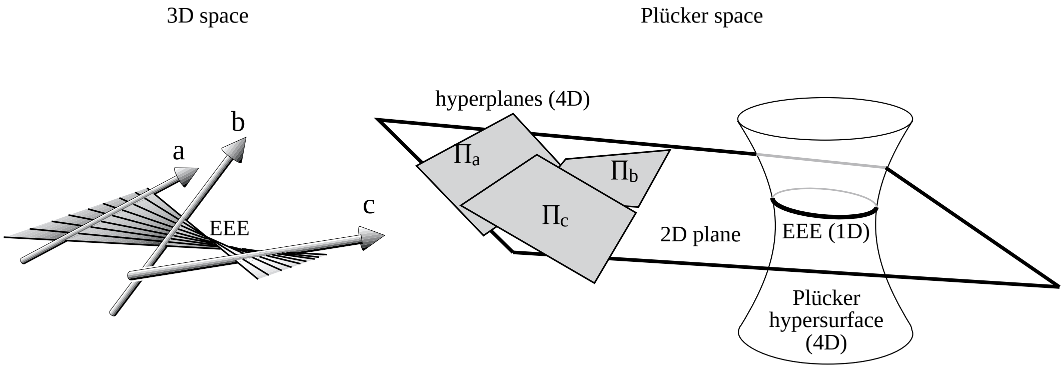


Figure 8.11: Three lines define a *EEE* critical line set in 3D space. This corresponds to the intersection of hyperplanes (not halfspaces) in Plücker space. Note that hyperplanes are 4D while their intersection is 2D. Unfortunately they are represented similarly because of the lack of dimensions of this sheet of paper.(adapted from [Tel92b]).

More detailed introductions to Plücker coordinates can be found in the books by Sommerville [Som51] or Stolfi [Sto91] and in the thesis by Teller [Tel92b] [1]. See also Appendix 11.

[1] Plücker coordinates can also be extended to use the 6 coordinates to describe forces and motion. See *e.g.* [MS85, PPR99]

3.2 Use in computational geometry

Plücker coordinates have been used in computational geometry mainly to find stabbers of sets of polygons, for ray-shooting and to classify lines with respect to sets of lines (given a set of lines composing the scene and two query lines, can we continuously move the first to the second without intersecting the lines of the scene).

We give an overview of a paper by Pellegrini [Pel93] which deals with ray-shooting in a scene composed of triangles. He builds the arrangement of hyperplanes in Plücker space corresponding to the scene edges. He shows that each cell of the arrangement corresponds to lines which intersect the same set of triangles. The whole 5D arrangement has to be constructed, but then only cells intersecting the Plücker hypersurface are considered. He uses results by Clarkson [Cla87] on point location using random sampling to build a point-location data-structure on this arrangement. Shooting a ray then consists in locating the corresponding line in Plücker space. Other results on ray shooting can be found in [Pel90, PS92, Pel94].

This method is different in spirit from ray-classification where the object-beam classification is calculated in object space. Here the edges of the scene are transformed into hyperplanes in Plücker space.

The first use of Plücker space in computational geometry can be found. in a paper by Chazelle *et al.* [CEG$^+$96]. The orientation of lines in space also has implications on the study of cycles in depth order as studied by Chazelle *et al.* [CEG$^+$92] who estimate the possible number of cycles in a scene . Other references on lines in space and the use of Plücker coordinates can be found in the survey by Pellegrini [Pel97b].

3.3 Implementations in computer graphics

Teller [Tel92a] has implemented the computation of the *antipenumbra* cast by a polygonal source through a sequence of polygonal openings *portals* (*i.e.* the part of space which may be visible from the source). He computes the polytope defined by the edges of all the openings, then intersects this polytope with the Plücker hypersurface, obtaining the critical line sets and extremal stabbing lines bounding the antipenumbra (see Fig. 8.12 for an example).

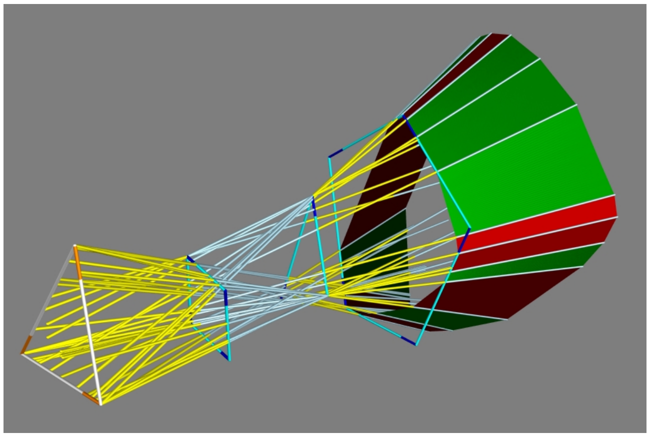


Figure 8.12: Antipenumbra cast by a triangular light source through a sequence of three polygonal openings. EEE boundaries are in red (image courtesy of Seth J. Teller, University of Berkeley).

He however later noted [TH93] that this algorithm is not robust enough for practical use.

Nevertheless, in this same paper he and Hanrahan [TH93] actually used Plücker coordinates to classify the visibility of objects with respect to parts of the scene in a global illumination context for architectural scenes (see section 7 of chapter 5). They avoid robustness issues because no *geometric construction* is performed in 5D space (like computing the intersection between two hyperplanes), only *predicates* are evaluated ("is this point above this hyperplane?").

4 Stochastic approaches

This section surveys methods which perform visibility calculation using a probabilistic sampling in line-space.

4.1 Integral geometry

The most relevant tool to study probability over sets of lines is *integral geometry* introduced by Santalo [San76]. Defining probabilities and measure in line-space is not straightforward. The most natural constraint is to impose that this measure be invariant under rigid motion. This defines a unique measure in line-space, up to a scaling factor.

Probabilities can then be computed on lines, which is a valuable tool to understand ray-casting. For example, the probability that a line intersects a convex object is proportional to its surface.

An unexpected result of integral geometry is that a uniform sampling of the lines intersecting a sphere can be obtained by joining pairs of points uniformly distributed on the surface of the sphere (note that this is not true in 2D).

The classic parameterization of lines $x = az + p, y = bz + q$ (similar to the two plane parameterization of Fig. 3.2(b) page 25) has density $\frac{da\,db\,dp\,dq}{(1+a^2+b^2)^2}$. If a, b, p, q are uniformly and randomly sampled, this formula expresses the probability that a line is picked. It also expresses the variation of sampling density for light-field approaches described in section 1.5. Regions of line space with large values of a, b will be more finely sampled. Intuitively, sampling is higher for lines that have a gazing angle with the two planes used for the parameterization.

Geometric probability is also covered in the book by Solomon [Sol78].

4.2 Computation of form factors using ray-casting

Most radiosity implementations now use ray-casting to estimate the visibility between two patches, as introduced by Wallace *et al.* [WEH89]. A number of rays (typically 4 to 16) are cast between a pair of patches. The number of rays can vary, depending on the *importance* of the given light transfer. Such issues will be treated in section 1.1 of chapter 9.

The integral geometry interpretation of form factors has been studied by Sbert [Sbe93] and Pellegrini [Pel97a]. They show that the form factor between two patches is proportional the probability that a line intersecting the first one intersects the second. This is the measure of lines intersecting the two patches divided by the measure of lines intersecting the first one. Sbert [Sbe93] proposes some estimators and derives expressions for the variance depending on the number of rays used.

4.3 Global Monte-Carlo radiosity

Buckalew and Fussel [BF89] optimize the intersection calculation performed on each ray. Indeed, in global illumination computation, all intersections of a line with the scene are relevant for light transfer. As shown in Fig. 8.13, the intersections can be sorted and the contribution computed for the interaction between each consecutive pair of objects. They however used a fixed number of directions and a deterministic approach.

Sbert [Sbe93] introduced *global Monte-Carlo radiosity*. As in the previous approach all intersections of a line are taken into account, but a uniform random sampling of lines is used, using pairs of points on a sphere.

Related results can be found in [Neu95, SPP95, NNB97]. Efficient hierarchical approaches have also been proposed [TWFP97, BNN$^+$98].

4.4 Transillumination plane

Lines sharing the same direction can be treated simultaneously in the previous methods. This results in a sort of orthographic view where light transfers are computed between consecutive pairs of objects overlapping in the view, as shown in Fig. 8.14.

The plane orthogonal to the projection direction is called the *transillumination plane*. An adapted hidden-surface removal method has to be used. The z-buffer can be extended to record the z values of all objects projecting on a pixel [SKFNC97], or an analytical method can be used [Pel99, Pel97a].

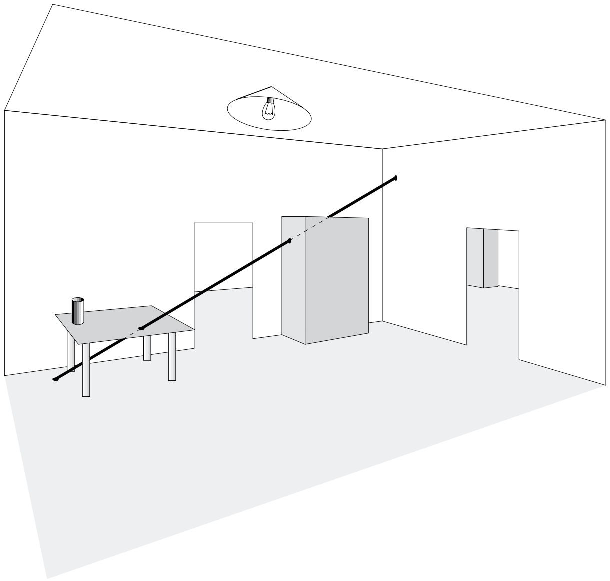


Figure 8.13: Global Monte-Carlo radiosity. The intersection of the line in bold with the scene allows the simulation of light exchanges between the floor and the table, between the table and the cupboard and between the cupboard and the ceiling.

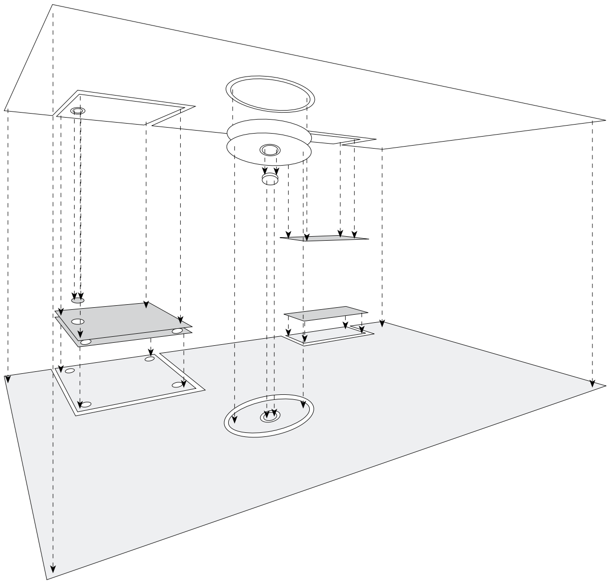


Figure 8.14: Transillumination plane. The exchanges for one direction (here vertical) are all evaluated simultaneously using an extended hidden surface removal algorithm.

CHAPTER 9

Advanced issues

> Au reste, il n'est pas inutile de remarquer que tout ce qu'on démontre, soit dans l'optique, soit dans la perspective sur les ombres des corps, est exact à la vérité du côté mathématique, mais que si on traite cette matière physiquement, elle devient alors fort différente. L'explication des effets de la nature dépend presque toujours d'une géométrie si compliquée qu'il est rare que ces effets s'accordent avec ce que nous en aurions attendu par nos calculs.
>
> FORMEY, article sur l'ombre de l'*Encyclopédie*.

WE NOW TREAT two issues which we believe crucial for visibility computations and which unfortunately have not received much attention. Section 1 deals with the control of the precision of computations either to ensure that a required precision is satisfied, or to simplify visibility information to make it manageable. Section 2 treats methods which attempt to take advantage of temporal coherence in scenes with moving objects.

1 Scale and precision

Visibility computations are often involved and costly. We have surveyed some approximate methods which may induce artifacts, and some exact methods which are usually resource-intensive. It is thus desirable to control the error in the former, and trade-off time versus accuracy in the latter. Moreover, all visibility information is not always relevant, and it can be necessary to extract what is useful.

1.1 Hierarchical radiosity: a paradigm for refinement

Hierarchical radiosity [HSA91] is an excellent paradigm of refinement approaches. Computational resources are spent for "important" light exchanges. We briefly review the method and focus on the visibility problems involved.

In hierarchical radiosity the scene polygons are adaptively subdivided into patches organised in a pyramid. The radiosity is stored using Haar wavelets [SDS96]: each quadtree node stores the average of its children. The light exchanges are simulated at different levels of precision: exchanges will be simulated between smaller elements of the quadtree to increase precision as shown in Fig. 9.1. *Clustering* improves hierarchical radiosity by using a full hierarchy which groups *clusters* of objects [SAG94, Sil95].

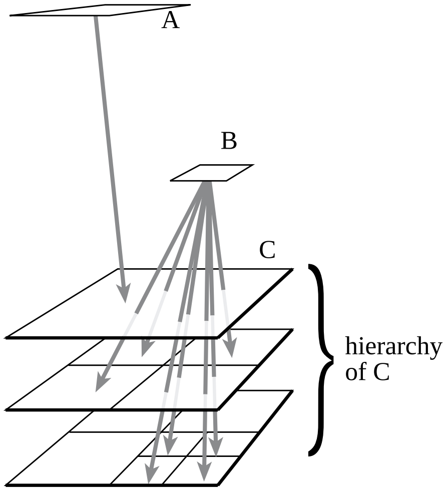


Figure 9.1: Hierarchical radiosity. The hierarchy and the exchanges arriving at *C* are represented. Exchanges with *A* are simulated at a coarser level, while those with *B* are refined.

The crucial component of a hierarchical radiosity system is the *refinement criterion* (or *oracle*) which decides at which level a light transfer will be simulated. Originally, Hanrahan *et al.* [HSA91] used a radiometric criterion (amount of energy exchanged) and a visibility criterion (transfers with partial visibility are refined more). This results in devoting more computational resources for light transfers which are important and in shadow boundary regions. See also [GH96].

For a deeper analysis and treatment of the error in hierarchical radiosity, see *e.g.*, [ATS94, LSG94, GH96, Sol98, HS99].

1.2 Other shadow refinement approaches

The volumetric visibility method presented in section 1.3 of chapter 5 is also well suited for a progressive refinement scheme. An oracle has to decide at which level of the volumetric hierarchy the transmittance has to be considered. Sillion and Drettakis [SD95] use the size of the *features* of the shadows.

The key observation is that larger object which are closer to the receiver cast more significant shadows, as illustrated by Fig. 9.2. They moreover take the *correlation* of multiple blockers into account using an image-based approach. The objects inside a cluster are projected in a given direction onto a plane. Bitmap erosion operators are then used to estimate the size of the connected portions of the blocker projection. This can be seen as a first approximation of the convolution method covered in section 6 of chapter 6 [SS98a].

Soler and Sillion [SS96b, Sol98] propose a more complete treatment of this refinement with accurate error bounds. Unfortunately, the bounds are harder to derive in 3D and provide looser estimates.

The refinement of shadow computation depending on the relative distances of blockers and source has also been studied by Asensio [Ase92] in a ray-tracing context.

Telea and van Overveld [Tv97] efficiently improve shadows in radiosity methods by performing costly visibility computations only for blockers which are close to the receiver.

1.3 Perception

The goal of most image synthesis methods is to produce images which will be seen by human observers. Gibson and Hubbold [GH97] thus perform additional computation in a radiosity method only if they may induce a change which will be noticeable. Related approaches can be found in [Mys98, BM98, DDP99, RPG99].

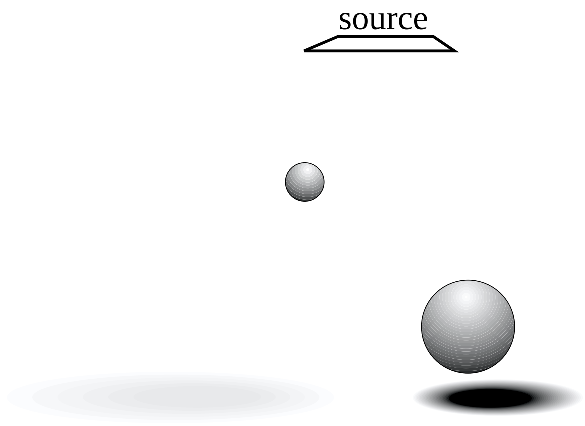


Figure 9.2: Objects which are larger and closer to the receiver cast more significant shadows. Note that the left hand sphere casts no umbra, only penumbra.

Perceptual metrics have also been applied to the selection of discontinuities in the illumination function [HWP97, DDP99].

1.4 Explicitly modeling scale

One of the major drawbacks of aspect graphs [FMA+92] is that they have been defined for perfect views: all features are taken into account, no matter the size of their projection.

The *Scale-space aspect graph* has been developed by Eggert *et al.* [EBD+93] to cope with this. They discuss different possible definitions of the concept of "scale". They consider that two features are not distinguishable when their subtended angle is less than a given threshold. This defines a new sort of visual event, which corresponds to the visual merging of two features. These are circles in 2D (the set of points which form a given angle with a segment is a circle). See Fig. 9.3.

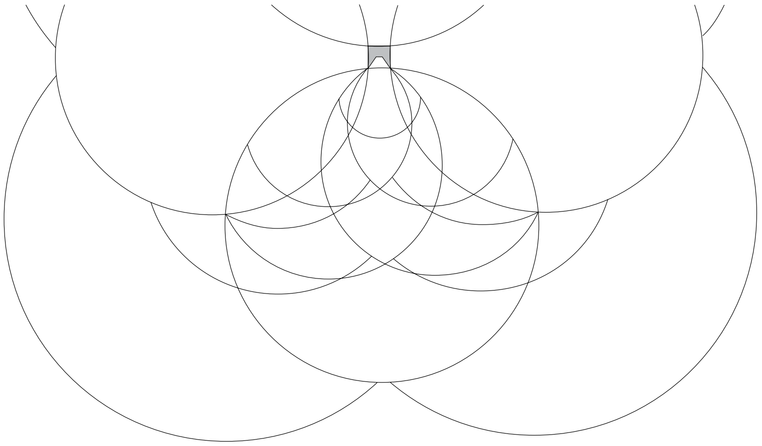


Figure 9.3: Scale-space aspect graph in 2D using perspective projection for the small object in grey. Features which subtend an angle of less than 4° are considered indistinguishable. The circles which subdivide the plane are the visual events where features of the object visually merge.

Scale (the angle threshold) defines a new dimension of the viewpoint space. Fig. 9.3 in fact represents a slice $scale = 4°$ of the scale-space aspect graph. Cells of this aspect graph have a scale extent, and their boundaries change with the scale parameter. This approach allows an explicit model of the resolution of features, at the cost of an increases complexity.

Shimshoni and Ponce [SP97] developed the *finite resolution aspect graph* in 3D. They consider orthographic projection and a single threshold. When resolution is taken into account, some *accidental* views are likely to be observed: An edge and a vertex seem superimposed in the neighbourhood of the exact visual event. Visual events are thus doubled as illustrated in Fig. 9.4.

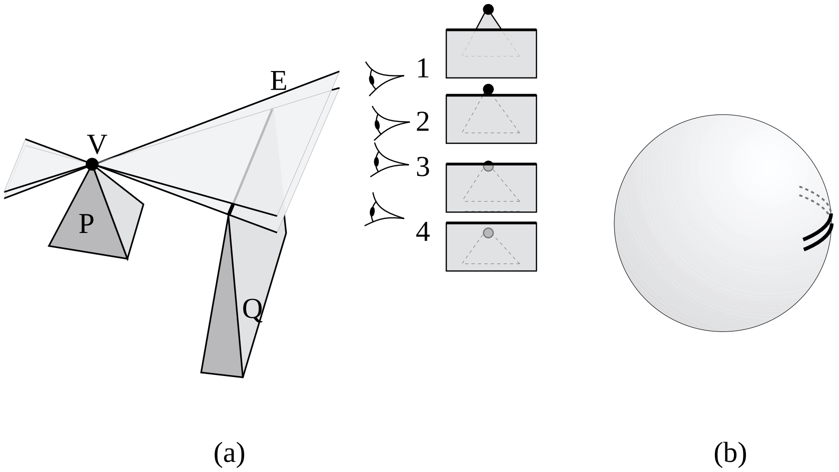


Figure 9.4: Finite resolution aspect graph. (a) The EV event is doubled. Between the two events (viewpoint 2 and 3), E and V are visually superimposed. (b) The doubled event on the viewing sphere.

For the objects they test, the resulting finite resolution aspect graph is larger. The number events discarded because the generators are merged does not compensate the doubling of the other events. However, tests on larger objects could exhibit different results.

See also the work by Weinshall and Werman on the likelihood and stability of views [WW97].

1.5 Image-space simplification for discontinuity meshing

Stewart and Karkanis [SK98] propose a finite resolution construction of discontinuity meshes using an image-space approach. They compute views from the vertices of the polygonal source using a z-buffer. The image is segmented to obtain a visibility map. The features present in the images are used as visual event generators.

This naturally eliminates small objects or features since they aggregate in the image. Robustness problems are also avoided because of the image-space computations. Unfortunately, only partial approximate discontinuity meshes are obtained, no backprojection computation is proposed yet.

2 Dynamic scenes

We have already evoked *temporal coherence* in the case of a moving viewpoint in a static scene (section 4.2 of chapter 7). In this section we treat the more general case of a scene where objects move. If the motions are continuous, and especially if few objects move, there is evidence that computation time can be saved by exploiting the similarity between consecutive timesteps.

In most cases, the majority of the objects are assumed static while a subset of objects actually move. We can distinguish cases where the motion of the objects is known in advance, and those where no *a priori* information is known, and thus updates must be computed on a per frame basis.

Different approaches can be chosen to take advantage of coherence:

- The computation is completely re-performed for a sub-region of space;
- The dynamic objects are deleted (and the visibility information related to them is discarded) then re-inserted at their new position;
- A validity time-interval is computed for each piece of information;

- The visibility information is “smoothly updated”.

2.1 Swept and motion volumes

A *swept volume* is the volume swept by an object during a time interval. Swept volumes can also be used to bound the possible motion of an object, especially in robotics where the degrees of freedom are well defined [AA95]. These swept volumes are used as static blockers.

A *motion volume* is a simplified version of swept volumes similar to the shafts defined in section 6.1 of chapter 5. They are simple volume which enclose the motion of an object. Motion volumes were first used in radiosity by Baum *et al.* [BWCG86] to handle the motion of one object. A hemicube is used for form-factor computation. Pixels where the motion volume project are those which need recomputation.

Shaw [Sha97] and Drettakis and Sillion [DS97] determine form factors which require recomputation using a motion volume-shaft intersection technique.

Sudarsky and Gotsman [SG96] use motion volumes (which they call *temporal bounding volumes*) to perform occlusion culling with moving objects. They alleviate the need to update the spatial data-structure (BSP or octree) for each frame, because these volumes are used in place of the objects, making computations valid for more than one frame.

2.2 4D methods

Some methods have been proposed to speed-up ray-tracing animations using a four dimensional space-time framework developed by Glassner [Gla88]. The temporal extent of ray-object intersections is determined, which avoids recomputation when a ray does not intersect a moving object. See also [MDC93, CCD91] for similar approaches.

Ray-classification has also been extended to 6D (3 for the origin of a ray, 2for its direction, and 1 for time) [Qua96, GP91].

Global Monte-Carlo radiosity presented in section 4.3 of chapter 8 naturally extends to 4D as demonstrated by Besuievsky *et al* [BS96]. Each ray-static object intersection is used for the whole length of the animation. Only intersections with moving objects require recomputation.

2.3 BSP

BSP trees have been developed for rapid view computation in static scenes. Unfortunately, their construction is a preprocessing which cannot be performed for each frame.

Fuchs *et al.* [FAG83] consider pre-determined paths and place bounding planes around the paths. Torres [Tor90] builds a multi-level BSP tree, trying to separate objects with different motion without splitting them.

Chrysanthou and Slater [CS92, CS95, CS97] remove the moving objects from the database, update the BSP tree, and then re-introduce the object at its new location. The most difficult part of this method is the update of the BSP tree when removing the object, especially when the polygons of the object are used at a high level of the tree as splitting planes. In this case, all polygons which are below it in the BSP-tree have to be updated in the tree. This approach was also used to update limits of umbra and penumbra [CS97].

Agarwal *et al.* [AEG98] propose an algorithm to maintain the cylindrical BSP tree which we have presented in section 1.4 of chapter 5. They compute the events at which their BSP actually needs a structural change. This happens when a triangle becomes vertical, when an edge becomes parallel to the *yz* plane, or when a triangle enters or leaves a cell defined by the BSP tree.

2.4 Aspect graph for objects with moving parts

Bowyer *et al.* [EB93] discuss the extension of aspect graphs for articulated assemblies. The degrees of freedom of the assembly increase the dimensionality of viewpoint space (which they call aspect space). For example, if the assembly has only one translational degree of freedom and if 3D perspective is used, the aspect graph has to be computed in 4D, 3 dimensions for the viewpoint and one for translation. This is similar to the scale-space aspect graph presented in section 1.4 where scale increases dimensionality.

Accidental configurations correspond to values of the parameters of the assembly where the aspect graph changes. They occur at a generalization of visual events in the higher dimensional aspect space. For example when two faces become parallel.

Two extensions of the aspect graph are proposed, depending on the way accidental configurations are handled. They can be used to partition aspect space like in the standard aspect graph definition. They can also be used to partition first the configuration space (in our example, it would result in intervals of the translational parameter), then a different aspect graph is computed for each cell of the configuration space partition. This latter approach is more memory demanding since cells of different aspect graphs are shared in the first approach. Construction algorithms are just sketched, and no implementation is reported.

2.5 Discontinuity mesh update

Loscos and Drettakis [LD97] and Worall *et al.* [WWP95, WHP98] maintain a discontinuity mesh while one of the blockers moves. Limits of umbra and penumbra move smoothly except when an object starts or stops casting shadows on another one. Detecting when a shadow limit goes off an object is easy.

To detect when a new discontinuity appears on one object, the discontinuities cast on other objects can be used as illustrated in Fig. 9.5.

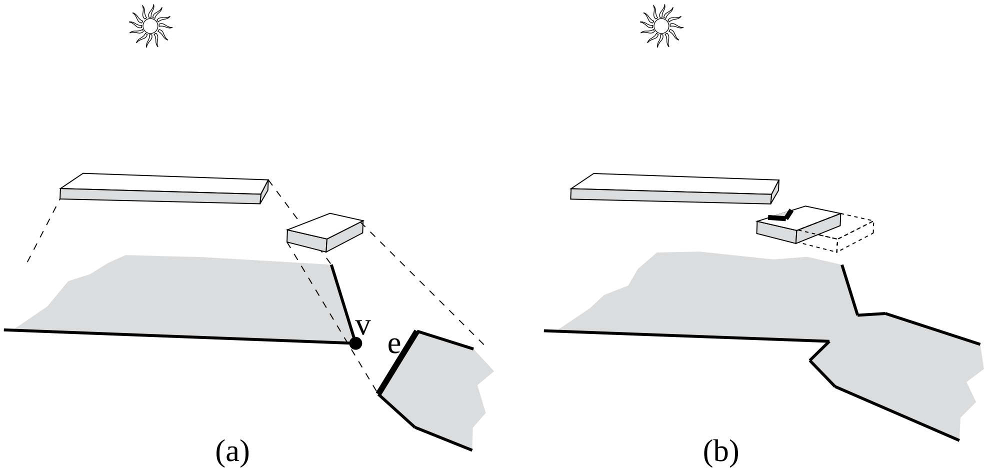


Figure 9.5: Dynamic update of limits of shadow. The situation where shadows appear on the moving object can be determined by checking the shadows on the floor. This can be generalized to discontinuity meshes (after [LD97]).

2.6 Temporal visual events and the visibility skeleton

In chapter 2 and 3 of [Dur99], we have presented the notion of a *temporal visual event*. Temporal visual events permit the generalization of the results presented in the previous section. They correspond to the accidental configurations studied for the aspect graph of an assembly.

Temporal visual events permit the update of the visibility skeleton while objects move in the scene. This is very similar to the static visibility skeleton, since temporal visual events describe adjacencies which determine which nodes and arcs of the skeleton should be modified.

Similarly, a catalogue of singularities has been developed for moving objects, defining a *temporal visibility complex*.

CHAPTER 10

Conclusions of the survey

Ils ont tous gagné !

Jacques MARTIN

SURVEYING work related to visibility reveals a great wealth of solutions and techniques. The organisation of the second part of this thesis has attempted to structure this vast field. We hope that this survey will be an opportunity to derive new methods or improvements from techniques developed in other fields. Considering a problem under different angles is a powerful way to open one's mind and find creative solutions. We again invite the reader not to consider our classification as restrictive; on the contrary, we suggest that methods which have been presented in one space be interpreted in another space. In what follows, we give a summary of the methods which we have surveyed, before presenting a short discussion.

1 Summary

In chapter 2 we have presented visibility problems in various domains: computer graphics, computer vision, robotics and computational geometry.

In chapter 3 we have propose a classification of these methods according to the space in which the computations are performed: object space, image space, viewpoint space and line-space. We have described the *visual events* and the *singularities of smooth mappings* which explain "how" visibility changes in a scene: the appearance or disappearance of objects when an observer moves, the limits of shadows, etc.

We have briefly surveyed the classic hidden-part removal methods in chapter 4.

In chapter 5 we have dealt with object-space methods. The two main categories of methods are those which use a "regular" spatial decomposition (grid, hierarchy of bounding volumes, BSP trees), and those which use frusta or shafts to characterize visibility. Among the latter class of methods, the main distinction is between those which are interested in determining if a point (or an object) lies inside the frustum or shaft, and those which compute the boundaries of the frustum (*e.g.*, shadow boundaries). Fundamental data-structures have also been presented: The 2D visibility graph used in motion planning links all pairs of mutually visible vertices of a planar polygonal scene, and the visual hull of an object *A* represents the largest object with the same occlusion properties as *A*.

Images-space methods, surveyed in chapter 6 perform computation directly in the plane of the final image, or use an intermediate plane. Most of them are based on the z-buffer algorithm.

Chapter 7 has presented methods which consider viewpoints and the the visibility properties of the corresponding views. The aspect graph encodes all the possible views of an object. The viewpoints are partitioned into cells where a view is qualitatively invariant, that is, the set of visible features remains constant. The boundaries of such cells are the visual events. This structure has important implications and applications in computer vision, robotics, and computer graphics. We have also presented methods which optimize the viewpoint according to the visibility of a feature, as well as methods based on visual events which take advantage of *temporal coherence* by predicting when a view changes.

In chapter 8 we have surveyed work in line or ray space. We have presented methods which partition the rays according to the object they see. We have seen that visual events can be encoded by lines in line-space. A powerful dualisation has been studied which maps lines into five dimensional points, allowing for efficient and elegant visibility characterization. We have presented some elements of probability over sets of lines, and their applications to lighting simulation.

Finally, in the previous chapter we have discussed two important issues: precision and moving objects. We have studied techniques which refine their computations where appropriate, as well as techniques which attempt to cope with intensive and intricate visibility information by culling too fine and unnecessary information. Techniques developed to deal with dynamic scenes include swept or motion volumes, 4D method (where time is the fourth dimension), and smooth updates of BSP trees or shadow boundaries.

Table 10.1 summarizes the techniques which we have presented, by domain and space.

2 Discussion

A large gap exists between exact and approximate methods. Exact methods are often costly and prone to robustness problems, while approximate methods suffer from aliasing artifacts. Smooth trade-off and efficient adaptive approximate solutions should be developed. This requires both to be able to refine a computation and to efficiently determine the required accuracy.

Visibility with moving objects and temporal coherence have received little attention. Dynamic scenes are mostly treated as successions of static timesteps for which everything is recomputed from scratch. Solutions should be found to efficiently identify the calculations which actually need to be performed after the movement of objects.

As evoked in the introduction of this survey, no practical guide to visibility techniques really exists. Some libraries or programs are available (see for example appendix 12) but the implementation of reusable visibility code in the spirit of *C-GAL* [FGK$^+$96] would be a major contribution, especially in the case of 3D visibility.

	Object space	**Image space**	**Viewpoint space**	**Line space**
view computation	BSP			
Hard shadow	shadow volume shadow BSP	shadow map shadow cache		
Soft shadows	limits of penumbra	sampling convolution	complete discontinuity mesh	antipenumbra
Occlusion culling	use of frusta architectural scenes from a volume	hierarchical z-buffer shadow footprints	temporal coherence use of the aspect graph	line-space subdivision
Ray-tracing	bounding volumes space subdivision beam-tracing	item and light buffer ZZ-buffer		ray classification
Radiosity	discontinuity mesh shaft culling volumetric visibility architectural scenes	hemicube convolution	complete discontinuity mesh	visibility skeleton global Monte Carlo Plücker for blockers
Image-based		epipolar rendering	silhouette tree	lumigraph, light-field
Object recognition			aspect graph	asp
Contour intersection	visual hull			
Sensor placement	viewpoint constraint best-next view		viewpoint optimization art gallery	
Motion planning	visibility graph	use of the z-buffer	pursuit-evasion self localisation target tracking	

Table 10.1: Recapitulation of the techniques presented by field and by space.

CHAPTER 11

Some Notions in Line Space

Plücker coordinates

Consider a *directed* line ℓ in 3D defined by two points $P(x_P, y_P, z_P)$ and $Q(x_Q, y_Q, z_Q)$. The *Plücker coordinates* [Plü65] of ℓ are:

$$\begin{pmatrix} \pi_{\ell 0} \\ \pi_{\ell 1} \\ \pi_{\ell 2} \\ \pi_{\ell 3} \\ \pi_{\ell 4} \\ \pi_{\ell 5} \end{pmatrix} = \begin{pmatrix} x_P y_Q - y_P x_Q \\ x_P z_Q - z_P x_Q \\ x_P - x_Q \\ y_P z_Q - z_P y_Q \\ z_P - z_Q \\ y_Q - y_P \end{pmatrix}$$

(The signs and order may vary with the authors). These coordinates are homogenous, any choice of P and Q will give the same Plücker coordinates up to a scaling factor (Plücker space is thus a 5D projective space).

The dot product between two lines a and b with Plücker duals Π_a and Π_b is defined by

$$\Pi_a \odot \Pi_b = \pi_{a0}\pi_{b4} + \pi_{a1}\pi_{b5} + \pi_{a2}\pi_{b3} + \pi_{a4}\pi_{b0} + \pi_{a5}\pi_{b1} + \pi_{a3}\pi_{b2}$$

The sign of the dot products indicates the relative orientation of the two lines. If the dot product is null, the two lines intersect. The equation $\Pi_a \odot \Pi_\ell = 0$ defines the hyperplane associated with a.

The *Plücker hypersurface* or *Grassman manifold* or *Klein quadric* is defined by

$$\Pi_\ell \odot \Pi_\ell = 0$$

CHAPTER 12

Online Ressources

1 General ressources

An index of computer graphics web pages can be found at
    http://www-imagis.imag.fr/~Fredo.Durand/book.html

A lot of computer vision ressources are listed at
    http://www.cs.cmu.edu/ cil/vision.html
    A commented and sorted vision bibliography:
    http://iris.usc.edu/Vision-Notes/bibliography/contents.html
    An excellent Compendium of Computer Vision:
    http://www.dai.ed.ac.uk/CVonline/

For robotics related pages, see
    http://www-robotics.cs.umass.edu/robotics.html
    http://www.robohoo.com/

Many sites are dedicated to computational geometry, *e.g.*:
    http://www.ics.uci.edu/~eppstein/geom.html
    http://compgeom.cs.uiuc.edu/~jeffe/compgeom/

Those interested in human and animal vision will find several links at:
    http://www.visionscience.com/.

An introduction to perception is provided under the form of an excellent web book at:
    http://www.yorku.ca/eye/

2 Available code.

CGAL is a robust and flexible computational geometry librairy
    http://www.cs.ruu.nl/CGAL

Nina Amenta maintains some links to geometrical softwares:
http://www.geom.umn.edu/software/cglist/welcome.html

The implementation of Luebke and George's online portal occlusion-culling technique is available at:
http://www.cs.virginia.edu/˜luebke/visibility.html

Electronic articles on shadows, portals, etc.:
http://www.flipcode.com/features.htm

Information on Open GL, including shadow computation:
http://reality.sgi.com/opengl/

Visibility graph programs can be found at:
http://www.cs.uleth.ca/˜wismath/vis.html
http://cs.smith.edu/˜halef/research.html
http://willkuere.informatik.uni-wuerzburg.de/ lupinho/java.html

Many ray-tracer are available *e.g.*:
http://www.povray.org/
http://www-graphics.stanford.edu/- cek/rayshade/rayshade.html
http://www.rz.tu-ilmenau.de/˜juhu/GX/intro.html (with different acceleration schemes, including ray-classification)

A radiosity implementation:
http://www.ledalite.com/software/software.htm

RenderPark provides many global illumination methods, such as radiosity or Monte-Carlo path-tracing:
http://www.cs.kuleuven.ac.be/cwis/research-/graphics/RENDERPARK/

Aspect graphs:
http://www.dai.ed.ac.uk/staff/-personal_pages/eggertd/software.html

BSP trees:
http://www.cs.utexas.edu/users/atc/

A list of info and links about BSP:
http://www.ce.unipr.it/ marchini/jaluit.html

Mel Slater's shadow volume BSP:
ftp://ftp.dcs.qmw.ac.uk/people/mel/BSP/

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INDEX

REFERENCES

[AA95] Steven Abrams and Peter K. Allen. Swept volumes and their use in viewpoint computation in robot work-cells. In *Proceedings IEEE International Symposium on Assembly and Task Planning*, August 1995. http://www.cs.columbia.edu/˜abrams/. (cited on page 95)

[AAA$^+$92] A. L. Abbott, N. Ahuja, P.K. Allen, J. Aloimonos, M. Asada, R. Bajcsy, D.H. Ballard, B. Bederson, R. M. Bolle, P. Bonasso, C. M. Brown, P. J. Burt, D. Chapman, J. J. Clark, J. H. Connell, P. R. Cooper, J. D. Crisman, J. L. Crowley, M. J. Daily, J.O. Eklundh, F. P. Firby, M. Herman, P. Kahn, E. Krotkov, N. da Vitoria Lobo, H. Moraff, R. C. Nelson, H. K. Nishihara, T. J. Olson, D. Raviv, G. Sandini, and E. L. Schwartz. Promising directions in active vision. *International Journal on Computer Vision*, 1992. (cited on page 16)

[AB91] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. *M. Landy and J. A. Movshon, (eds) Computational Models of Visual Processing*, 1991. (cited on page 83)

[Abr96] Michael Abrash. Inside quake: Visible-surface determination. *Dr. Dobb's Journal of Software Tools*, 21(??):41–??, January/February 1996. (cited on page 42)

[ACW$^+$99] D. Aliaga, J. Cohen, A. Wilson, Eric Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. Mmr: An interactive massive model rendering system using geometric and image-based acceleration. In *1999 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, April 1999. http://www.cs.unc.edu:80/˜walk/research/index.html. (cited on page 61)

[AEG98] Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-98)*, pages 107–116, New York, January 25–27 1998. ACM Press. (cited on pages 41, 42, 95)

[Aga84] P. K. Agarwal. *The art gallery problem: Its Variations, applications, and algorithmic aspects*. PhD thesis, Johns Hopkins University, Baltimore, 1984. (cited on page 74)

[AGMV97] P. Agarwal, L. Guibas, T. Murali, and J. Vitter. Cylindrical static and kinetic binary space partitions. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 39–48, New York, June 4–6 1997. ACM Press. (cited on pages 41, 42)

[AH97] Laurent Alonso and Nicolas Holzschuch. Using graphics hardware to speed up your visibility queries. Technical Report 99-R-030, LORIA, March 1997. http://www.loria.fr. (cited on page 60)

[Air90] John M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, Dept. of CS, U. of North Carolina, July 1990. TR90-027. (cited on pages 54, 55)

[AK87] James Arvo and David Kirk. Fast ray tracing by ray classification. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21-4, pages 55–64, July 1987. (cited on page 81)

[AK89] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. In Andrew S. Glassner, editor, *An introduction to ray tracing*, pages 201–262. Academic Press, 1989. (cited on pages 11, 40)

[Ama84] John Amanatides. Ray tracing with cones. *Computer Graphics*, 18(3):129–135, July 1984. (cited on page 47)

[App67] Arthur Appel. The notion of quantitative invisibility and the machine rendering of solids. *Proc. ACM Natl. Mtg.*, page 387, 1967. (cited on page 32)

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968. (cited on pages 10, 36)

[ARB90] John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24-2, pages 41–50, March 1990. (cited on page 55)

[Arn69] V.I. Arnold. Singularities of smooth mappings. *Russian Mathematical Surveys*, pages 3–44, 1969. (cited on page 28)

[Arv94] James Arvo. The irradiance Jacobian for partially occluded polyhedral sources. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 343–350. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0. (cited on pages 32, 50)

[Ase92] Frederic Asensio. A hierarchical ray-casting algorithm for radiosity shadows. *Third Eurographics Workshop on Rendering*, pages 179–188, May 1992. (cited on page 92)

[Ash94] Ian Ashdown. *Radiosity: A Programmer's Perspective*. John Wiley & Sons, New York, NY, 1994. (cited on page 11)

[ATS94] James Arvo, Kenneth Torrance, and Brian Smits. A Framework for the Analysis of Error in Global Illumination Algorithms. In *Computer Graphics Proceedings, Annual Conference Series, 1994 (ACM SIGGRAPH '94 Proceedings)*, pages 75–84, 1994. (cited on page 92)

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In G. Marechal, editor, *Eurographics '87*, pages 3–10. North-Holland, August 1987. (cited on page 40)

[AWG78] P. Atherton, K. Weiler, and D. Greenberg. Polygon shadow generation. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12(3), pages 275–281, August 1978. (cited on pages 32, 46)

[Bax95] Michael Baxandall. *Shadows and enlightenment*. Yale University Press, London, UK, 1995. (Ombres et Lumières, Gallimard). (cited on page 8)

[BB82] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice Hall, Englewood Cliffs, 2 edition, 1982. (cited on page 13)

[BB84] Lynne Shapiro Brotman and Norman I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):5–12, October 1984. (cited on page 45)

[BD98] Amy J. Briggs and Bruce R. Donald. Visibility-based planning of sensor control strategies. *Algorithmica*, 1998. accepted for publication, http://www.middlebury.edu/~briggs/Research/alg.html. (cited on page 51)

[BDEG90] Marshall Bern, David Dobkin, David Eppstein, and Robert Grossman. Visibility with a moving point of view. In David Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 107–117, San Francisco, CA, USA, January 1990. SIAM. (cited on page 85)

[Ber86] P. Bergeron. A general version of Crow's shadow volumes. *IEEE Computer Graphics and Applications*, 6(9):17–28, 1986. (cited on page 45)

[Ber93] M. De Berg. Ray shooting, depth orders and hidden surface removal. In *Lecture Notes in Computer Science*, volume 703. Springer Verlag, New York, 1993. (cited on pages 7, 19, 34, 37)

[BEW$^+$98] Lars Bishop, Dave Eberly, Turner Whitted, Mark Finch, and Michael Shantz. Designing a PC game engine. *IEEE Computer Graphics and Applications*, 18(1):46–53, January/February 1998. (cited on page 48)

[BF89] Chris Buckalew and Donald Fussell. Illumination networks: Fast realistic rendering with general reflectance functions. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 89–98, July 1989. (cited on page 88)

[BGRT95] P. Bose, F. Gomez, P. Ramos, and G. Toussaint. Drawing nice projections of objects in space. *Lecture Notes in Computer Science*, 1027:52–??, 1995. (cited on page 70)

[BHS98] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In IEEE Computer Society Press, editor, *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 327–335, June 1998. (cited on page 48)

[BK70] W. J. Bouknight and K. C. Kelly. An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources. In *Proc. AFIPS JSCC*, volume 36, pages 1–10, 1970. (cited on page 57)

[Bli78] J. F. Blinn. A scan line algorithm for displaying parametrically defines surfaces. *Computer Graphics (Special SIGGRAPH '78 Issue, preliminary papers)*, pages 1–7, August 1978. (cited on page 36)

[BM98] Mark R. Bolin and Gary W. Meyer. A perceptually based adaptive sampling algorithm. In Michael F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings- 1998, pages 299–310, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison-Wesley. (cited on page 92)

[BMT98] D. Bartz, M. Meissner, and T.Huettner. Extending graphics hardware for occlusion queries in opengl. In *Eurographics/Siggraph Workshop on Graphics Hardware Lisbon, Portugal August 31 and September 1*, 1998. (cited on page 62)

[BNN$^+$98] Ph. Bekaert, L. Neumann, A. Neumann, M. Sbert, and Y.D. Willems. Hierarchical monte carlo radiosity. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, New York City, NY, June 1998. Eurographics, Springer Wein. (cited on page 88)

[Bou70] W. Jack Bouknight. A procedure for generation of three-dimensional half-toned computer graphics presentations. *Communications of the ACM*, 13(9):527–536, September 1970. (cited on page 36)

[Bou85] C. Bouville. Bounding ellipsoids for ray-fractal intersection. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 45–52, July 1985. (cited on page 40)

[BP96] Normand Brière and Pierre Poulin. Hierarchical view-dependent structures for interactive scene manipulation. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 83–90. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. (cited on page 54)

[Bre92] M. Quinn Brewster. *Thermal Radiative Transfer & Properties*. John Wiley & Sons, New York, 1992. (cited on page 11)

[BRW89] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 325–334, July 1989. (cited on pages 32, 60)

[BS96] Gonzalo Besuievsky and Mateu Sbert. The multi-frame lighting method: A monte carlo based solution for radiosity in dynamic environments. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 185–194, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4. (cited on page 95)

[BWCG86] Daniel R. Baum, John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. The Back-Buffer Algorithm: An Extension of the Radiosity Method to Dynamic Environments. *The Visual Computer*, 2(5):298–306, September 1986. (cited on page 95)

[BY98] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*. Cambridge University Press, 1998. (cited on pages 19, 66)

[BZW$^+$95] J.E. Banta, Y. Zhien, X.Z. Wang, G. Zhang, M.T. Smith, and M.A. Abidi. A “best-next-view” algorithm for three dimensional scene reconstruction using range images. In *Computer Graphics (SIGGRAPH ’87 Proceedings)SPI Intelligent Robots and Computer Vision XIV. Algorithms, Techniques, Active Vision and Material Handling, Philadelphia, PA*, 1995. (cited on page 48)

[Cam91] A. T. Campbell, III. *Modeling Global Diffuse Illumination for Image Synthesis*. PhD thesis, CS Dept, University of Texas at Austin, December 1991. Tech. Report TR-91-39, http://www.cs.utexas.edu/users/atc/. (cited on pages 46, 50, 51, 54)

[Can88] John F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Massachusetts, 1988. (cited on page 42)

[Car84] Loren Carpenter. The A-buffer, an antialiased hidden surface method. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH ’84 Proceedings)*, volume 18, pages 103–108, July 1984. (cited on pages 35, 36)

[Cat74] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.d. thesis, University of Utah, December 1974. (cited on page 35)

[Cat75] Edwin E. Catmull. Computer display of curved surfaces. In *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pages 11–17, May 1975. (cited on page 35)

[Cat78] E. Catmull. A hidden-surface algorithm with anti-aliasing. In *Computer Graphics (SIGGRAPH ’78 Proceedings)*, volume 12(3), pages 6–11, August 1978. (cited on page 35)

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH ’87 Proceedings)*, volume 21(4), pages 95–102, July 1987. (cited on page 36)

[CCD91] J. Chapman, T. W. Calvert, and J. Dill. Spatio-temporal coherence in ray tracing. In *Proceedings of Graphics Interface ’91*, pages 101–108, June 1991. (cited on page 95)

[CCOL98] Yiorgos Chrysanthou, Daniel Cohen-Or, and Dani Lischinski. Fast approximate quantitative visibility for complex scenes. In *Proceedings of Computer Graphics International*, June 1998. http://www.math.tau.ac.il/˜daniel/. (cited on page 83)

[CDL$^+$96] Bradford Chamberlain, Tony DeRose, Dani Lischinski, David Salesin, and John Snyder. Fast rendering of complex environments using a spatial hierarchy. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface ’96*, pages 132–141. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996. ISBN 0-9695338-5-3. (cited on page 41)

[CDP95] Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes. *Computer Graphics Forum*, 14(3):371–382, August 1995. Proceedings of Eurographics ’95. ISSN 1067-7055. (cited on page 40)

[CEG$^+$92] B. Chazelle, H. Edelsbrunner, L. J. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting cycles of lines and rods in space. *Comput. Geom. Theory Appl.*, 1:305–323, 1992. (cited on page 87)

[CEG$^+$96] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15(5):428–447, May 1996. (cited on page 87)

[CF89] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH ’89 Proceedings)*, volume 23(3), pages 99–106, July 1989. (cited on page 46)

[CF90] A. T. Campbell, III and Donald S. Fussell. Adaptive Mesh Generation for Global Diffuse Illumination. In *Computer Graphics (ACM SIGGRAPH ’90 Proceedings)*, volume 24-4, pages 155–164, August 1990. http://www.cs.utexas.edu/users/atc/. (cited on page 46)

[CF91a] A. T. Campbell III and Donald S. Fussell. An analytic approach to illumination with area light sources. Technical Report CS-TR-91-25, University of Texas, Austin, August 1, 1991. http://www.cs.utexas.edu/users/atc/. (cited on page 50)

[CF91b] S. Chen and H. Freeman. On the characteristic views of quadric-surfaced solids. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 34–43, June 1991. (cited on page 69)

[CF92] Norman Chin and Steven Feiner. Fast object-precision shadow generation for area light source using BSP trees. In Marc Levoy and Edwin E. Catmull, editors, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 21–30, Cambridge, MA, March–April 1992. ACM Press. (cited on page 50)

[CG85] M. F. Cohen and D. P. Greenberg. The hemicube: A radiosity solution for complex environments. In *Proceedings of SIGGRAPH '85*, volume 19(3), pages 31–40, San Francisco, CA, 22–26 July 1985. ACM SIGGRAPH. (cited on page 60)

[Cha96] B. Chazelle. The computational geometry impact task force report: An executive summary. *Lecture Notes in Computer Science*, 1148:59–??, 1996. http://www.cs.princeton.edu/˜chazelle/. (cited on page 19)

[Chv75] V. Chvátal. A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B*, 18:39–41, 1975. (cited on page 74)

[CK88] Cregg K. Cowan and Peter D. Kovesi. Automatic sensor placement from vision task requirements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-10(3):407–416, May 1988. (cited on page 51)

[CL97] D. Crevier and R. Lepage. Knowledge-based image understanding systems: A survey. *Computer Vision and Image Understanding: CVIU*, 67(2):161–??, ???? 1997. (cited on page 13)

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976. (cited on pages 9, 48)

[Cla79] James H. Clark. A fast scan-line algorithm for rendering parametric surfaces. *Computer Graphics, Special SIGGRAPH '79 Issue*, 13(2):7–11, August 1979. (cited on page 36)

[Cla87] K. L. Clarkson. New applications of random sampling to computational geometry. *Discrete and Computational Geometry*, 2:195–222, 1987. (cited on page 87)

[CLF98] Emilio Camahort, Apostolos Lerios, and Don Fussell. Uniformly sampled light fields. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, New York City, NY, June 1998. Eurographics, Springer Wein. (cited on page 83)

[CLR80] Elaine Cohen, Tom Lyche, and Richard F. Riesenfeld. Discrete B-spline subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14:87–111, October 1980. (cited on page 35)

[CLSS97] Per H. Christensen, Dani Lischinski, Eric J. Stollnitz, and David H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, January 1997. ISSN 0730-0301. (cited on page 41)

[COFHZ98] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for visibility partitionning of densely occluded scenes. In *Proceedings of Eurographics*, September 1998. http://www.math.tau.ac.il/˜daniel/. (cited on page 54)

[Col75] George Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Berlin, 1975. Springer-Verlag. (cited on page 69)

[COZ98] Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. In *Proceedings of Graphics Interface*, June 1998. http://www.math.tau.ac.il/˜daniel/. (cited on pages 9, 54)

[CP97] Frédéric Cazals and Claude Puech. Bucket-like space partitioning data structures with applications to ray-tracing. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 11–20, New York, June 4–6 1997. ACM Press. (cited on page 40)

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In James George, editor, *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11(2), pages 242–248, July 1977. (cited on page 45)

[Cro84] Gary A. Crocker. Invisibility coherence for faster scan-line hidden surface algorithms. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 95–102, July 1984. (cited on page 36)

[CS92] Y. Chrysanthou and M. Slater. Computing dynamic changes to BSP trees. In A. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, volume 11-3, pages 321–332, September 1992. (cited on page 95)

[CS94] D. Cohen and Z. Sheffer. Proximity clouds: An acceleration technique for 3D grid traversal. *The Visual Computer*, 11?:27–38, 1994? (cited on page 40)

[CS95] Yiorgos Chrysanthou and Mel Slater. Shadow volume BSP trees for computation of shadows in dynamic scenes. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 45–50. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7. (cited on pages 46, 95)

[CS97] Yiorgos Chrysanthou and Mel Slater. Incremental updates to scenes illuminated by area light sources. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 103–114, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4. (cited on pages 58, 95)

[CT96] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility. In *Proceedings of the Twelfth Annual Symposium On Computational Geometry (ISG '96)*, pages 78–87, New York, May 1996. ACM Press. http://graphics.lcs.mit.edu/˜satyan/pubs.html. (cited on pages 48, 77)

[CT97a] Michael Capps and Seth Teller. Communications visibility in shared virtual worlds. In *Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 187–192. ACM Press, June18–2- 1997. http://graphics.lcs.mit.edu/˜capps/. (cited on page 9)

[CT97b] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders (color plate S. 189). In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 83–90, New York, April27–30 1997. ACM Press. http://graphics.lcs.mit.edu/˜satyan/pubs.html. (cited on page 48)

[CW85] J. Callahan and R. Weiss. A model for describing surface shape. In *Proceedings, CVPR '85 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, June 10–13, 1985)*, IEEE Publ. 85CH2145-1., pages 240–245. IEEE, IEEE, 1985. (cited on page 69)

[CW93a] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993. (cited on page 12)

[CW93b] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993. (cited on pages 9, 11)

[CW96] H.-M. Chen and W.-T. Wang. The feudal priority algorithm on hidden-surface removal. *Computer Graphics*, 30(Annual Conference Series):55–64, 1996. (cited on page 41)

[Dal96] Bengt-Inge L. Dalenbäck. Room acoustic predictiom based on a unified treatment of diffuse and specular reflection. *Journal of the Acoustical Society of America*, 100(2):899–909, August 1996. (cited on page 9)

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997. (cited on pages 19, 33, 66, 71, 74)

[DDP96] Frédo Durand, George Drettakis, and Claude Puech. The 3D visibility complex: A new approach to the problems of accurate visibility. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 245–256, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4, http://www-imagis.imag.fr/Publications/. (cited on page 81)

[DDP97a] F. Durand, G. Drettakis, and C. Puech. 3d visibility made visibly simple. In *video 13th Annu. ACM Sympos. Comput. Geom.*, 1997. (cited on page 83)

[DDP97b] Frédo Durand, George Drettakis, and Claude Puech. The 3d visibility complex: a unified data-structure for global visibility of scenes of polygons and smooth objects. In *Canadian Conference on Computational Geometry*, August 1997. http://www-imagis.imag.fr/˜Fredo.Durand. (cited on page 81)

[DDP97c] Fredo Durand, George Drettakis, and Claude Puech. The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. *Computer Graphics*, 31(3A):89–100, August 1997. http://www-imagis.imag.fr/Publications/. (cited on page 83)

[DDP99] Frédo Durand, George Drettakis, and Claude Puech. Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, April 1999. to appear. http://www-imagis.imag.fr/˜Fredo.Durand. (cited on pages 84, 92, 93)

[DF94] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. *Computer Graphics*, 28(Annual Conference Series):223–230, July 1994. http://www-imagis.imag.fr/˜George.Drettakis/pub.html. (cited on pages 72, 73)

[DKW85] Nou Dadoun, David G. Kirkpatrick, and John P. Walsh. The geometry of beam tracing. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 55–61, June 1985. (cited on page 46)

[Dor94] Susan Dorward. A survey of object-space hidden surface removal. *International Journal of Computational Geometry and Applications*, 4(3):325–362, 1994. (cited on pages 7, 19, 37)

[DORP96] Frédo Durand, Rachel Orti, Stéphane Rivière, and Claude Puech. Radiosity in flatland made visibly simple. In *video 12th Annu. ACM Sympos. Comput. Geom.*, 1996. (cited on page 81)

[DP95] Frédo Durand and Claude Puech. The visibility complex made visibly simple. In *video 11th Annu. ACM Sympos. Comput. Geom.*, 1995. (cited on page 80)

[DPR92] S. J. Dickinson, A. P. Pentland, and A. Rosenfeld. 3-D shape recovery using distributed aspect matching. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):174–198, February 1992. (cited on page 70)

[Dru87] M. Drumheller. Mobile robot localization using sonar. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-9(2):325–332, 1987. See also: S. B. Thesis, Dept. of Mechanical Engineering, MIT, 1984 and MIT AI Lab Memo BZ6, Mobile Robot Localization Osing Sonar. (cited on page 18)

[DS96] George Drettakis and François Sillion. Accurate visibility and meshing calculations for hierarchical radiosity. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 269–278, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4, http://www-imagis.imag.fr/˜George.Drettakis/pub.html. (cited on page 73)

[DS97] George Drettakis and François Sillion. Interactive update of global illumination using A line-space hierarchy. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 57–64. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7, http://www-imagis.imag.fr/˜George.Drettakis/pub.html. (cited on pages 54, 95)

[DSSD99] Xavier Decoret, Gernot Schaufler, Francois Sillion, and Julie Dorsey. Improved image-based impostors foraccelerated rendering. In *Eurographics'99*, August 1999. (cited on page 34)

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. (cited on page 12)

[Dub57] L. E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *Amer. J. Math.*, 79:497–516, 1957. (cited on page 43)

[Dür38] Albrecht Dürer. *Underweysung der Messung mit dem Zirckel und richtscheyd*. Nuremberg, 1538. (cited on pages 37, 46)

[Dur99] Frédo Durand. *3D Visibility, analysis and applications*. PhD thesis, U. Joseph Fourier, Grenoble, 1999. http://www-imagis.imag.fr. (cited on pages 58, 96)

[DW85] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 69–78, July 1985. (cited on page 36)

[DZ95] Steven M. Drucker and David Zeltzer. CamDroid: A system for implementing intelligent camera control. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 139–144. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7. (cited on page 13)

[EB90] David W. Eggert and Kevin W. Bowyer. Computing the orthographic projection aspect graph of solids of revolution. *Pattern Recognition Letters*, 11(11):751–763, November 1990. (cited on page 69)

[EB92] Shimon Edelman and Heinrich H. Bulthoff. Orientation dependence in the recognition of familiar and novel views of 3D objects. *Vision Research*, 32:2385–2400, 1992. http://eris.wisdom.weizmann.ac.il/- ˜edelman/abstracts.html#visres. (cited on page 13)

[EB93] D.W. Eggert and K.W. Bowyer. Computing the generalized aspect graph for objects with moving parts. *PAMI*, 15(6):605–610, June 1993. (cited on page 95)

[EBD92] David Eggert, Kevin Bowyer, and Chuck Dyer. Aspect graphs: State-of-the-art and applications in digital photogrammetry. In *Proceedings of the 17th Congress of the International Society for Photogrammetry and Remote Sensing, Part B5*, pages 633–645, 1992. (cited on page 66)

[EBD$^+$93] David W. Eggert, Kevin W. Bowyer, Charles R. Dyer, Henrik I. Christensen, and Dmitry B. Goldgof. The scale space aspect graph. *Pattern Analysis and Machine Intelligence*, 15(11):1114–1130, November 1993. (cited on page 93)

[EC90] Gershon Elber and Elaine Cohen. Hidden curve removal for free form surfaces. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24-4, pages 95–104, August 1990. (cited on page 32)

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, 1987. (cited on pages 19, 66)

[EG86] Herbert Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 389–403, Berkeley, California, 28–30 May 1986. (cited on page 43)

[Egg91] D.W. Eggert. Aspect graphs of solids of revolution. In *Ph. D.*, 1991. (cited on page 69)

[EHW97] P. Eades, M. E. Houle, and R. Webber. Finding the best viewpoints for three-dimensional graph drawings. *Lecture Notes in Computer Science*, 1353:87–??, 1997. (cited on page 70)

[EK89] K. S. Eo and C. M. Kyung. Hybrid shadow testing scheme for ray tracing. *Computer-Aided Design*, 21(1):38–48, January/February 1989. (cited on page 45)

[ES94] R. Endl and M. Sommer. Classification of ray-generators in uniform subdivisions and octrees for ray tracing. *Computer Graphics Forum*, 13(1):3–19, March 1994. (cited on page 40)

[ESB95] D.W. Eggert, L. Stark, and K.W. Bowyer. Aspect graphs and their use in object recognition. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):347–375, 1995. (cited on page 70)

[Eve90] H. Everett. *Visibility Graph Recognition*. PhD thesis, Department of. Computer Science, University of Toronto, 1990. Tech. Report 231/90. (cited on page 20)

[FAG83] H. Fuchs, G. D. Abram, and E. D. Grant. Near real-time shaded display of rigid objects. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17(3), pages 65–72, July 1983. (cited on pages 41, 95)

[Fau93] O. Faugeras. *Three-dimensional computer vision*. MIT Press, Cambridge, MA, 1993. (cited on pages 13, 62)

[FCE$^+$98] Thomas Funkhouser, Ingrid Carlbom, Gary Elko, Gopal Pingali, Mohan Sondhi, and Jim West. A beam tracing approach to acoustic modeling for interactive virtual environments. In Michael F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings- 1998, pages 21–32, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison-Wesley. (cited on pages 9, 47)

[FD84] G. Fekete and L. S. Davis. Property spheres: a new representation for 3-D object recognition. In *Proceedings of the Workshop on Computer Vision: Representation and Control (Annapolis, MD, April 30-May 2, 1984)*, IEEE Publ. 84CH2014-9., pages 192–201. IEEE, IEEE, 1984. (cited on page 66)

[FF88] Alain Fournier and Donald Fussell. On the power of the frame buffer. *ACM Transactions on Graphics*, 7(2):103–128, 1988. (cited on page 58)

[FFR83] E. Fiume, A. Fournier, and L. Rudolph. A parallel scan conversion algorithm with anti-aliasing for a general purpose ultracomputer. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17(3), pages 141–150, July 1983. (cited on page 35)

[FGH$^+$85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 111–120, July 1985. (cited on page 46)

[FGK$^+$96] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schoenherr. The cgal kernel: A basis for geometric computation. in proceedings workshop on applied computational geometry. In *roceedings of the Workshop on Applied Computational Geometry, May 27-28, 1996, Philadelphia, Pennsylvania.*, 1996. http://www.cs.ruu.nl/CGAL. (cited on pages 19, 98)

[Fis78] S. Fisk. Short proof of chvatal's watchman theorem. *Journal of Combinatorial Theory*, 24:374, 1978. (cited on page 74)

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14(3), pages 124–133, July 1980. (cited on page 41)

[FL98] Patrick Fabiani and Jean-Claude Latombe. Tracking a partially predictable object with uncertainty and visibility constraints: a game-theoretic approach. Technical report, Univeristy of Stanford, December 1998. http://underdog.stanford.edu/. (cited on page 76)

[FMA$^+$92] Olivier Faugeras, Joe Mundy, Narendra Ahuja, Charles Dyer, Alex Pentland, Ramesh Jain, and Katsushi Ikeuchi. Why aspect graphs are not (yet) practical for computer vision. *Computer Vision and Image Understanding: CVIU*, 55(2):322–335, March 1992. (cited on pages 70, 93)

[FMC99] Thomas A. Funkhouser, Patrick Min, and Ingrid Carlbom. Real-time acoustic modeling for distributed virtual environments. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999. (cited on page 47)

[FPM98] L. De Floriani, E. Puppo, and P. Magillo. Geometric structures and algorithms for geographical information systems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook on Computational Geometry*. Elsevier Science, 1998. Preliminary version available as: Technical Report DISI-TR-97-08, Department of Computer and Information Sciences, University of Genova, http://www.disi.unige.it/person/DeflorianiL/. (cited on page 6)

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, August 1993. http://www.bell-labs.com/user/funk/. (cited on page 55)

[FST92] Thomas A. Funkhouser, Carlo H. Sequin, and Seth J. Teller. Management of large amounts of data in interactive building walkthroughs. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25-2, pages 11–20, March 1992. http://www.bell-labs.com/user/funk/. (cited on page 55)

[FTI86] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986. (cited on page 40)

[Fun95] T. Funkhouser. RING - A client-server system for multi-user virtual environments. *SIGGRAPH Symposium on Interactive 3D Graphics*, pages 85–92, 1995. (cited on pages 9, 55)

[Fun96a] T. Funkhouser. Network topologies for scalable multi-user virtual environments. *Proceedings of VRAIS'96, Santa Clara CA*, pages 222–229, 1996. (cited on page 9)

[Fun96b] Thomas A. Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 343–352. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. (cited on page 55)

[Fun96c] Thomas A. Funkhouser. Database management for interactive display of large architectural models. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 1–8. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996. ISBN 0-9695338-5-3, http://www.bell-labs.com/user/funk. (cited on pages 9, 55)

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Co., Reading, MA, 2nd edition, 1990. T 385.C587. (cited on pages 7, 9, 24, 31, 40, 61)

[GCS91] Ziv Gigus, John Canny, and Raimund Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. on Pat. Matching & Mach. Intelligence*, 13(6), June 1991. (cited on pages 67, 72, 83)

[GGC97] X. Gu, S. J. Gortler, and M. Cohen. Polyhedral geometry and the two-plane parameterization. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, New York City, NY, June 1997. Eurographics, Springer Wein. ISBN 3-211-83001-4, http://hillbilly.deas.harvard.edu/˜sjg/. (cited on page 83)

[GGH+99] X. Gu, S.J. Gortler, H. Hoppe, L. Mcmillan, B. Brown, and A. Stone. Silhouette mapping. Technical Report TR-1-99, Harvard Computer Science, March 1999. http://cs.harvard.edu/˜xgu/paper/Silhouette_Map/. (cited on page 70)

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996, http://hillbilly.deas.harvard.edu/˜sjg/. (cited on page 83)

[GH94] Neil Gatenby and W. T. Hewitt. Optimizing Discontinuity Meshing Radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 249–258, Darmstadt, Germany, June 1994. (cited on page 50)

[GH96] S. Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, 1996. ISSN 0167-7055. (cited on page 92)

[GH97] S. Gibson and R. J. Hubbold. Perceptually-driven radiosity. *Computer Graphics Forum*, 16(2):129–141, 1997. ISSN 0167-7055. (cited on page 92)

[GH98] Djamchid Ghazanfarpour and Jean-Marc Hasenfratz. A beam tracing with precise antialiasing for polyhedral scenes. *Computer & Graphics*, 22(1):103–115, 1998. (cited on page 47)

[Gho97] Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *GEOMETRY: Discrete & Computational Geometry*, 17, 1997. (cited on page 20)

[GI87] Keith D. Gremban and Katsushi Ikeuchi. Planning multiple observation for object recognition. *International Journal of Computer Vision*, 1(2):145–65, 1987. (cited on page 70)

[GKM93] Ned Greene, Michael Kass, and G.avin Miller. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993. (cited on page 61)

[GL99] Héctor González-Baños and Jean-Claude Latombe. Planning robot motions for range-image acquisition and automatic 3d model construction. In *AAAI Spring Symposium*, 1999. (cited on page 75)

[Gla84] Andrew S. Glassner. Space sub-division for fast ray tracing. *IEEE Computer Graphics and Applications, October 1984*, 4:15–22, 1984. (cited on page 40)

[Gla88] Andrew S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, March 1988. (cited on page 95)

[Gla89] Andrew Glassner. *An introduction to raytracing*. Academic Press, Reading, MA, 1989. (cited on page 10)

[Gla95] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, CA, 1995. (cited on page 9)

[GLL+97] Leonidas J. Guibas, Jean-Claude Latombe, Steven M. LaValle, David Lin, and Rajeev Motwani. Visibility-based pursuit-evasion in a polygonal environment. In *Algorithms and Data Structures, 5th International Workshop*, Lecture Notes in Computer Science, Halifax, Nova Scotia, Canada, 6–8 August 1997. Springer. http://underdog.stanford.edu/. (cited on page 71)

[GLLL98] L. J. Guibas, J.-C. Latombe, S. M. LaValle, and D. Lin. Visibility-based pursuit-evasion in a polygonal environment. *Lecture Notes in Computer Science*, 1272:17–??, 1998. (cited on page 71)

[GM90] Ziv Gigus and Jitendra Malik. Computing the aspect graph for the line drawings of polyhedral objects. *IEEE Trans. on Pat. Matching & Mach. Intelligence*, 12(2), February 1990. (cited on pages 67, 68, 69, 80)

[GM91] Subir Kumar Ghosh and David M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, October 1991. (cited on page 43)

[GMR95] Guibas, Motwani, and Raghavan. The robot localization problem. In Goldberg, Halperin, Latombe, and Wilson, editors, *Algorithmic Foundations of Robotics, The 1994 Workshop on the Algorithmic Foundations of Robotics, A. K. Peters*, 1995. (cited on pages 70, 71)

[GN71] Robert A. Goldstein and Roger Nagel. 3-D visual simulation. *Simulation*, 16(1):25–31, January 1971. (cited on page 36)

[Goa83] Chris Goad. Special purpose automatic programming for 3D model-based vision. In Martin A Fischler Oscar Firschein, editor, *Readings in Computer Vision*, pages 371–381. Morgan Kaufman Publishers, August 1983. (cited on page 66)

[GP91] E. Gröller and W. Purgathofer. Using temporal and spatial coherence for accelerating the calculation of animation sequences. In Werner Purgathofer, editor, *Eurographics '91*, pages 103–113. North-Holland, September 1991. (cited on page 95)

[Gra92] Charles W. Grant. *Visibility Algorithms in Image Synthesis*. PhD thesis, U. of California, Davis, 1992. http://www.hooked.net/˜grant/. (cited on pages 7, 22, 31, 59, 63)

[Gre96] Ned Greene. Hierarchical polygon tiling with coverage masks. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 65–74. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. (cited on page 61)

[GS96] Sherif Ghali and A. James Stewart. A complete treatment of D1 discontinuities in a discontinuity mesh. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 122–131. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996. ISBN 0-9695338-5-3, http://www.dgp.toronto.edu/people/ghali/papers/. (cited on page 73)

[GSHG98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The irradiance volume. *IEEE Computer Graphics & Applications*, 18(2):32–43, 1998. (cited on page 83)

[Gue98] Concettina Guerra. Vision and image processing algorithms. In M. Atallah, editor, *CRC Handbook of Algorithms and Theory of Computation*. CRC Press, 1998. (cited on page 13)

[HA90] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 309–318, August 1990. (cited on page 59)

[Hae90] Paul Haeberli. Paint by numbers: Abstract image representations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 207–214, August 1990. (cited on page 58)

[Hai] Eric Haines. Ray tracing news. http://www.povray.org/rtn/. (cited on page 10)

[Hai91] Eric A. Haines. Beams O' Light: Confessions of a Hacker. In *SIGGRAPH '91 Course Notes - Frontiers in Rendering*. ACM, July 1991. (cited on page 46)

[Hal98] Michael Halle. Multiple viewpoint rendering. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 243–254. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8. (cited on page 83)

[Han89] Pat Hanrahan. A survey of ray-surface intersection algorithms. In Andrew S. Glassner, editor, *An introduction to ray tracing*, pages 79–119. Academic Press, 1989. (cited on page 36)

[Has98] Jean-Marc Hasenfratz. *Lancer de Faisceaux en Synthèse d'Image*. PhD thesis, Université de Limoges, 1998. (cited on page 47)

[HCS96] Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. (cited on page 13)

[HDG99] David Hart, Philip Dutré, and Donald P. Greenberg. Direct illumination with lazy visibility evaluation. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999. (cited on page 64)

[Hec92a] Paul Heckbert. Discontinuity meshing for radiosity. *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992. http://www.cs.cmu.edu/ ph. (cited on page 50)

[Hec92b] Paul S. Heckbert. Radiosity in flatland. In A. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, volume 11(3), pages 181–192, September 1992. (cited on page 50)

[HG86] Eric A. Haines and Donald P. Greenberg. The light buffer: A ray tracer shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, September 1986. (cited on pages 59, 63)

[HG94] P. Heckbert and M. Garland. Multiresolution modelling for fast rendering. *Proceedings of Graphics Interface'94*, pages 43–50, 1994. (cited on page 9)

[HH84] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18(3), pages 119–127, July 1984. (cited on page 46)

[HH89] Charles Hansen and Thomas C. Henderson. CAGD-based computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(11):1181–1193, November 1989. (cited on page 70)

[HH96] Seth Hutchinson and Gregory D. Hager. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, October 1996. (cited on page 15)

[HH97] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, CS Dept., Carnegie Mellon U., January 1997. http://www.cs.cmu.edu/˜ph. (cited on page 59)

[HK85] M. Hebert and T. Kanade. The 3-D profile method for object recognition. In *Proceedings, CVPR '85 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, June 10–13, 1985)*, IEEE Publ. 85CH2145-1., pages 458–463. IEEE, IEEE, 1985. (cited on page 66)

[HK89] Seth A. Hutchinson and Avinash C. Kak. Planning sensing strategies in a robot work cell with multi-sensor capabilities. *IEEE Journal of Robotics and Automation*, 5(6):765–783, December 1989. (cited on page 70)

[HKL97] Dan Halperin, Lydia Kavraki, and Jean-Claude Latombe. Robotics. In J.E. Goodman and J. O'Rourke, editors, *CRC Handbook of Discrete and Computational Geometry*. CRC Press, 1997. http://robotics.stanford.edu/˜latombe/pub.html. (cited on page 16)

[HKL98] D. Halperin, L.E. Kavraki, and J.C. Latombe. Robot algorithms. In M. Atallah, editor, *CRC Handbook of Algorithms and Theory of Computation*. CRC Press, 1998. http://robotics.stanford.edu/˜latombe/pub.html. (cited on pages 15, 16, 17)

[HLW96] V. Hlavac, A. Leonardis, and T. Werner. Automatic selection of reference views for image-based scene representations. *Lecture Notes in Computer Science*, 1064:526–??, 1996. (cited on page 75)

[HM96] André Hinkenjann and Heinrich Müller. Hierarchical blocker trees for global visibility calculation. Research Report 621/1996, University of Dortmund, Universität Dortmund, 44221 Dortmund, Germany, August 1996. http://ls7-www.cs.uni-dortmund.de/˜hinkenja/. (cited on page 82)

[HMC$^+$97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 1–10, New York, June4–6 1997. ACM Press. http://www.cs.unc.edu/ hudson/projects/occlusion/scg97.ps. (cited on page 48)

[HMK$^+$97] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 27–34. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. (cited on page 48)

[Hof92] Georg Rainer Hofmann. Who invented ray tracing? a historical remark. *The Visual Computer*, 9(1):120–125, 1992. (cited on page 37)

[HS99] Nicolas Holzschuch and Franois X. Sillion. An exhaustive error-bounding algorithm for hierarchical radiosity. *Computer Graphics Forum*, 1999. to appear, http://www.loria.fr/˜holzschu. (cited on page 92)

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991. (cited on pages 91, 92)

[HSD94] Nicolas Holzschuch, Francois Sillion, and George Drettakis. An Efficient Progressive Refinement Strategy for Hierarchical Radiosity. In *Fifth Eurographics Workshop on Rendering*, pages 343–357, Darmstadt, Germany, June 1994. http://www.loria.fr/˜holzschu. (cited on page 11)

[HT96] Stephen Hardt and Seth Teller. High-fidelity radiosity rendering at interactive rates. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 71–80, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4. (cited on page 50)

[HW91] Eric Haines and John Wallace. Shaft culling for efficient ray-traced radiosity. In *Eurographics Workshop on Rendering*, 1991. (cited on page 53)

[HW98] Michael E. Houle and Richard Webber. Approximation algorithms for finding best viewpoints. In Sue H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing*, number vol. 1547 in Lecture Notes in Computer Science, pages 210–223. Springer, Heidelberg, Germany, 1998. (cited on page 70)

[HWP97] David Hedley, Adam Worrall, and Derek Paddon. Selective culling of discontinuity lines. In J. Dorsey and P. Slusallek, editors, *Rendering Techniques '97*, pages 69–81, 8th EG workshop on Rendering, Saint Etienne, France, June 1997. Springer Verlag. (cited on page 93)

[HZ81] H. Hubschman and S. W. Zucker. Frame-to-frame coherence and the hidden surface computation: constraints for a convex world. *Computer Graphics*, 15(3):45–54, August 1981. (cited on page 76)

[HZ82] H. Hubschman and S. W. Zucker. Frame-to-frame coherence and the hidden surface computation: constraints for a convex world. *ACM Transactions on Graphics*, 1(2):129–162, April 1982. (cited on page 76)

[ICK$^+$99] Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999. (cited on page 58)

[Ike87] Katsushi Ikeuchi. Generating an interpretation tree from a CAD model for 3-D object recognition in bin-picking tasks. *International Journal of Computer Vision*, 1(2):145–65, 1987. (cited on page 70)

[JF93] A. K. Jain and P. J. Flynn. *Three Dimensional Object Recognition Systems*. Elsevier, Amsterdam, 1993. (cited on page 13)

[JK88] J. W. Jaromczyk and M. Kowaluk. Skewed projections with an application to line stabbing in R^3. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 362–370, New York, 1988. ACM, ACM Press. (cited on page 84)

[Jon71] C. B. Jones. A new approach to the 'hidden line' problem. *The Computer Journal*, 14(3):232–237, August 1971. (cited on page 48)

[JW89] David Jevans and Brian Wyvill. Adaptive voxel subdivision for ray tracing. In *Proceedings of Graphics Interface '89*, pages 164–172, June 1989. (cited on page 40)

[Kaj82] James T. Kajiya. Ray tracing parametric patches. In *Computer Graphics (SIGGRAPH '82 Proceedings)*, volume 16(3), pages 245–254, July 1982. (cited on page 36)

[Kaj86] J. T. Kajiya. The rendering equation. In David C. Evans and Rusell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20(4), pages 143–150, August 1986. (cited on page 9)

[KB98] D. Kriegman and P. Belhumeur. What shadows reveal about object structure. In *Proceedings European Conference on Computer Vision*, pages 399–414, 19998. http://www-cvr.ai.uiuc.edu/˜kriegman/. (cited on page 6)

[Kel97] Alexander Keller. Instant radiosity. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 49–56. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. (cited on page 59)

[Ker81] Y.L. Kergosien. La famille des projections orthogonales d'une surface et ses singularités. *C.R. Acad. Sc. Paris*, 292:929–932, 1981. (cited on pages 29, 69)

[KG79] Douglas S. Kay and Donald P. Greenberg. Transparency for computer synthesized images. In *Computer Graphics (SIGGRAPH '79 Proceedings)*, volume 13(3), pages 158–164, August 1979. (cited on page 36)

[Kir87] D. B. Kirk. The simulation of natural features using cone tracing. *The Visual Computer*, 3(2):63–71, August 1987. (cited on page 47)

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986. (cited on page 40)

[KKCS98] Bomjun Kwon, Dae Seoung Kim, Kyung-Yong Chwa, and Sung Yong Shin. Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics*, 4(3), July – September 1998. ISSN 1077-2626. (cited on page 82)

[KKMB96] D. Kersten, D.C. Knill, P. Mamassian, and I. Bülthoff. Illusory motion from shadow. *Nature*, 379(31), 1996. (cited on page 8)

[KM94] S. Krishnan and D. Manocha. Global visibility and hidden surface algorithms for free form surfaces. Technical Report TR94-063, UNC Chapel Hill, February 1994. http://www.cs.unc.edu/Research/tech-report.html. (cited on page 33)

[Kø84] J. J. Kœnderink. What does the occluding contour tell us about solid shape? *Perception*, 13:321–330, 1984. (cited on page 28)

[Koe87] J. J. Koenderink. An internal representation for solid shape based on the topological properties of the apparent contour. In W. Richards and S. Ullman, editors, *Image Understanding 1985–86*, pages 257–285, Norwood, NJ, 1987. Ablex. (cited on page 28)

[Koe90] Jan J. Koenderink. *Solid Shape*. MIT Press, Cambridge, Massachusetts, 1990. (cited on page 28)

[KP90] D. Kriegman and J. Ponce. Computing exact aspect graphs of curved objects: Solids of revolution. *International Journal of Computer Vision*, 5(2):119–135, 1990. (cited on page 69)

[KS97] Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, January 1997. bounding volume hierarchy with grids at each level & more. (cited on page 40)

[Kut91] H. Kutruff. *Room Acoustics (3rd edition)*. Elseiver Applied Science, 1991. (cited on page 9)

[Kv76] J.J. Koenderink and A.J. vanDoorn. The singularities of the visual mapping. *BioCyber*, 24(1):51–59, 1976. (cited on pages 28, 65)

[Kv79] J.J. Koenderink and A.J. vanDoorn. The internal representation of solid shape with respect to vision. *BioCyber*, 32:211–216, 1979. (cited on page 65)

[KvD82] Jan J. Kœnderink and Andrea J van Doorn. What does the occluding contour tell us about solid shape? *Perception*, 11:129–137, 1982. (cited on page 28)

[KWCH97] Kris Klimaszewski, Andrew Woo, Frederic Cazals, and Eric Haines. Additional notes on nested grids. *Ray Tracing News*, 10(3), December 1997. http://www.povray.org/rtn/. (cited on page 40)

[KYCS98] Kim, Yoo, Chwa, and Shin. Efficient algorithms for computing a complete visibility region in three-dimensional space. *Algorithmica*, 20, 1998. (cited on page 52)

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer, Dordrecht, The Netherlands, 1991. (cited on pages 16, 17, 69)

[Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. *T-PAMI*, 16:150–162, 1994. (cited on pages 44, 45)

[Lau95] A. Laurentini. How far 3d shapes can be understood from 2d silhouettes. *T-PAMI*, 17:188–195, 1995. (cited on page 44)

[Lau97] A. Laurentini. How many 2D silhouettes does it take to reconstruct a 3D object? *Computer Vision and Image Understanding: CVIU*, 67(1):81–??, ???? 1997. (cited on page 44)

[Lau99] A. Laurentini. Computing the visual hull of solids of revolution. *Pattern Recognition*, 32(3), 1999. (cited on pages 44, 45)

[LC79] Jeff Lane and Loren Carpenter. A generalized scan line algorithm for the computer display of parametrically defined surfaces. *Computer Graphics and Image Processing*, 11(3):290–297, November 1979. (cited on page 36)

[LCWB80] Jeffrey M. Lane, Loren C. Carpenter, J. Turner Whitted, and James F. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of the ACM*, 23(1):23–34, January 1980. (cited on page 36)

[LD97] Celine Loscos and George Drettakis. Interactive high-quality soft shadows in scenes with moving objects. *Computer Graphics Forum*, 16(3):C219–C230, September 4–8 1997. http://www-imagis.imag.fr/Publications/. (cited on page 96)

[LF94] Stephane Laveau and Olivier Faugeras. 3-D scene representation as a collection of images and fundamental matrices. Technical Report RR-2205, Inria, Institut National de Recherche en Informatique et en Automatique, 1994. (cited on page 13)

[LF96] Robert R. Lewis and Alain Fournier. Light-driven global illumination with a wavelet representation of light transport. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 11–20, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4. (cited on page 83)

[LG95] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7, http://www.cs.unc.edu/˜luebke/publications/portals.html. (cited on page 48)

[LGBL97] Steven LaValle, Hector H. González-Baños, Craig Becker, and Jean-Claude Latombe. Motion strategies for maintaining visibility of a moving target. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1997. (cited on page 76)

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996, http://www-graphics.stanford.edu/papers/light/. (cited on page 83)

[LH99] Steven M. LaValle and John E. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. In *IEEE International Conference on Robotics and Automation*, August 1999. http://janowiec.cs.iastate.edu/˜lavalle. (cited on page 72)

[LL86] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32:276–282, 1986. (cited on page 74)

[LLG$^+$97] Steven M. LaValle, David Lin, Leonidas J. Guibas, Jean-Claude Latombe, and Rajev Motwani. Finding an unpredictable target in a workspace with obstacles. In *IEEE International Conference on Robotics and Automation*, August 1997. http://janowiec.cs.iastate.edu/˜lavalle. (cited on page 71)

[LM98] M. Lin and D. Manocha. Applied computational geometry. In *Encyclopedia on Computer Science and Technology*. Marcel Dekker, 1998. http://www.cs.unc.edu/˜lin. (cited on page 19)

[LMW90] Bernd Lamparter, Heinrich Müller, and Jorg Winckler. The ray-z-buffer - an approach for ray-tracing arbitrarily large scenes. Technical Report 675/1998, University of Freiburg, Institut fur Mathematik, April 1990. (cited on page 82)

[LPW79] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979. (cited on page 42)

[LRDG90] Jed Lengyel, Mark Reichert, Bruce R. Donald, and Donald P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 327–335, August 1990. (cited on page 58)

[LS97] Jed Lengyel and John Snyder. Rendering with coherent layers. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 233–242. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. (cited on page 13)

[LSG94] Dani Lischinski, Brian Smits, and Donald P. Greenberg. Bounds and error estimates for radiosity. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 67–74. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0. (cited on page 92)

[LT99] Fei-Ah Law and Tiow-Seng Tan. Preprocessing occlusion for real-time selective refinement. In *1999 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, April 1999. (cited on page 45)

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992. http://www.cs.huji.ac.il/%7Edanix/publications.html. (cited on page 50)

[LTG93] D. Lischinski, F. Tampieri, and D. P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. *Computer Graphics*, 27(Annual Conference Series):199–208, 1993. http://www.cs.huji.ac.il/%7Edanix/publications.html. (cited on page 50)

[LW95] Eric P. Lafortune and Yves D. Willems. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 11–20, New York, NY, 1995. Springer-Verlag. (cited on page 83)

[LZ97] M. S. Langer and S. W. Zucker. Casting light on illumination: A computational model and dimensional analysis of sources. *Computer Vision and Image Understanding: CVIU*, 65(2):322–335, February 1997. (cited on page 81)

[MAG68] MAGI. 3-D simulated graphics offered by service bureau. *Datamation*, 14:69, February 1968. (cited on page 36)

[Mal87] Jitendra Malik. Interpreting line drawings of curved objects. *International Journal of Computer Vision*, 1:73–103, 1987. (cited on page 28)

[Max91] Nelson L. Max. Unified sun and sky illumination for shadows under trees. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 53(3):223–230, May 1991. (cited on page 63)

[May99] Using Maya, rendering. Alias Wavefront, (Maya user manual), 1999. (cited on page 59)

[MB93] J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(5):417–433, May 1993. (cited on page 48)

[MB95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995. (cited on pages 13, 62)

[MBGN98] Tom McReynolds, David Blythe, Brad Grantham, and Scott Nelson. Advanced graphics programming techniques using Open GL. Siggraph'1998 course notes, 1998. (cited on pages 46, 59)

[McK87] Michael McKenna. Worst-case optimal hidden-surface removal. *ACM Transactions on Graphics*, 6(1):19–28, January 1987. (cited on page 19)

[MDC93] Herve Maurel, Ives Duthen, and Rene Caubet. A 4D ray tracing. In R. J. Hubbold and R. Juan, editors, *Eurographics '93*, pages 285–294, Oxford, UK, 1993. Eurographics, Blackwell Publishers. (cited on page 95)

[Mey90] Urs Meyer. Hemi-cube ray-tracing: A method for generating soft shadows. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 365–376. North-Holland, September 1990. (cited on page 60)

[MG95] Scott O. Mason and Armin Grün. Automatic sensor placement for accurate dimensional inspection. *Computer Vision and Image Understanding: CVIU*, 61(3):454–467, May 1995. (cited on page 75)

[MGBY99] Yohai Makbily, Craig Gotsman, and Reuven Bar-Yehuda. Geometric algorithms for message filtering in decentralized virtual environments. In *1999 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, April 1999. (cited on page 9)

[MI98] Jun Miura and Katsushi Ikeuchi. Task-oriented generation of visual sensing strategies in assembly tasks. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 20(2):126–138, February 1998. also available as Carnegie Mellon University Technical Report CS-95-116. (cited on page 15)

[Mit87] Don P. Mitchell. Generating antialiased images at low sampling densities. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 65–72, July 1987. (cited on page 36)

[MKT$^+$97] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-time nonphotorealistic rendering. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 415–420. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. (cited on pages 8, 32)

[MO88] M. McKenna and J. O'Rourke. Arrangements of lines in 3-space: a data structure with applications. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 371–380, New York, 1988. ACM, ACM Press. (cited on page 84)

[MOK95] Karol Myszkowski, Oleg G. Okunev, and Tosiyasu L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995. ISSN 0178-2789. (cited on page 58)

[MS85] Matthew T. Mason and J. Kenneth Salisbury Jr. *Robot hands and the mechanics of manipulation*. Cambridge, Mass. : MIT Press, c1985, 298 p. (The MIT Press series in artificial intelligence) CALL NUMBER: TJ211 .M366 1985, 1985. (cited on page 86)

[Mue95] Carl Mueller. Architecture of image generators for flight simulators. Technical Report TR-95-015, UNC Chapel Hill, February 1995. http://www.cs.unc.edu/Research/tech-report.html. (cited on pages 7, 36)

[Mul89] Ketan Mulmuley. An efficient algorithm for hidden surface removal. In Jeffrey Lane, editor, *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics (SIGGRAPH '89)*, pages 379–388, Boston, MA, USA, July 1989. ACM Press. (cited on page 33)

[Mul91] Mulmuley. Hidden surface removal with respect to a moving view point (extended abstract). In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1991. (cited on pages 37, 42)

[Mun95] Olaf Munkelt. Aspect-trees: Generation and interpretation. *Computer Vision and Image Understanding: CVIU*, 61(3):365–386, May 1995. (cited on page 70)

[MWCF90] Joseph Marks, Robert Walsh, Jon Christensen, and Mark Friedell. Image and intervisibility coherence in rendering. In *Proceedings of Graphics Interface '90*, pages 17–30, May 1990. (cited on pages 33, 54)

[Mys98] Karol Myszkowski. The visible differences predictor: applications to global illumination problems. In *Eurographics Workshop on Rendering*, Vienna, Austria, June 1998. (cited on page 92)

[MZ92] J.L. Mundy and A. Zisserman. *Geometric Invariance in Computer Vision*. MIT press, Cambridge, MA, 1992. (cited on page 14)

[NAT90] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 115–124, August 1990. (cited on page 41)

[Nay92] Bruce F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, May 1992. (cited on page 42)

[Neu95] Laszlo Neumann. Monte Carlo Radiosity. *Computing*, 55(1):23–42, 1995. (cited on page 88)

[Ney96] Fabrice Neyret. Synthesizing verdant landscapes using volumetric textures. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 215–224, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4. (cited on page 41)

[Ney98] Fabrice Neyret. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, January 1998. (cited on page 41)

[Nil69] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In Donald E. Walker and Lewis M. Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, Washington, D. C., May 1969. William Kaufmann. (cited on page 42)

[NMN87] Tomoyuki Nishita, Yasuhiro Miyawaki, and Eihachiro Nakamae. A shading model for atmospheric scattering considering luminous intensity distribution of light sources. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH ’87 Proceedings)*, volume 21(4), pages 303–310, July 1987. (cited on page 46)

[NN83] Tomoyuki Nishita and Eihachiro Nakamae. Half-tone representation of 3-D objects illuminated by area or polyhedron sources. In *Proc. of IEEE Computer Society’s Seventh International Computer Software and Applications Conference (COMPSAC83)*, pages 237–242, November 1983. (cited on page 49)

[NN85] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH ’85 Proceedings)*, volume 19(3), pages 23–30, July 1985. (cited on pages 32, 49, 50)

[NNB97] L. Neumann, A. Neumann, and P. Bekaert. Radiosity with well distributed ray sets. *Computer Graphics Forum*, 16(3):261–270, August 1997. Proceedings of Eurographics ’97. ISSN 1067-7055. (cited on page 88)

[NNS72] M. E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. In *Proceedings of the ACM Annual Conference*, volume I, pages 443–450, Boston, Massachusetts, August 1972. (cited on page 33)

[NON85] T. Nishita, I. Okamura, and E. Nakamae. Shading models for point and linear sources. *ACM Transactions on Graphics*, 4(2):124–146, April 1985. (cited on page 49)

[NR95] Bruce Naylor and Lois Rogers. Constructing partitioning trees from bezier-curves for efficient intersections and visibility. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface ’95*, pages 44–55. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1995. ISBN 0-9695338-4-5. (cited on page 41)

[NSL99] C. Nissoux, T. Simeon, and J.P. Laumond. Visibility based probabilistic roadmaps. Technical Report 99057, LAAS, February 1999. doc@laas.fr. (cited on page 43)

[O’R87] J. O’Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987. (cited on pages 20, 74)

[O’R94] Joseph O’Rourke. *Computational geometry in C*. Cambridge University Press, 1994. (cited on pages 19, 66)

[ORDP96] Rachel Orti, Stéphane Rivière, Frédo Durand, and Claude Puech. Radiosity for dynamic scenes in flatland with the visibility complex. In Jarek Rossignac and François Sillion, editors, *Computer Graphics Forum (Proc. of Eurographics ’96)*, volume 16(3), pages 237–249, Poitiers, France, September 1996. (cited on page 81)

[OS97] J. O’Rourke and I. Streinu. Vertex-edge pseudo-visibility graphs: Characterization and recognition. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 119–128, New York, June 4–6 1997. ACM Press. (cited on page 20)

[OW88] M. H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana-Champaign, IL, June 6–8, 1988)*, pages 164–171, New York, 1988. ACM, ACM Press. (cited on page 43)

[PA91] Pierre Poulin and John Amanatides. Shading and shadowing with linear light sources. *Computers and Graphics*, 15(2):259–265, 1991. (cited on pages 59, 82)

[PBG92] Cary B. Phillips, Norman I. Badler, and John Granieri. Automatic viewing control for 3D direct manipulation. In Marc Levoy and Edwin E. Catmull, editors, *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pages 71–74, Cambridge, MA, March–April 1992. ACM Press. (cited on page 13)

[PD86] W. H. Plantinga and C. R. Dyer. An algorithm for constructing the aspect graph. In *27th Annual Symposium on Foundations of Computer Science*, pages 123–131, Los Angeles, Ca., USA, October 1986. IEEE Computer Society Press. (cited on page 66)

[PD87] H. Plantinga and C. R. Dyer. The asp: a continuous viewer-centered representation for 3D object recognition. In *First International Conference on Computer Vision, (London, England, June 8–11, 1987)*, pages 626–630, Washington, DC., 1987. IEEE Computer Society Press. (cited on page 79)

[PD90] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990. (cited on pages 68, 79, 80, 83)

[PDS90] Harry Plantinga, Charles R. Dyer, and W. Brent Seales. Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation. In *Proceedings of Graphics Interface '90*, pages 9–16, May 1990. (cited on page 70)

[Pel90] M. Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 177–186, 1990. (cited on page 87)

[Pel93] Marco Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993. (cited on page 87)

[Pel94] Pellegrini. On lines missing polyhedral sets in 3-space. *GEOMETRY: Discrete & Computational Geometry*, 12, 1994. (cited on page 87)

[Pel96] Marco Pellegrini. Repetitive hidden surface removal for polyhedra. *Journal of Algorithms*, 21(1):80–101, July 1996. http://www.imc.pi.cnr.it/˜marcop. (cited on page 37)

[Pel97a] Pellegrini. Monte carlo approximation of form factors with error bounded a priori. *Discrete & Computational Geometry*, 17, 1997. (cited on page 88)

[Pel97b] Marco Pellegrini. Ray-shooting and lines in space. In J.E. Goodman and J. O'Rourke, editors, *CRC Handbook of Discrete and Computational Geometry*, pages 599–614. CRC Press, 1997. http://www.imc.pi.cnr.it/˜marcop/. (cited on pages 20, 87)

[Pel99] Marco Pellegrini. A geometric approach to computing higher-order form factors. In *Proceedings of the 15th International Annual Symposium on Computational Geometry (SCG-99)*, New York, June 4–6 1999. ACM Press. (cited on pages 70, 88)

[Pet92] Sylvain PetitJean. Computing exact aspect graphs of smooth objects bounded by smooth algebraic surfaces. Master's thesis, University of Illinois, Urbana-Champaign, IL, June 1992. availabel as technical report UIUC-BI-AI-RCV-92-04. (cited on pages 28, 69)

[Pet95] Sylvain Petitjean. The number of views of piecewise-smooth algebraic objects. *Proceedings of Proceedings of the Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, 900:571–??, 1995. (cited on page 69)

[Pet96] Sylvain Petitjean. The enumerative geometry of projective algebraic surfaces and the complexity of aspect graphs. *International Journal of Computer Vision*, 19(3):1–27, 1996. (cited on page 69)

[Pet98] Sylvain Petitjean. A computational geometric approach to visual hulls. *International Journal of Computational Geometry and Applications*, 8(4):407–436, 1998. Special issue on applied computational geometry, edited by Ming Lin and Dinesh Manocha. http://www.loria.fr/ petitjea/. (cited on page 45)

[PF92] Pierre Poulin and Alain Fournier. Lights from highlights and shadows. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25(2), pages 31–38, March 1992. http://www.iro.umontreal.ca/labs/infographie/papers/. (cited on page 53)

[Pie93] Georg Pietrek. Fast Calculation of Accurate Formfactors. In *Fourth Eurographics Workshop on Rendering*, Series EG 93 RW, pages 201–220, Paris, France, June 1993. (cited on page 60)

[PJ91] Andrew Pearce and David Jevans. Exploiting shadow coherence in ray-tracing. In *Proceedings of Graphics Interface '91*, pages 109–116, June 1991. (cited on page 64)

[PK90] Jean Ponce and David J. Kriegman. Computing exact aspect graphs of curved objects: parametric surfaces. In William Dietterich, Tom; Swartout, editor, *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 1074–1081, Hynes Convention Centre?, July 29–August 3 1990. MIT Press. (cited on page 69)

[Pla88] William Harry Plantinga. *The asp: a continuous, viewer-centered object representation for computer vision*. PhD thesis, The University of Wisconsin, Madison, December 1988. (cited on page 79)

[Pla93] Harry Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3D scenes. In *Proceedings of Graphics Interface '93*, pages 166–173, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society. (cited on page 70)

[Plü65] J. Plücker. On a new geometry of space. *Phil. Trans. Royal Soc. London*, 155:725–791, 1865. (cited on pages 85, 101)

[Pop94] Arthur R. Pope. Model-based object recognition: A survey of recent research. Technical Report TR-94-04, University of British Columbia, Computer Science Department, January 1994. http://www.cs.ubc.ca/tr/1994/TR-94-04. (cited on page 13)

[Pot87] Michael Potmesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, October 1987. (cited on page 14)

[PPK92] S. Petitjean, J. Ponce, and D.J. Kriegman. Computing exact aspect graphs of curved objects: Algebraic surfaces. *IJCV*, 1992. (cited on page 69)

[PPR99] Helmut Pottmann, Martin Peternell, and Bahram Ravani. An introduction to line geometry with application. *Computer-Aided Design*, 31:3–16, 1999. (cited on page 86)

[PRJ97] Pierre Poulin, Karim Ratib, and Marco Jacques. Sketching shadows and highlights to position lights. In *Proceedings of Computer Graphics International 97*, pages 56–63. IEEE Computer Society, June 1997. (cited on page 53)

[PS92] Pellegrini and Shor. Finding stabbing lines in 3-space. *GEOMETRY: Discrete & Computational Geometry*, 8, 1992. (cited on page 87)

[PV96a] Pocchiola and Vegter. Topologically sweeping visibility complexes via pseudotriangulations. *GEOMETRY: Discrete & Computational Geometry*, 16, 1996. (cited on page 80)

[PV96b] M. Pocchiola and G. Vegter. The visibility complex. *Internat. J. Comput. Geom. Appl.*, 1996. special issue devoted to ACM-SoCG'93. (cited on page 80)

[PY90] Paterson and Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *GEOMETRY: Discrete & Computational Geometry*, 5, 1990. (cited on page 41)

[Qua96] Matthew Quail. Space time ray-tracing using ray classification. Master's thesis, Macquarie University, November 1996. (cited on page 95)

[RA96] Michael Reed and Peter K. Allen. Automated model acquisition using volumes of occlusion. In *IEEE International Conference on Robotics and Automation*, April 1996. http://www.cs.columbia.edu/robotics/. (cited on page 48)

[Rie87] J.H. Rieger. On the classification of views of piecewise smooth objects. *Image and Vision Computing*, 1987. (cited on page 29)

[Rie90] J. H. Rieger. The geometry of view space of opaque objects bounded by smooth surfaces. *Artificial Intelligence(1-2)*, 44:1–40, 1990. (cited on page 29)

[Rie92] J.H. Rieger. Global bifurcation sets and stable projections of non-singular algebraic surface. *IJCV*, 1992. (cited on page 69)

[Rie93] J. H. Rieger. Computing view graphs of algebraic surfaces. *Journal of Symbolic Computation*, 16(3):259–272, September 1993. (cited on page 69)

[Riv95] S. Rivière. Topologically sweeping the visibility complex of polygonal scenes. In *Comm. 11th Annu. ACM Sympos. Computat. Geom.*, 1995. (cited on page 80)

[Riv97] S. Rivière. *Calculs de visibilit dans un environnement polygonal 2D.* PhD thesis, Universit Joseph Fourier (Grenoble), 1997. PhD Thesis. (cited on page 80)

[RM97] D.R. Roberts and A.D. Marshall. A review of viewpoint planning. Technical Report 97007, Univeristy of Wales, Cardiff, August 1997. http://www.cs.cf.ac.uk/Research/Rrs/1997/detail007.html. (cited on page 16)

[Rob63] L.G. Roberts. Machine perception of three dimensional solids. Technical Report TR-315, Lincoln Laboratory, MIT, Cambridge, MA, May 1963. Also in Tippet, J.T *et al.*, eds., *Optical and Electro Optical Information Processing*, MIT Press, Cambridge, MA, 1964. (cited on page 32)

[Rog97] David F. Rogers. *Procedural Elements for Computer Graphics*. Mc Graw-Hill, 2 edition, 1997. (cited on pages 7, 31, 36, 40, 61)

[RPG99] Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. A perceptually based physical error metric for realistic image synthesis. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999. (cited on page 92)

[RS90] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2), 1990. (cited on page 43)

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21(4), pages 283–291, July 1987. (cited on page 59)

[RW80] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14(3), pages 110–116, July 1980. (cited on page 40)

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435–442. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0. (cited on page 92)

[San76] L. A. Santaló. *Integral Geometry and Geometric Probability*. Addison-Wesley, Reading, MA, 1976. (cited on pages 83, 88)

[SB87] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21(4), pages 119–128, July 1987. (cited on page 40)

[SB88] J. Stewman and K. Bowyer. Creating the perspective projection aspect graph of polyhedral objects. In *Second International Conference on Computer Vision (Tampa,, FL, December 5–8, 1988)*, pages 494–500, Washington, DC,, 1988. Computer Society Press. (cited on page 68)

[SB90] John H. Stewman and Kevin W. Bowyer. Direct construction of the perspective projection aspect graph of convex polyhedra. *Computer Vision, Graphics, and Image Processing*, 51(1):20–37, July 1990. (cited on page 66)

[Sbe93] M. Sbert. An integral geometry based method for fast form-factor computation. *Computer Graphics Forum*, 12(3):C409–C420, 1993. (cited on page 88)

[SBGS69] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL–TR–69–14, U.S. Air Force Human Resources Laboratory, 1969. (cited on page 34)

[SD92] W. Brent Seales and Charles R. Dyer. Viewpoint from occluding contour. *Computer Vision, Graphics and Image Processing: Image Understanding*, 55:198–211, 1992. (cited on page 69)

[SD95] François Sillion and George Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 145–152. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995, http://www-imagis.imag.fr/˜Francois.Sillion/. (cited on pages 40, 92)

[SDB85] L. R. Speer, T. D. Derose, and B. A. Barsky. A theoretical and empirical analysis of coherent ray-tracing. In M. Wein and E. M. Kidd, editors, *Graphics Interface '85 Proceedings*, pages 1–8. Canadian Inf. Process. Soc., 1985. (cited on page 47)

[SDB97] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In D. Fellner and L. Szirmay-Kalos, editors, *Computer Graphics Forum (Proc. of Eurographics '97)*, volume 16-3, pages 207–218, Budapest, Hungary, September 1997. http://www-imagis.imag.fr/˜Francois.Sillion/Papers/Index.html. (cited on page 13)

[SDS96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgann Kaufmann, San Francisco, CA, 1996. (cited on page 92)

[SG82] S. Sechrest and D. P. Greenberg. A visible polygon reconstruction algorithm. *ACM Transactions on Graphics*, 1(1):25–42, January 1982. (cited on page 36)

[SG94] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 231–238. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0, http://www.dgp.toronto.edu/people/JamesStewart/papers/. (cited on page 73)

[SG96] Oded Sudarsky and Craig Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):C249–C258, September 1996. http://www.cs.technion.ac.il/˜sudar/cv_eng.html. (cited on page 95)

[SH93] Peter Schröder and Pat Hanrahan. On the form factor between two polygons. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 163–164, 1993. (cited on page 11)

[Sha97] Erin Shaw. Hierarchical radiosity for dynamic environments. *Computer Graphics Forum*, 16(2):107–118, 1997. ISSN 0167-7055. (cited on page 95)

[She92] Thomas Shermer. Recent results in art galleries. In *Proc. IEEE*, pages 80:1384–1399, September 1992. (cited on pages 20, 74)

[Sil95] Francois X. Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):240–254, September 1995. (cited on pages 40, 92)

[Sil99] François Sillion. Will anyone really use radiosity? In *Invited talk at Graphics Interface, Kingston, Ontario*, 1999. http://www.dgp.toronto.edu/gi99/papers/programme.html. (cited on page 12)

[SJ89] T. Sripradisvarakul and R. Jain. Generating aspect graphs for curved objects. In *Proceedings, Workshop on Interpretation of 3D Scenes (Austin, TX, November 27–29, 1989)*, pages 109–115, Washington, DC., 1989. Computer Society Press, Computer Society Press. (cited on page 69)

[SK97] Sudhanshu K. Semwal and Hakan Kvarnstrom. Directed safe zones and the dual extend algorithms for efficient grid tracing during ray tracing. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, editors, *Graphics Interface '97*, pages 76–87. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1997. ISBN 0-9695338-6-1 ISSN 0713-5424. (cited on page 40)

[SK98] A. James Stewart and Tasso Karkanis. Computing the approximate visibility map, with applications to form factors and discontinuity meshing. *Eurographics Workshop on Rendering*, June 1998. http://www.dgp.toronto.edu/people/JamesStewart/. (cited on page 94)

[SKFNC97] L. Szirmay-Kalos, T. Fóris, L. Neumann, and B. Csébfalvi. An analysis of quasi-monte carlo integration applied to the transillumination radiosity method. *Computer Graphics Forum*, 16(3):271–282, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055. (cited on page 88)

[SKvW$^+$92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In Edwin E. Catmull, editor, *Proceedings of the 19th Annual ACM Conference on Computer Graphics and Interactive Techniques*, pages 249–252, New York, NY, USA, July 1992. ACM Press. (cited on page 59)

[SL98] John Snyder and Jed Lengyel. Visibility sorting and compositing without splitting for image layer decomposition. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 219–230. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8. (cited on page 34)

[Sla92] M. Slater. A comparison of three shadow volume algorithms. *The Visual Computer*, 9(1):25–38, 1992. (cited on page 57)

[SLH89] Partha Srinivasan, Ping Liang, and Susan Hackwood. Computational Geometric Methods in Volumetric Intersection for 3D Reconstruction. In *Proc. 1989 IEEE Int. Conf. Robotics and Automation*, pages 190–195, 1989. (cited on page 14)

[SLSD96] J. Shade, D. Lischinski, D. H. Salesin, and T. DeRose. Hierarchical image caching for accelerated walkthroughs of complex environments. *Computer Graphics*, 30(Annual Conference Series):75–82, 1996. (cited on page 13)

[Smi99] Brian Smits. Efficiency issues for ray tracing. *Journal of Graphics Tools*, 1999. to appear, http://www2.cs.utah.edu/˜bes/. (cited on page 40)

[Sny92] John M. Snyder. Interval analysis for computer graphics. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26(2), pages 121–130, July 1992. (cited on page 32)

[Sol78] Herbert Solomon. *Geometruc Probability*. SIAM CBMS-NSF Regional Conference Series in Applied Mathematics, Philadelphia, PA, 1978. (cited on page 88)

[Sol98] Cyril Soler. *Représentation hiérarchique de la visibilité pour le contrôle de l'erreur en simulation de l'éclairage*. PhD thesis, Université Joseph Fourier, Grenoble I, December 1998. http://www-imagis.imag.fr/˜Cyril.Soler. (cited on pages 63, 92)

[Som51] D. M. Y. Sommerville. *Analytical Geometry in Three Dimensions*. Cambridge University Press, Cambridge, 1951. (cited on page 86)

[SON96] Kristian T. Simsarian, Thomas J. Olson, and N. Nandhakumar. View-invariant regions and mobile robot self-localization. *IEEE Transactions on Robotics and Automation*, 12(5):810–816, October 1996. (cited on page 70)

[SP89] Francois Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 335–344, July 1989. (cited on page 60)

[SP94] Francois Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann, San Francisco, CA, 1994. (cited on pages 9, 11)

[SP97] I. Shimshoni and J. Ponce. Finite-resolution aspect graphs of polyhedral objects. *PAMI*, 19(4):315–327, April 1997. http://pete.cs.uiuc.edu/˜trpethe/ponce_publ.html. (cited on page 94)

[Spe92a] L. Richard Speer. An updated cross-indexed guide to the ray-tracing literature. *Computer Graphics*, 26(1):41–72, January 1992. (cited on page 10)

[Spe92b] R. Speer. A new data structure for high-speed, memory efficient ray shooting. In *Eurographics Workshop on Rendering*, 1992. (cited on page 82)

[SPP95] Mateu Sbert, Frederic Perez, and Xavier Pueyo. Global Monte Carlo: A Progressive Solution. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 231–239, New York, NY, 1995. Springer-Verlag. (cited on page 88)

[SS89] David Salesin and Jorge Stolfi. The ZZ-buffer: a simple and efficient rendering algorithm with reliable antialiasing. In *Proceedings of the PIXIM '89*, pages 451–466, 1989. (cited on page 59)

[SS90] David Salesin and Jorge Stolfi. Rendering CSG models with a ZZ-buffer. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 67–76, August 1990. (cited on page 59)

[SS96a] G. Schaufler and W. Stürzlinger. A three-dimensional image cache for virtual reality. In *Proceedings of EUROGRAPHICS'96*, 1996. (cited on page 13)

[SS96b] Cyril Soler and François Sillion. Accurate error bounds for multi-resolution visibility. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 133–142, New York City, NY, June 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4, http://www-imagis.imag.fr/˜Cyril.Soler/csoler.gb.html. (cited on page 92)

[SS98a] Cyril Soler and François Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '98 (Orlando, FL), page ?? ACM SIGGRAPH, New York, July 1998. http://www-imagis.imag.fr/˜Cyril.Soler/csoler.gb.html. (cited on pages 63, 92)

[SS98b] Cyril Soler and François Sillion. Automatic calculation of soft shadow textures for fast, high-quality radiosity. In Nelson Max and George Drettakis, editors, *Eurographics Rendering Workshop 1998*, New York City, NY, June 1998. Eurographics, Springer Wein. http://www-imagis.imag.fr/˜Cyril.Soler/csoler.gb.html. (cited on page 63)

[SSS74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*, 6(1):1–55, March 1974. (cited on pages 7, 23, 31, 36)

[Ste82] Ian Stewart. *Oh Catastrophe !* Belin, 1982. (cited on page 28)

[Ste91] J.H. Stewman. *Viewer-centered Representation for Polyhedral Objects*. PhD thesis, Department of Computer Science and Engineering, University of South Florida, Tampa, 1991. PhD Dissertation. (cited on page 68)

[STN87] Mikio Shinya, Tokiichiro Takahashi, and Seiichiro Naito. Principles and applications of pencil tracing. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21(4), pages 45–54, July 1987. (cited on page 47)

[Sto91] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, 1991. (cited on page 86)

[Stu94] W. Sturzlinger. Adaptive Mesh Refinement with Discontinuities for the Radiosity Method. In *Fifth Eurographics Workshop on Rendering*, pages 239–248, Darmstadt, Germany, June 1994. http://prometheus.gup.uni-linz.ac.at:8001/papers/. (cited on page 50)

[Stu99] Wolfgang Stuerzlinger. Imaging all visible surfaces. In *Proceedings of Graphics Interface, Kingston, Ontario*, 1999. http://www.dgp.toronto.edu/gi99/papers/programme.html. (cited on page 75)

[Sun92] Kelvin Sung. Area sampling buffer: Tracing rays with Z-buffer hardware. In A. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, volume 11(3), pages 299–310, September 1992. (cited on page 59)

[SZ89] Thomas W. Sederberg and Alan K. Zundel. Scan line display of algebraic surfaces. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 147–156, July 1989. (cited on page 36)

[TA96] Raj Talluri and J.K. Aggarwal. Mobile robot self-location using model-image feature correspondence. *IEEE Transactions on Robotics and Automation*, 12(1):63–77, February 1996. (cited on page 70)

[TA98] Seth Teller and John Alex. Frustum casting for progressive interactive rendering. Technical Report TR-740, MIT Laboratory for Computer Science, January 1998. (cited on page 47)

[TAA$^+$96] Roberto Tamassia, Pankaj K. Agarwal, Nancy Amato, Danny Z. Chen, David Dobkin, Scot Drysdale, Steven Fortune, Michael T. Goodrich, John Hershberger, Joseph O'Rourke, Franco P. Preparata, Joerg-Rudiger Sack, Subhash Suri, Ioannis Tollis, Jeffrey S. Vitter, and Sue Whitesides. Strategic directions in computational geometry. *ACM Computing Surveys*, 28(4):591–606, December 1996. (cited on page 19)

[TAT95] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on robotics and automation*, 11(1):86–104, February 1995. (cited on page 16)

[Tel92a] Seth J. Teller. Computing the antipenumbra of an area light source. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26-2, pages 139–148, July 1992. http://graphics.lcs.mit.edu/˜seth/pubs/pubs.html. (cited on page 87)

[Tel92b] Seth J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, CS Division, UC Berkeley, October 1992. Tech. Report UCB/CSD-92-708, http://graphics.lcs.mit.edu/˜seth/pubs/pubs.html. (cited on pages 54, 55, 84, 86)

[TFFH94] Seth Teller, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 443–450. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0, http://graphics.lcs.mit.edu/˜seth/pubs/pubs.html. (cited on page 55)

[TG95] G. H. Tarbox and S. N. Gottschlich. Planning for complete sensor coverage in inspection. *Computer Vision and Image Understanding: CVIU*, 61(1):84–111, January 1995. (cited on page 74)

[TG97a] Nicholas Tsingos and Jean-Dominique Gascuel. Sound rendering in dynamic environments with occlusions. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, editors, *Graphics Interface '97*, pages 9–16. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1997. ISBN 0-9695338-6-1 ISSN 0713-5424, http://www.bell-labs.com/user/tsingos/. (cited on page 60)

[TG97b] Nicolas Tsingos and Jean-Dominique Gascuel. Fast rendering of sound occlusion and diffraction effects for virtual acoustic environments. In *104th AES convention, Amsterdam, The Netherlands, preprint n. 4699 (P4-7)*, May 1997. http://www.bell-labs.com/user/tsingos/. (cited on page 60)

[TH93] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 239–246, 1993. http://graphics.lcs.mit.edu/˜seth/pubs/pubs.html. (cited on pages 54, 55, 87)

[Tho56] R. Thom. Les singularités des applications différentiables. *Annales Institut Fourier*, 6:43–87, 1956. (cited on page 28)

[Tho72] R. Thom. *Structural Stability and Morphogenesis*. Benjamin, New-York, 1972. (cited on page 28)

[Tor90] Enric Torres. Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 507–518. North-Holland, September 1990. (cited on page 95)

[TS91] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991. http://graphics.lcs.mit.edu/˜seth/pubs/pubs.html. (cited on pages 54, 55)

[TT90] Toshimitsu Tanaka and Tokiichiro Takahashi. Cross scanline algorithm. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 63–74. North-Holland, September 1990. (cited on page 36)

[TT95] Toshimitsu Tanaka and Tokiichiro Takahashi. Fast shadowing algorithm for linear light sources. *Computer Graphics Forum*, 14(3):205–216, August 1995. Proceedings of Eurographics '95. ISSN 1067-7055. (cited on page 82)

[TT97] T. Tanaka and T. Takahashi. Fast analytic shading and shadowing for area light sources. *Computer Graphics Forum*, 16(3):231–240, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055. (cited on pages 36, 82)

[TTK96] Konstantinos Tarabanis, Roger Y. Tsai, and Anil Kaul. Computing occlusion-free viewpoints. *PAMI*, 18(3):279–292, March 1996. (cited on page 51)

[TUWR97] Emmanuelle Trucco, Manickam Umasuthan, Andrew M. Wallace, and Vito Roberto. Model-based planning of optimal sensor placement for inspection. *IEEE Transactions on Robotics and Automation*, 13(2):182–194, April 1997. (cited on page 75)

[Tv97] A. C. Telea and C. W. A. M. van Overveld. The close objects buffer: a sharp shadow detection technique for radiosity methods. *Journal of Graphics Tools*, 2(2):1–8, 1997. ISSN 1086-7651. (cited on page 92)

[TWFP97] Robert F. Tobler, Alexander Wilkie, Martin Feda, and Werner Purgathofer. A hierarchical subdivision algorithm for stochastic radiosity methods. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 193–204, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4. (cited on page 88)

[Ull89] Shimon Ullman. Aligning pictorial descriptions: An approach to object recognition. *Cognition*, 32(3):193–254, August 1989. (cited on page 13)

[Urr98] Jorge Urrutia. Art gallery and illumination problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook on Computational Geometry*. Elsevier Science, 1998. http://www.csi.uottawa.ca:80/˜jorge/online_papers. (cited on pages 20, 74)

[Vea97] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. Ph.d. thesis, Stanford University, December 1997. http://www-graphics.stanford.edu/˜ericv/. (cited on page 10)

[Ved93] Christophe Vedel. Computing Illumination from Area Light Sources by Approximate Contour Integration. In *Proceedings of Graphics Interface '93*, pages 237–244, San Francisco, CA, May 1993. Morgan Kaufmann. (cited on page 32)

[VLN96] M. Venditelli, J.P. Laumond, and C. Nissoux. Obstacle distances and visibility in the car-like robot metrics. Technical Report 96437, LAAS, November 1996. doc@laas.fr. (cited on pages 43, 44)

[WA77] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics*, 11(2):214–222, July 1977. Proceedings of SIGGRAPH'77, held in San Jose, California; 20–22 July 1977. (cited on pages 32, 46)

[WA90] Andrew Woo and John Amanatides. Voxel occlusion testing: a shadow accelerator for ray tracing. In *Proceedings of Graphics Interface '90*, pages 213–220, June 1990. (cited on page 48)

[Wal75] David Waltz. Understanding lines drawings of scenes with shadows. In Patrick H. Winston, editor, *The Psychology of Computer Vision*, Computer Science Series, pages 19–91. McGraw-Hill, 1975. (cited on page 6)

[Wan92] Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25(2), pages 39–42, March 1992. (cited on page 8)

[War69] J. Warnock. A hidden-surface algorithm for computer generated half-tone pictures. Technical Report TR 4–15, NTIS AD-733 671, University of Utah, Computer Science Department, 1969. (cited on page 33)

[Wat70] G.S. Watkins. *A Real Time Visible Surface Algorithm*. Ph.d. thesis, University of Utah, June 1970. (cited on page 36)

[Wat88] N. A. Watts. Calculating the principal views of a polyhedron. In *Ninth International Conference on Pattern Recognition (Rome, Italy, November 14–17, 1988)*, pages 316–322, Washington, DC, 1988. Computer Society Press. (cited on page 66)

[Wat90] Mark Watt. Light-water interaction using backward beam tracing. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 377–385, August 1990. (cited on page 48)

[WBP98] Y. Wang, H Bao, and Q. Peng. Accelerated walkthroughs of virtual environments based on visibility processing and simplification. In *Computer Graphics Forum (Proc. of Eurographics '98)*, volume 17-3, pages C–187–C195, Lisbon, Portugal, September 1998. (cited on page 83)

[WC91] Andrew Woo and Steve Chall. An efficient scanline visibility implementation. In *Proceedings of Graphics Interface '91*, pages 135–142, June 1991. (cited on page 36)

[WEH89] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 315–324, July 1989. (cited on page 88)

[Wei93] I. Weiss. Geometric invariant and object recognition. *Internat. J. Comput. Vision*, 10(3):207–231, 1993. (cited on page 14)

[WF90] R. Wang and H. Freeman. Object recognition based on characteristic view classes. In *Proceedings* 10*th International Conference on Pattern Recognition*, Atlantic City, NJ, 17-21 June 1990. (cited on page 66)

[WH96] L. R. Williams and A. R. Hanson. Perceptual completion of occluded surfaces. *Computer Vision and Image Understanding: CVIU*, 64(1), 1996. (cited on page 29)

[WHG84] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984. (cited on pages 40, 59)

[Whi55] H. Whitney. On singularities of mappings of euclidean spaces. i. mappings of the plane into the plane. *Annals of Mathematica*, 62(3):374–410, 1955. (cited on page 28)

[Whi78] T. Whitted. A scan line algorithm for computer display of curved surfaces. In *Computer Graphics (Special SIGGRAPH '78 Issue, preliminary papers)*, pages 8–13, August 1978. (cited on page 36)

[Whi80] Turner Whitted. An improved illumination model for shaded display. *CACM, 1980*, 23(6):343–349, 1980. (cited on pages 10, 36, 40, 46)

[WHP98] Adam Worrall, David Hedley, and Derek Paddon. Interactive animation of soft shadows. In *Proceedings of Computer Animation 1998*, pages 88–94. IEEE Computer Society, June 1998. http://www.cs.bris.ac.uk/˜worrall/scope/port95.html. (cited on page 96)

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12(3), pages 270–274, August 1978. (cited on page 59)

[Wil96] L. R. Williams. Topological reconstruction of a smooth manifold-solid from its occluding contour. *Computer Vision and Image Understanding: CVIU*, 64(2), 1996. (cited on pages 29, 30)

[Woo92] Andrew Woo. The shadow depth map revisited. In David Kirk, editor, *Graphics Gems III*, pages 338–342, 582. Academic Press, Boston, MA, 1992. (cited on page 59)

[Woo97] Andrew Woo. Recursive grids and ray bounding box comments and timings. *Ray Tracing News*, 10(3), December 1997. http://www.povray.org/rtn/. (cited on page 40)

[Wor97] Steve Worley. Fast soft shadows. *Ray Tracing News*, 10(1), January 1997. http://www.povray.org/rtn/. (cited on page 64)

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990. http://www.iro.umontreal.ca/labs/infographie/papers/. (cited on pages 8, 24)

[WREE67] C. Wylie, G.W. Romney, D.C. Evans, and A.C. Erdahl. Halftone perspective drawings by computer. In *FJCC*, pages 49–58, 1967. (cited on page 36)

[WS94] G. Winkenbach and D. H. Salesin. Computer-generated pen-and-ink illustration. *Computer Graphics*, 28(Annual Conference Series):91–100, July 1994. http://www.cs.washington.edu/research/graphics/pub/. (cited on page 8)

[WS99] Peter Wonka and Dieter Schmalstieg. Occluder shadows for fast walkthroughs ofurban environments. In *Eurographics'99*, August 1999. (cited on page 62)

[WW92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley, 1992. (cited on pages 7, 35, 40)

[WW97] Daphna Weinshall and Michael Werman. On view likelihood and stability. *T-PAMI*, 19-2:97–108, 1997. (cited on page 94)

[WWP95] Adam Worrall, Claire Willis, and Derek Paddon. Dynamic discontinuities for radiosity. In *Edugraphics + Compugraphics Proceedings*, pages 367–375, P.O. Box 4076, Massama, 2745 Queluz, Portugal, December 1995. GRASP- Graphic Science Promotions and Publications. http://www.cs.bris.ac.uk/˜worrall/scope/port95.html. (cited on page 96)

[Yan85] Johnson K. Yan. Advances in computer-generated imagery for flight simulation. *IEEE Computer Graphics and Applications*, 5(8):37–51, August 1985. (cited on page 34)

[YHS95] Seungku Yi, Robert M. Haralick, and Linda G. Shapiro. Optimal sensor and light source positioning for machine vision. *Computer Vision and Image Understanding: CVIU*, 61(1):122–137, January 1995. (cited on page 75)

[YKSC98] Kwan-Hee Yoo, Dae Seoung Kim, Sung Yong Shin, and Kyung-Yong Chwa. Linear-time algorithms for finding the shadow volumes from a convex area light source. *Algorithmica*, 20(3):227–241, March 1998. (cited on page 50)

[YR96] R. Yagel and W. Ray. Visibility computation for efficient walkthrough complex environments. *PRESENCE*, 5(1):1–16, 1996. http://www.cis.ohio-state.edu/volviz/Papers/1995/presence.ps.gz. (cited on page 55)

[Zat93] Harold R. Zatz. Galerkin radiosity: A higher order solution method for global illumination. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 213–220, 1993. (cited on page 59)

[ZD93] Ze Hong (Jenny) Zhao and David Dobkin. Continuous Algorithms for Visibility: the Space Searching Approach. In *Fourth Eurographics Workshop on Rendering*, pages 115–126, Paris, France, June 1993. (cited on page 54)

[Zha91] Ning Zhang. Two Methods for Speeding up Form-factor Calculation. In *Second Eurographics Workshop on Rendering*, Barcelona, Spain, May 1991. (cited on page 54)

[Zha98a] Hanson Zhang. Forward shadow mapping. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, New York City, NY, June 1998. Eurographics, Springer Wien. SBN 3-211-83213-0, http://www.cs.unc.edu/ zhangh/shadow.html. (cited on page 59)

[Zha98b] Hansong Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. PhD thesis, University of North Carolina, Chapel Hill, 1998. http://www.cs.unc.edu/ zhangh/research.html. (cited on page 61)

[ZMHH97] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7, http://www.cs.unc.edu/˜zhangh/hom.html. (cited on page 61)

Object-Space Visibility Culling

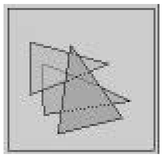
Claudio T. Silva

Information Visualization Research Department
AT&T Labs-Research
csilva@research.att.com
http://www.research.att.com/~csilva

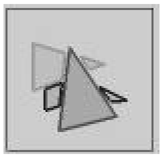
Some slides as well as material for slides have been provided by
Satyan Coorg (MIT/IBM), Sigal Dahan (Tel-Aviv),
David Luebke (UNC/Virginia), Jim Klosowski (IBM),
and Dudu Sayag (Tel-Aviv)

Visibility Problems

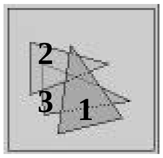
Given a collection of triangles:

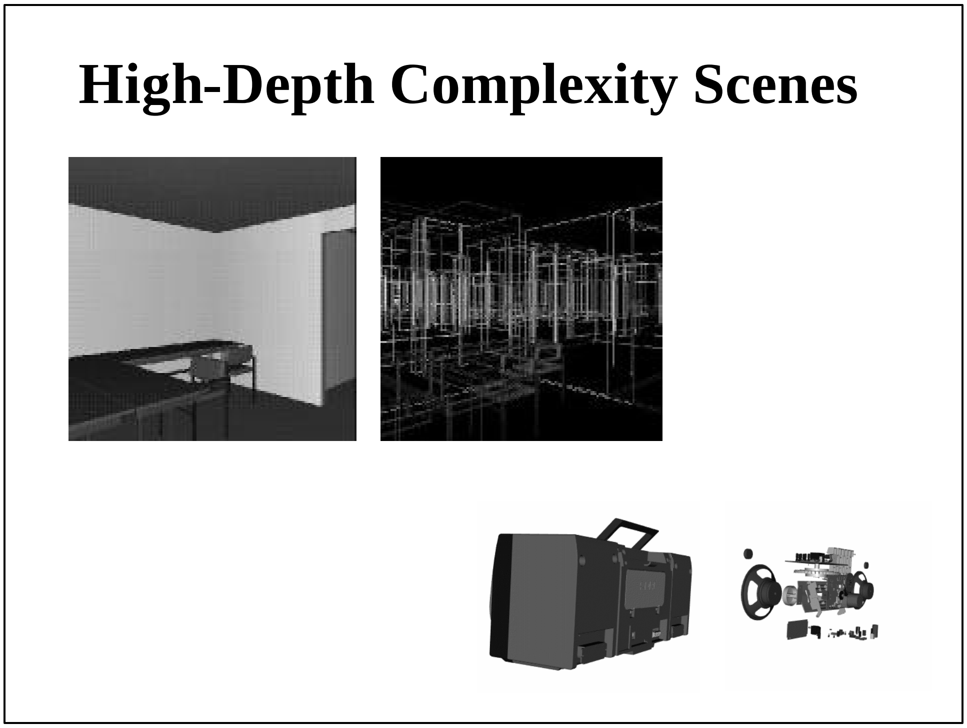


Visibility Determination
Find the visible fragments:

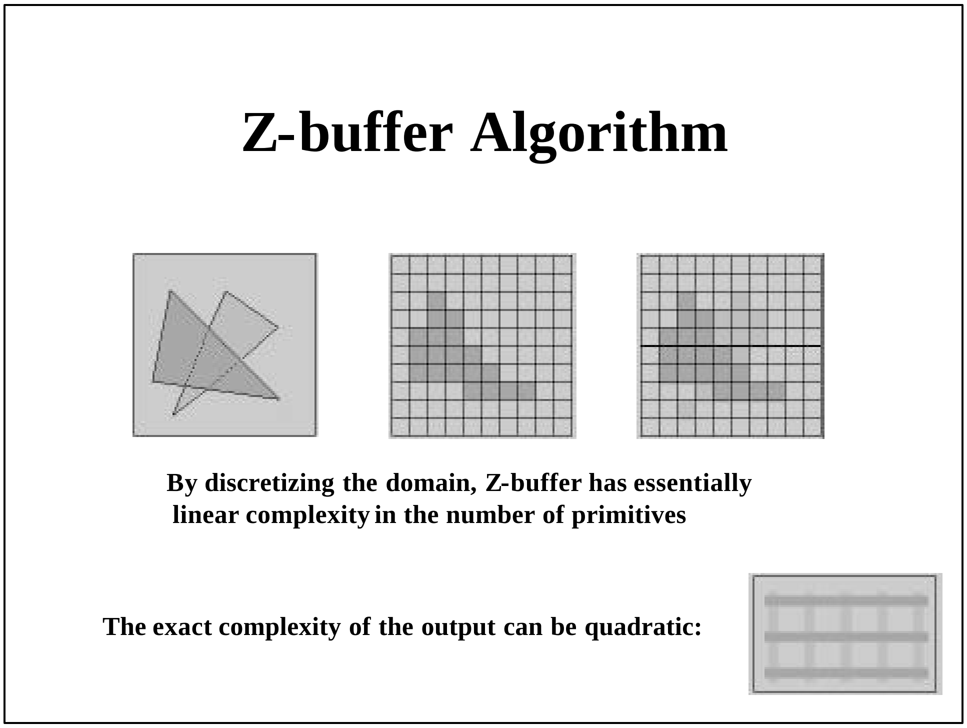


Visibility Ordering
Find a “visibility” labeling
of the fragments:





High-Depth Complexity Scenes

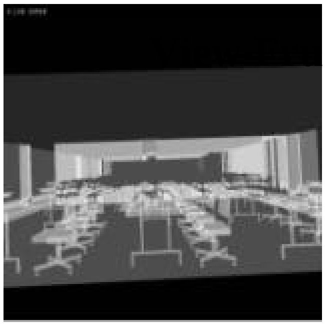


Z-buffer Algorithm

By discretizing the domain, Z-buffer has essentially linear complexity in the number of primitives

The exact complexity of the output can be quadratic:

Depth-Complexity



View Frustum

Visibility Culling

Approximate Visibility Determination

Develop algorithms that are output sensitive, that is, if out of the N triangles, only K of them are visible, the algorithm has complexity that depends more closely on K

Drop the exact visibility requirement, and instead attempt to develop algorithms that estimate the triangles which have visible fragments

Algorithms that overestimate the visible fragments, the so called conservative visibility algorithms

Graphics Hardware Performance

- Current graphics hardware "peaks" at approximate 15 million triangles per second, but actually only renders 1-3 million triangles per second
- Real-time usually means 30Hz (at least 15 frames per second)
- 1M-3M at 30Hz = 33K-100K triangles per frame
- 33-100 thousand triangles is not much!

- Hardware will improve, but so will datasets, mostly due to better 3D scanning and modeling technology

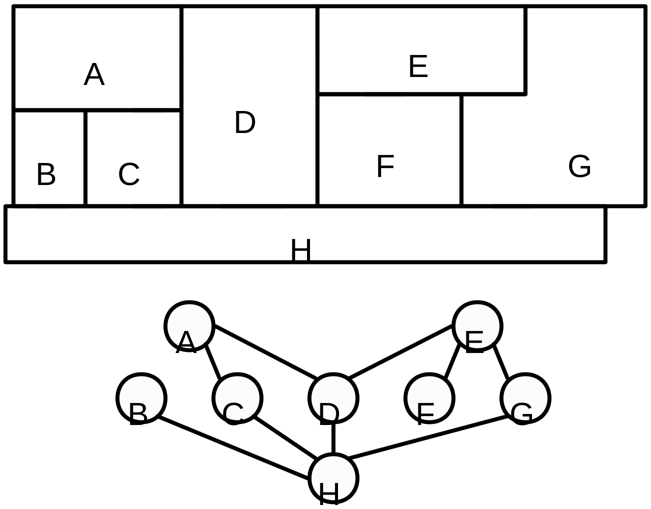
Talk Summary

- Cells and portals
  - Teller and Sequin, Siggraph 91
  - Luebke and Georges, I3D 95
- Visibility culling with large occluders
  - Coorg and Teller, SoCG 96 and I3D 97
  - Hudson et al, SoCG 97
- Prioritized-Layer Projection Algorithm
  - Klosowski and Silva Vis99 and TVCG00

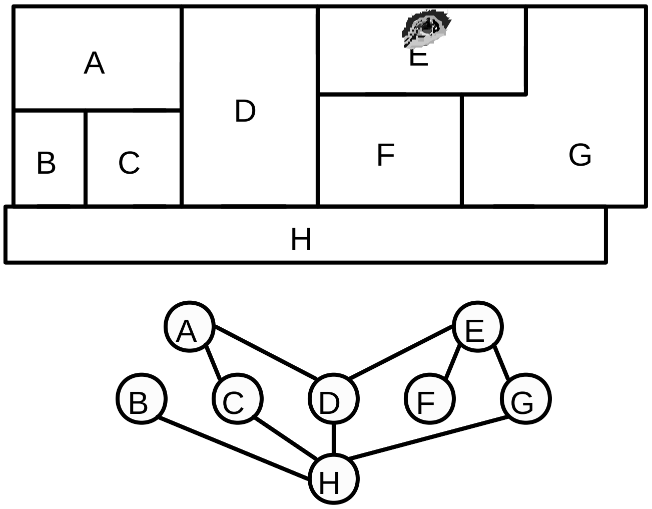
Talk Summary

- Cells and portals
  - – Teller and Sequin, Siggraph 91
  - – Luebke and Georges, I3D 95
- Visibility culling with large occluders
  - – Coorg and Teller, SoCG 96 and I3D 97
  - – Hudson et al, SoCG 97
- Prioritized-Layer Projection Algorithm
  - – Klosowski and Silva Vis99 and TVCG00

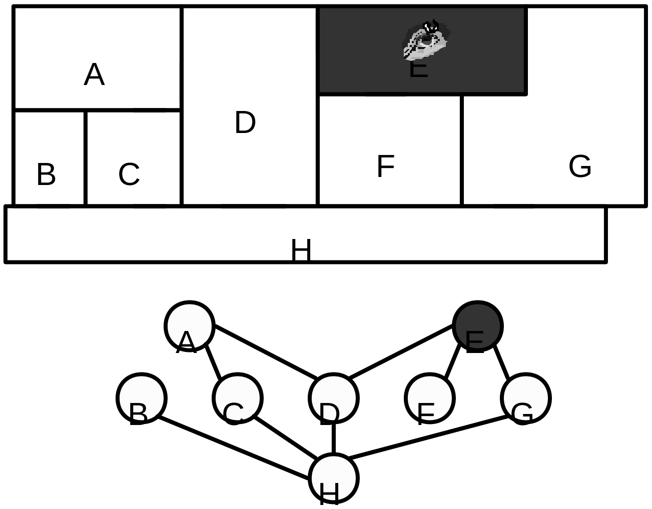
Cells & Portals



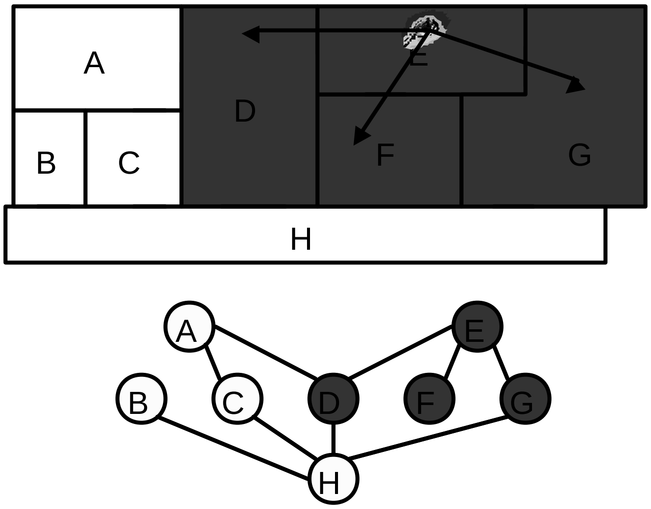
Cells & Portals



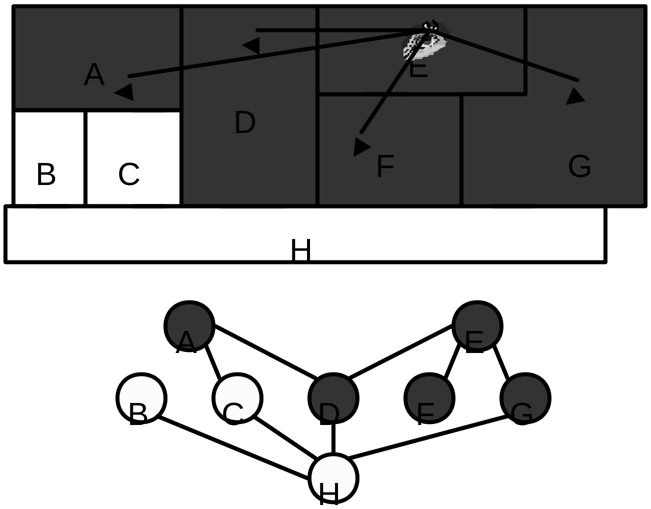
Cells & Portals



Cells & Portals



Cells & Portals



Teller and Sequin’s Approach

(1) Decompose space into convex cells

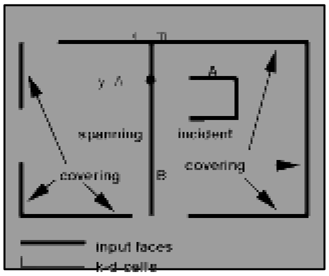
(2) For each cell, identify its boundary edges into two sets: opaque or portal

(3) Precompute visibility among cells

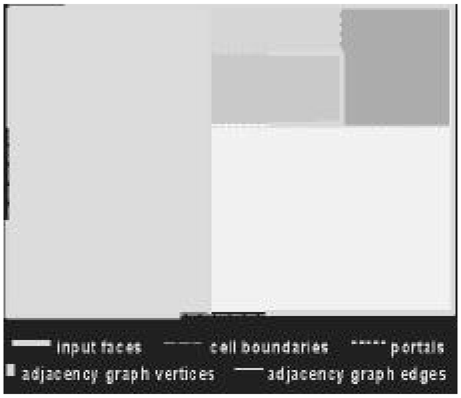
(4) During viewing (eg, walkthrough phase), use the precomputed potentially visible polygon set (PVS) of each cell to speed-up rendering

Space Subdivision

Input Scene:

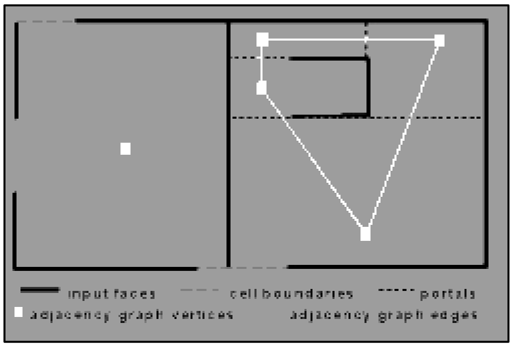


Convex subdivision:

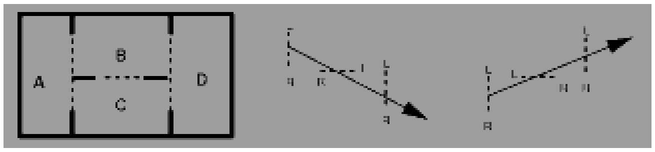


Generated by computing a k-d tree of the input faces

Determining Adjacent Information



Computing the PVS of a cell



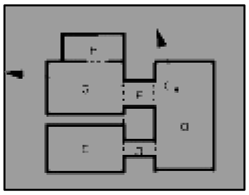
Linear programming problem:

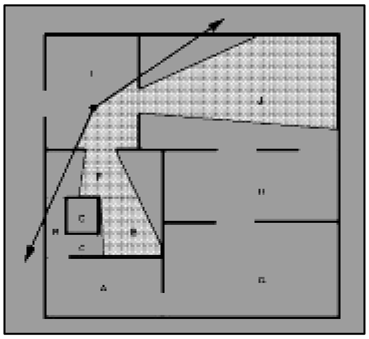
$$
\begin{aligned}
&S \bullet R \geq 0, && \forall L \in \mathrm{L}\\
&S \bullet R \leq 0, && \forall R \in \mathrm{R}
\end{aligned}
$$

```
Find_Visible_Cells(cell C, portal sequence P, visible cell set V)
  V=V ∪ C
  for each neighbor N of C
    for each portal p connecting C and N
      orient p from C to N
      P’ = P concatenate p
      if Stabbing_Line(P’) exists then
        Find_Visible_Cells (N, P’, V)
```

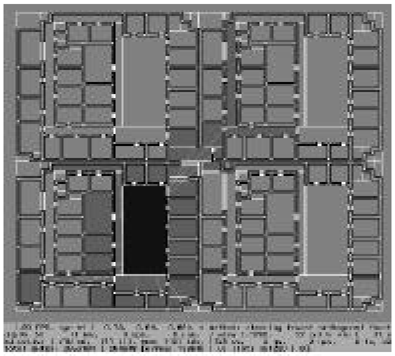
Eye-to-Cell Visibility

The eye-to-cell visibility of any observer is a subset of the cell-to-cell visibility for the cell containing the observer

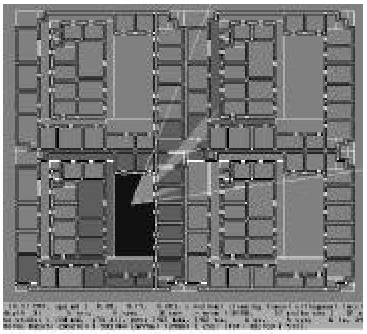




Results



(a) A source cell (dark blue), its cell-to-cell visibility (light blue), and stabbing lines (green).



(d) The same observer, with a 60° view cone. The eye-to-cell visibility is shown in blue; the exact visible area is shown in blue-green. The green cells have been dynamically culled.

Luebke and Georges, I3D 95

- Instead of pre-processing all the PVS calculation, it is possible to use image-space portals to make the computation easier
- No preprocessing
- Can be used in a dynamic setting

pfPortals

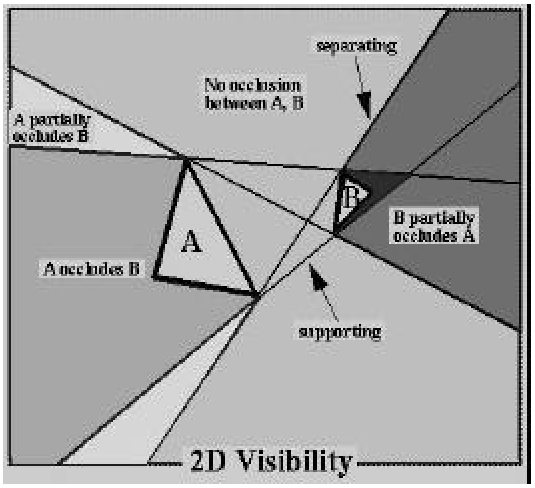
code available at http://www.cs.virginia.edu/~luebke

- Depth-first adjacency graph traversal
  - – Render cell containing viewer
  - – Treat portals as special polygons
    - If portal is visible, render adjacent cell
    - But clip to boundaries of portal!
    - Recursively check portals in that cell against new clip boundaries (and render)
  - – Each visible portal sequence amounts to a series of nested portal boundaries

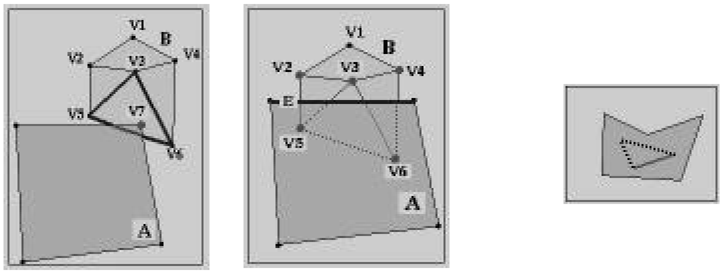
Talk Summary

- Cells and portals
  - – Teller and Sequin, Siggraph 91
  - – Luebke and Georges, I3D 95
- Visibility culling with large occluders
  - – Coorg and Teller, SoCG 96 and I3D 97
  - – Hudson et al, SoCG 97
- Prioritized-Layer Projection Algorithm
  - – Klosowski and Silva 1999 and 2000

When does A occludes B ?

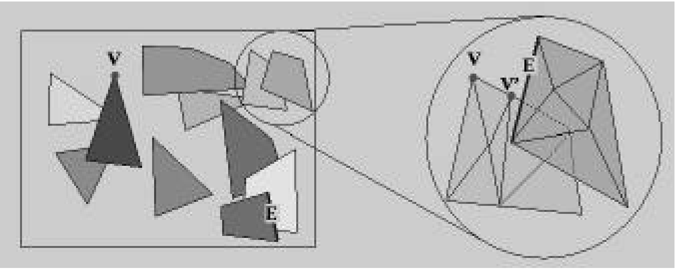


Idea: Track Visibility Changes



Possible because visibility changes little from frame to frame

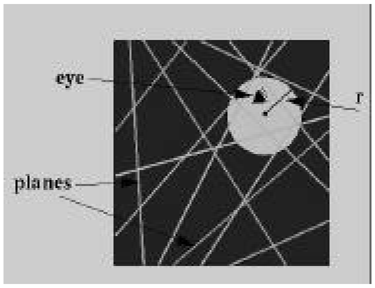
Events to care about...



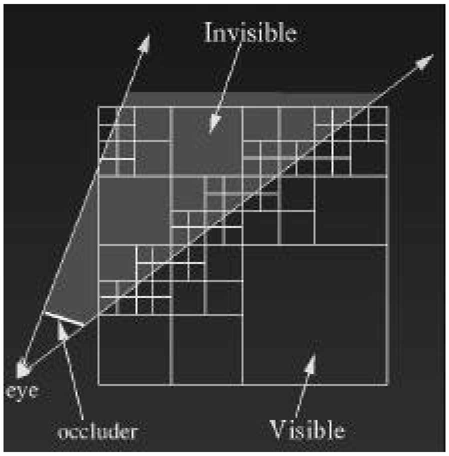
Coorg and Teller, SoCG 96

To reduce the number of events to be tracked:

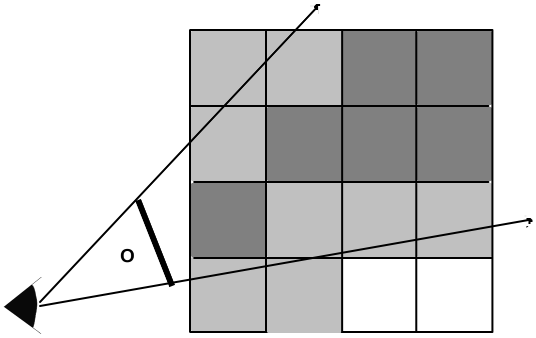
(1) use a sphere



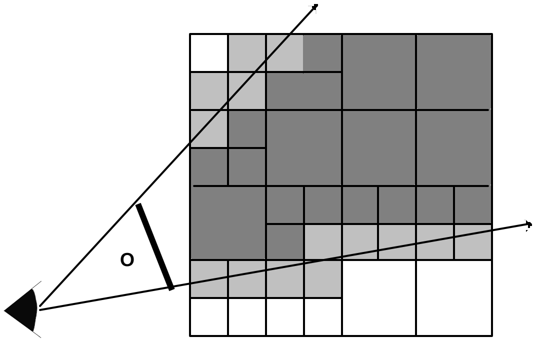
(2) and a hierarchy of objects



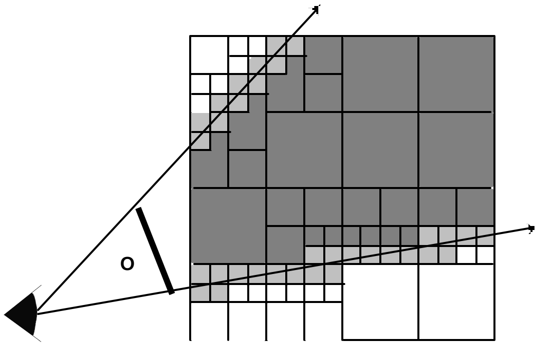
Hierarchical Tests



Hierarchical Tests

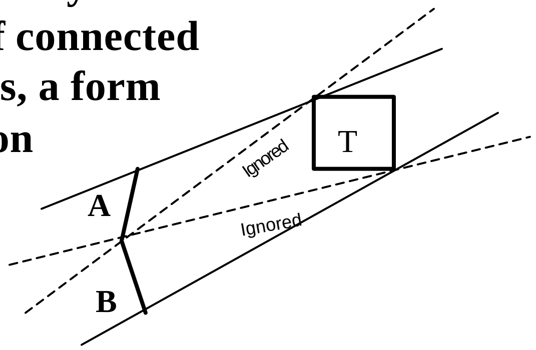


Hierarchical Tests

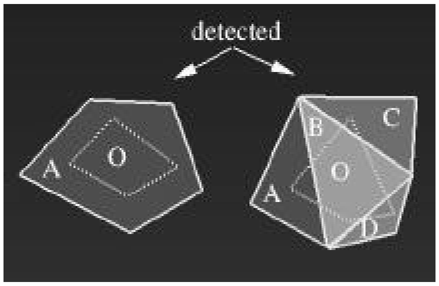


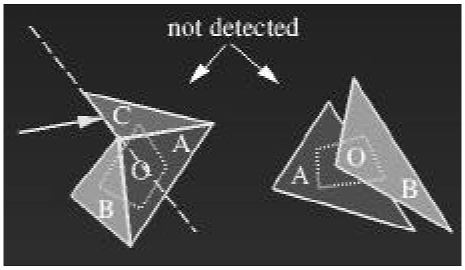
Coorg and Teller, I3D 97

Added the capability to join the effect of connected occluders, that is, a form of occluder fusion

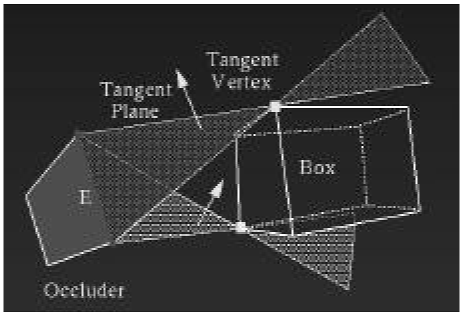


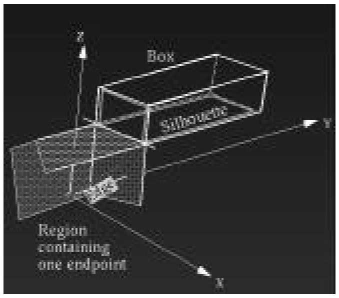
Occluder Fusion

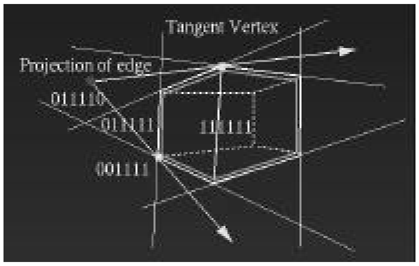




Fast Tangent Plane Computation

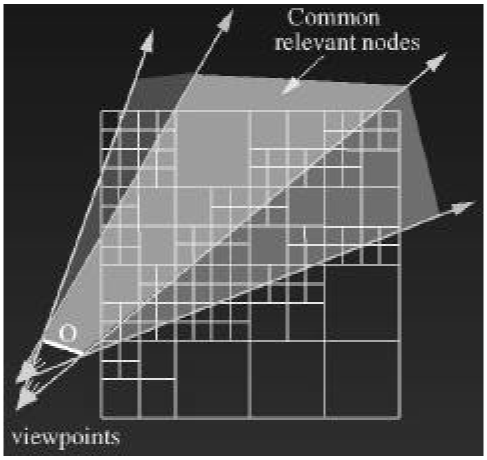




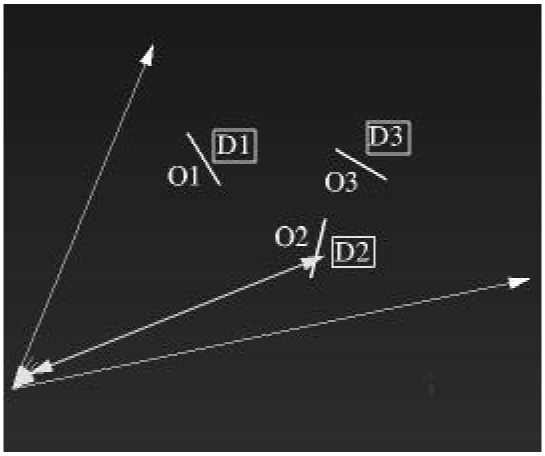


Because this computation is fast, it is no longer necessary to keep fine-grain visibility events

Use Temporal Coherence to Cache Relevant Events



Detail Occluders



Metric for Comparing Occluder Quality

Occluder quality: $(-A\ (N * V)) / ||D||^2$

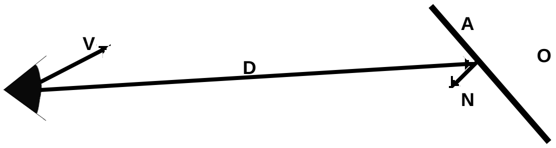
A : the occluder's area

N : normal

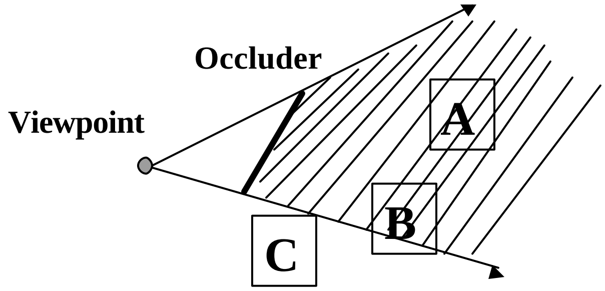
V : viewing direction

D : the distance between the viewpoint and the occluder center

Large polygon have large area-angle.



Hudson et al, SoCG 97



Occluder Quality

- **Solid Angle** **(similar to Coorg and Teller)**
- **Depth Complexity**

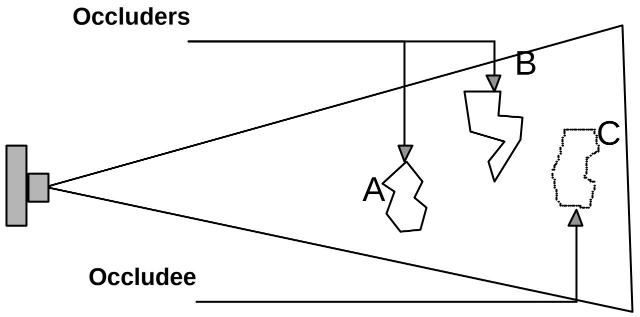
Talk Summary

- Cells and portals
  - – Teller and Sequin, Siggraph 91
  - – Luebke and Georges, I3D 95
- Visibility culling with large occluders
  - – Coorg and Teller, SoCG 96 and I3D 97
  - – Hudson et al, SoCG 97
- Prioritized-Layer Projection Algorithm
  - – Klosowski and Silva 1999 and 2000

PLP: A Framework for Time-Critical Rendering

- Rendering within a budget
- Low-complexity preprocessing
- No pre-selection of occluders
- Object-space occluder fusion
- Simple to implement

Combining Occluders

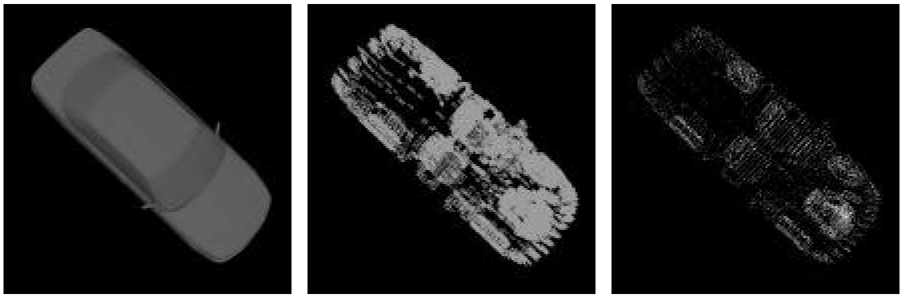


PLP Overview

- Occupancy-based spatial tessellation
- Prioritized cell traversal algorithm

Spatial Tessellation Algorithm

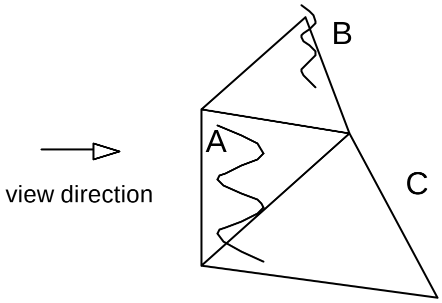
- **Insert original vertices into octree**
- **Leaves of octree define the spatial tessellation**
- **Insert geometry into mesh cells**



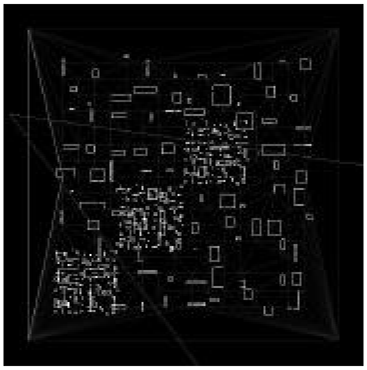
Obs: Other types of spatial tessellations (such as Delaunay triangulations) also work fine!

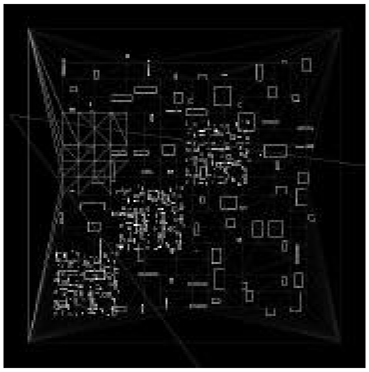
Priority-Based Traversal Algorithm

```
while (PQ is not empty)
    project cell with minimum
    opacity;
  if (budget reached) stop;
  for each adjacent cell c
      if (c not projected)
      update opacity value o;
      enqueue c;
```

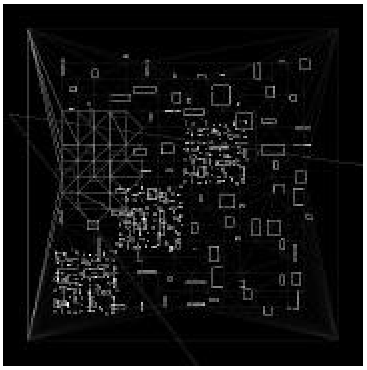


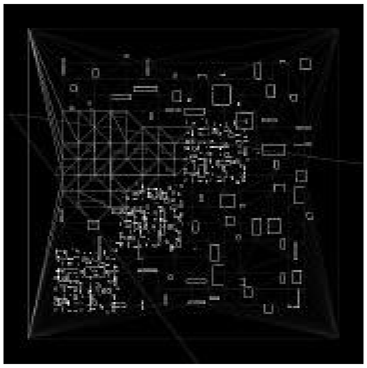
2D Prototype



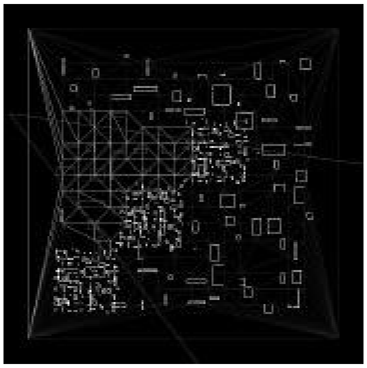


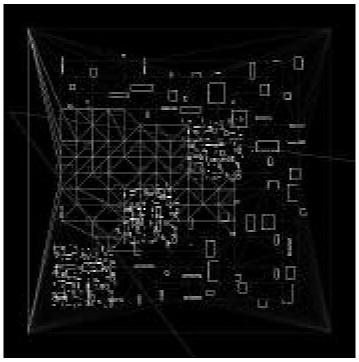
2D Prototype





2D Prototype

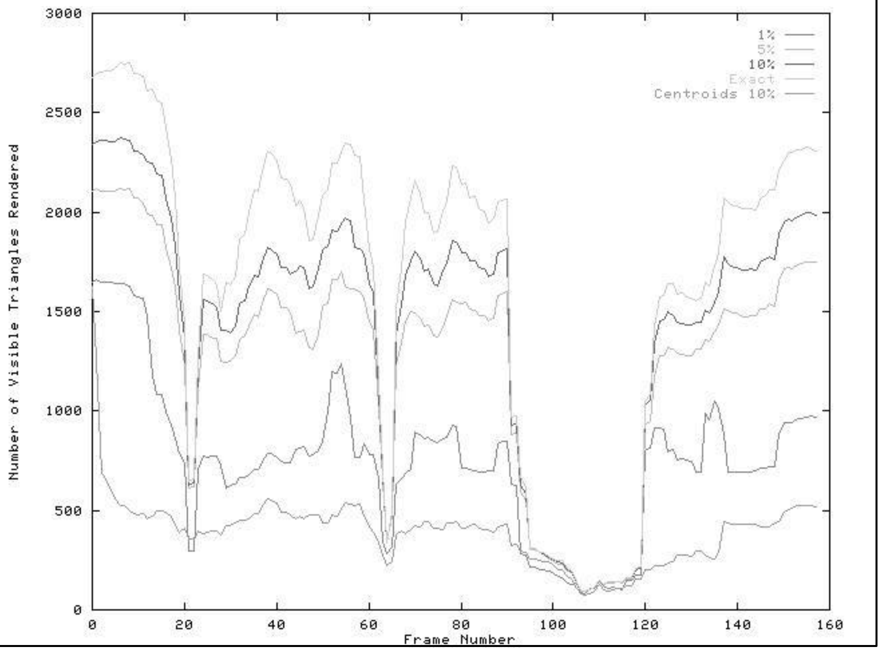




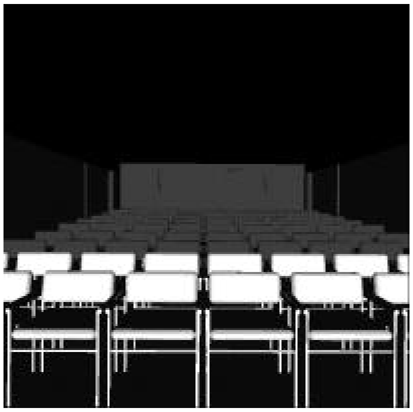
2D Prototype

DEMO!

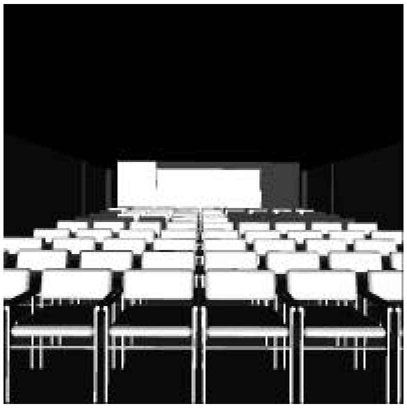
Finding Visible Triangles



Finding Visible Triangles

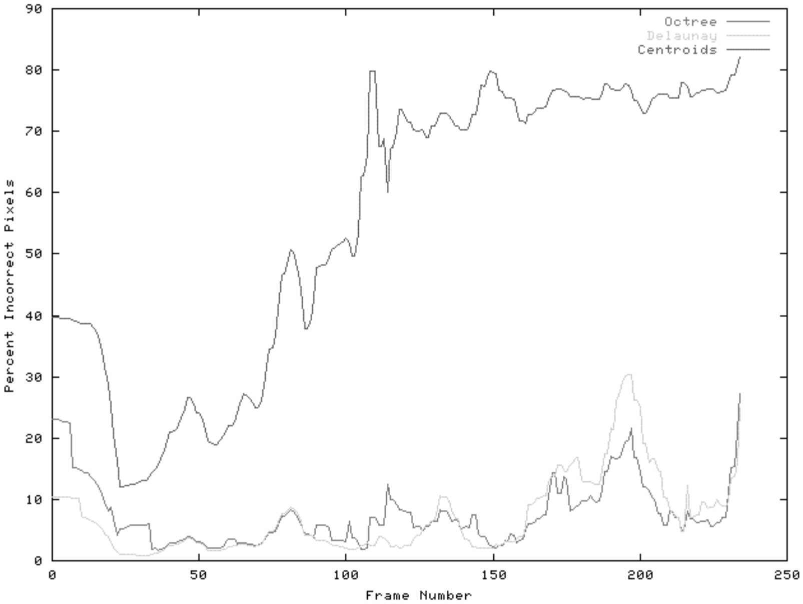


Front-to-back projection at 10%



PLP at 10%

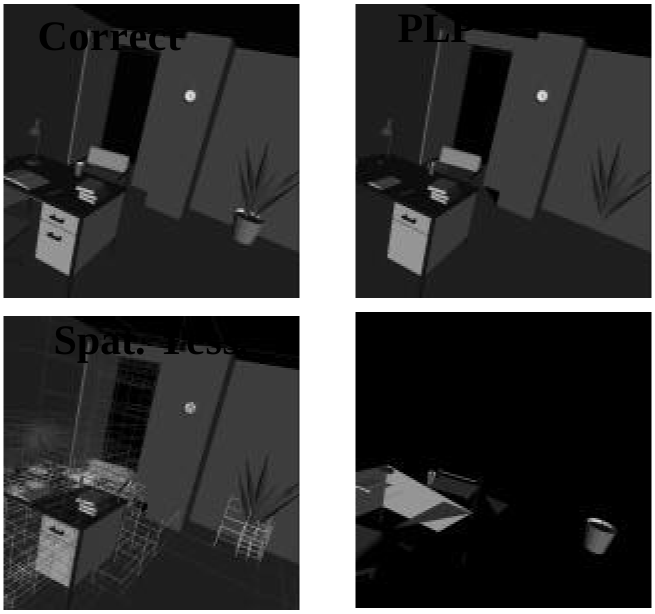
Wrong Pixels Over Several Frames



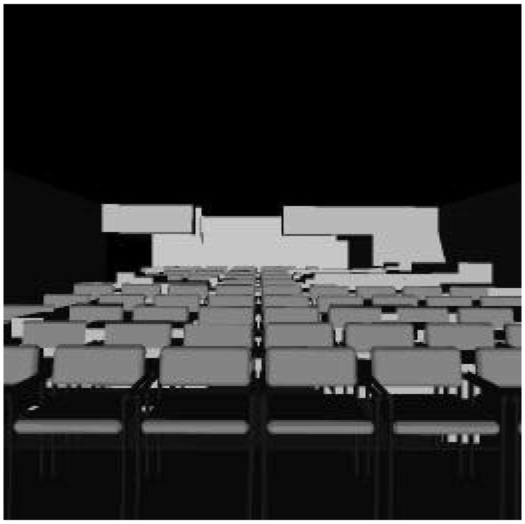
Some Quantitative Results

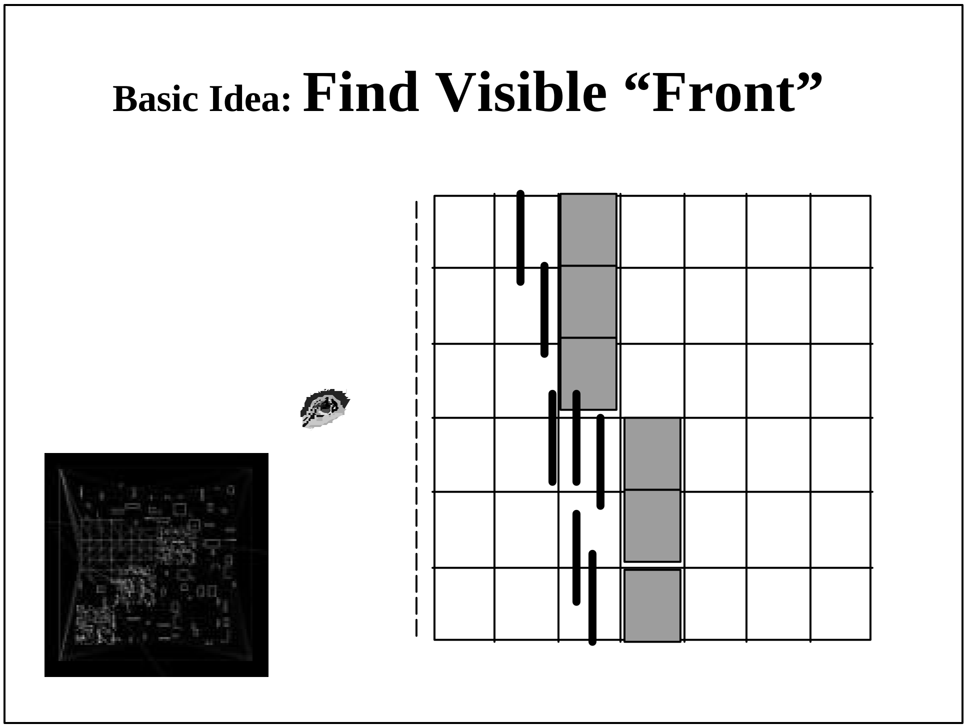
- **With 1% budget PLP finds over 50% of visible set on average**
- **For the 500K-triangle city model, it takes 2 minutes of preprocessing**
  For this model, a 5% budget, PLP finds about 80% of the visible set
  At most 4% of the pixels in a given image are wrong!

Mistakes

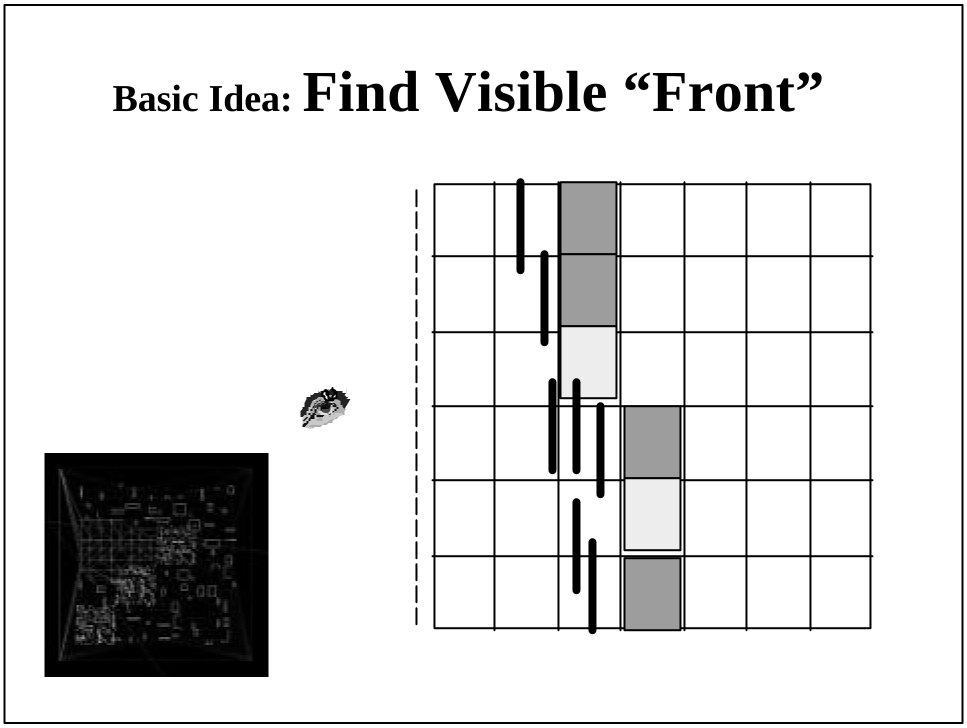


Filling Up The “Gaps”
A Conservative PLP (cPLP)



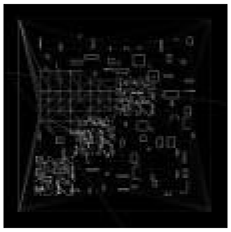


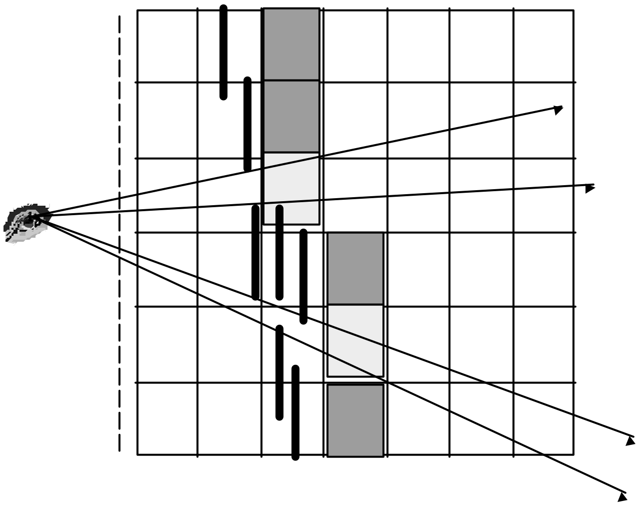
Basic Idea: Find Visible “Front”



Basic Idea: Find Visible “Front”

Basic Idea: Find Visible “Front”





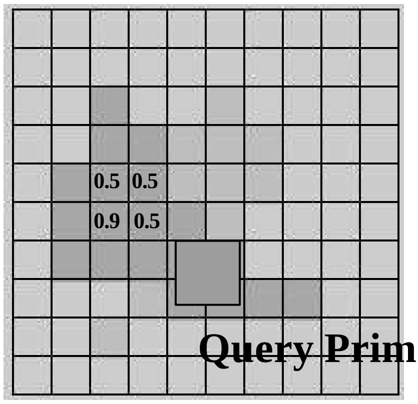
How to identify the visible Front ?

- HP Occlusion-Culling Test
- A novel Stencil-Buffer Technique

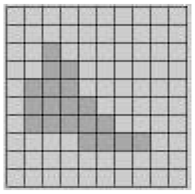
The HP Occlusion-Cullig Test

- Very simple to use

- HP fx6 hardware can perform about 1000 to 6000 tests per second



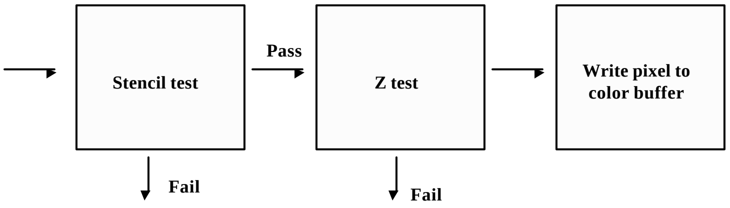
OpenGL buffers



(R, G, B) 8-bit per coordinate	Color buffer
(Z) 24-bit per coordinate	Z buffer
(S) 4 to 8 bits per coordinate	Stencil buffer

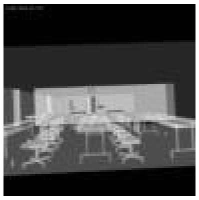
There are other buffers!

OpenGL Pixel Pipeline

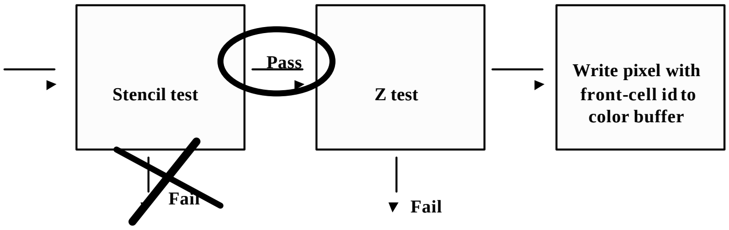


glStencilFunc ()

glStencilOp()



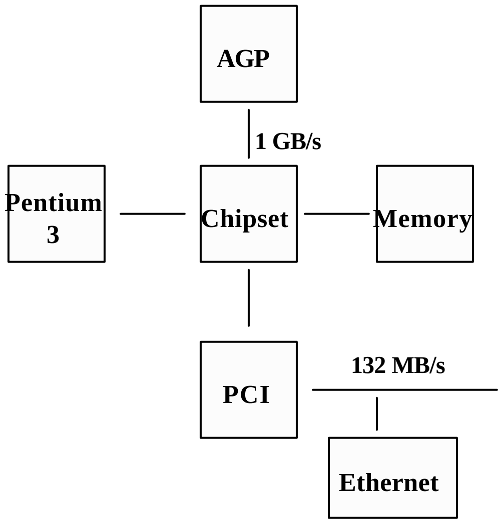
Computing Visibility with OpenGL



Basic (non-optimized) algorithm:

(1) save color buffer
(2) disable changes to the Z-buffer
(3) clear Stencil-buffer
(4) render
(5) each cell on a non-zero stencil buffer is visible
(6) restore color buffer and enable changes to Z-buffer

The PC and Graphics Hardware



AGP

1 GB/s

Pentium 3

Chipset

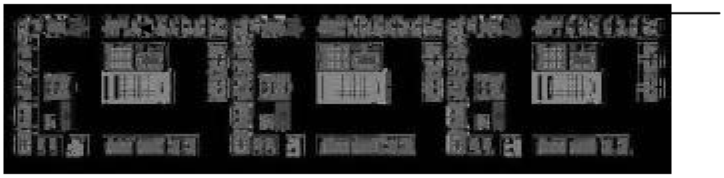
Memory

PCI

132 MB/s

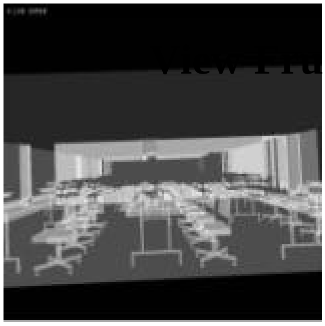
Ethernet

Test Dataset -- 3rd Floor of SODA Hall



Movies

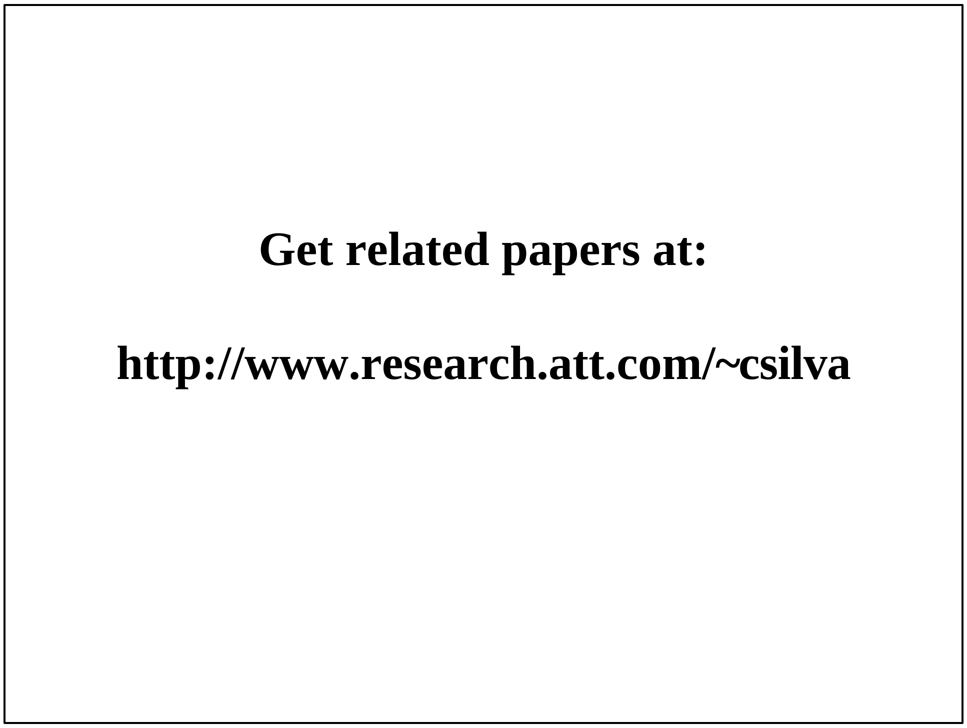
Depth-Complexity



View Frustum

Front-to-back HP

cPLP



Get related papers at:

http://www.research.att.com/~csilva

The Prioritized-Layered Projection Algorithm for Visible Set Estimation

James T. Klosowski and Cláudio T. Silva, *Member*, *IEEE*

Abstract—*Prioritized-Layered Projection* (PLP) is a technique for fast rendering of high depth complexity scenes. It works by *estimating* the visible polygons of a scene from a given viewpoint incrementally, one primitive at a time. It is not a conservative technique, instead PLP is suitable for the computation of partially correct images for use as part of time-critical rendering systems. From a very high level, PLP amounts to a modification of a simple view-frustum culling algorithm, however, it requires the computation of a special occupancy-based tessellation and the assignment to each cell of the tessellation a *solidity* value, which is used to compute a special ordering on how primitives get projected. In this paper, we detail the PLP algorithm, its main components, and implementation. We also provide experimental evidence of its performance, including results on two types of spatial tessellation (using octree- and Delaunay-based tessellations), and several datasets. We also discuss several extensions of our technique.

Index Terms—Visibility, time-critical rendering, occlusion culling, visible set, spatial tessellation.

1 Introduction

Recent advances in graphics hardware have not been able to keep up with the increase in scene complexity. In order to support a new set of demanding applications, a multitude of rendering algorithms have been developed to both augment and optimize the use of the hardware. An effective way to speed up rendering is to avoid rendering geometry that cannot be seen from the given viewpoint, such as geometry that is outside the view frustum, faces away from the viewer, or is obscured by geometry closer to the viewer. Quite possibly, the hardest part of the visibility-culling problem is to avoid rendering geometry that cannot be seen due to its being obscured by closer geometry. In this paper, we propose a new algorithm for solving the visibility culling problem. Our technique is an effective way to cull geometry with a very simple and general algorithm.

Our technique optimizes for rendering by estimating the visible set for a given frame and only rendering those polygons. It is based on computing, on demand, a priority order for the polygons that maximizes the likelihood of projecting visible polygons before occluded ones for any given scene. It does so in two steps: 1) As a preprocessing step, it computes an occupancy-based tessellation of space, which tends to have smaller spatial cells where there are more geometric primitives, e.g., polygons; 2) in real-time, rendering is performed by traversing the cells in an order determined by their intrinsic solidity (likelihood of being occluded) and some other view-dependent information. As cells are projected, their geometry is scheduled for rendering (see Fig. 1). Actual rendering is constrained by a user-defined budget, e.g., time or number of triangles.

Some highlights of our technique:

- **Budget-based rendering.** Our algorithm generates a projection ordering for the geometric primitives that mimics a depth-layered projection ordering, where primitives directly visible from the viewpoint are projected earlier in the rendering process. The ordering and rendering algorithms strictly adhere to a user-defined budget, making the PLP approach time-critical.
- **Low-complexity preprocessing.** Our algorithm requires inexpensive preprocessing that basically amounts to computing an Octree and a Delaunay triangulation on a subset of the vertices of the original geometry.
- **No need to choose occluders beforehand.** Contrary to other techniques, we do not require that occluders be found before geometry is rendered.
- **Object-space occluder fusion.** All of the occluders are found automatically during a space traversal that is part of the normal rendering loop without resorting to image-space representation.
- **Simple and fast to implement.** Our technique amounts to a small modification of a well-known rendering loop used in volume rendering of unstructured grids. It only requires negligible overhead on top of view-frustum culling techniques.

Our paper is organized as follows: In Section 2, we give some preliminary definitions and briefly discuss relevant related work. In Section 3, we propose our novel visibility-culling algorithm. In Section 4, we give some details on our prototype implementation. In Section 5, we provide experimental evidence of the effectiveness of our algorithm. In Section 6, we describe a few extensions and other avenues for future work. In Section 7, we conclude the paper with some final remarks.

- *J.T. Klosowski is with the IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598. E-mail: jklosow@us.ibm.com.*
- *C.T. Silva is with AT&T Labs-Research, 180 Park Ave., PO Box 971, Florham Park, NJ 07932. E-mail: csilva@research.att.com*

Manuscript received 15 Mar. 2000; accepted 3 Apr. 2000.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 111484.

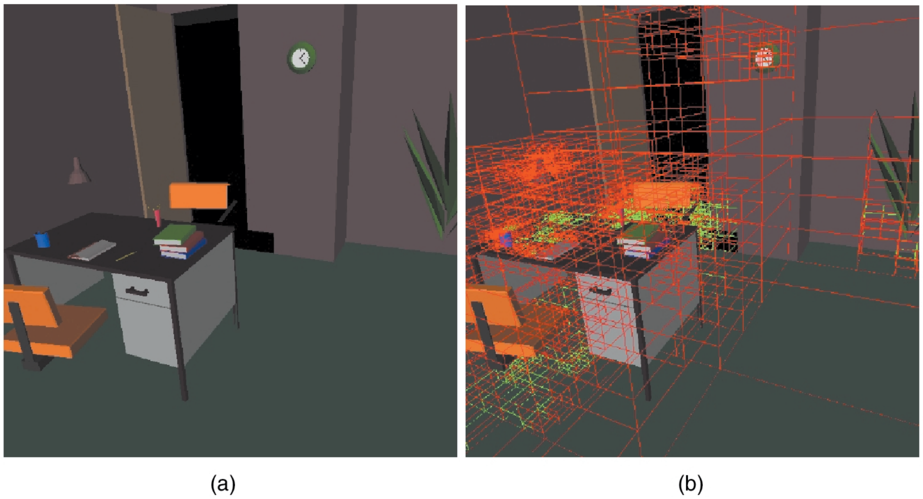


Fig. 1. The Prioritized-Layered Projection Algorithm. PLP attempts to prioritize the rendering of geometry along layers of occlusion. Cells that have been projected by the PLP algorithm are highlighted in red wireframe and their associated geometry is rendered, while cells that have not been projected are shown in green. Notice that the cells occluded by the desk are outlined in green, indicating that they have not been projected.

2 Preliminaries and Related Work

The visibility problem is defined in [9] as follows: Let the scene, $\mathcal{S}$, be composed of modeling primitives (e.g., triangles or spheres) $\mathcal{S} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and a viewing frustum defining an eye position, a view direction, and a field of view. The visibility problem encompasses finding the points or fragments within the scene that are visible, that is, connected to the eye point by a line segment that meets the closure of no other primitive. For a scene with $n = O(|\mathcal{S}|)$ primitives, the complexity of the set of visible fragments might be as high as $O(n^2)$, but, by exploiting the discrete nature of the screen, the Z-buffer algorithm [2] solves the visibility problem in time $O(n)$ since it only touches each primitive once. The Z-buffer algorithm solves the visibility problem by keeping a depth value for each pixel and only updating the pixels when geometry closer to the eye point is rendered. In the case of high depth-complexity scenes, the Z-buffer might overdraw each pixel a considerable number of times. Despite this potential inefficiency, the Z-buffer is a popular algorithm, widely implemented in hardware.

In light of the Z-buffer being widely available, and exact visibility computations being potentially too costly, one idea is to use the Z-buffer as a filter and design algorithms that lower the amount of overdraw by computing an approximation of the *visible set*. In more precise terms, define the visible set $\mathcal{V} \subset \mathcal{S}$ to be the set of modeling primitives which contribute to at least one pixel of the screen.

In computer graphics, visibility-culling research mainly focused on algorithms for computing conservative (hopefully tight) estimations of $\mathcal{V}$, then using the Z-buffer to obtain correct images. The simplest example of visibility-culling algorithms are backface and view-frustum culling [11]. Backface-culling algorithms avoid rendering geometry that faces away from the viewer, while viewing-frustum culling algorithms avoid rendering geometry that is outside of the viewing frustum. Even though both of these techniques are very effective at culling geometry, more complex techniques can lead to substantial improvements in rendering time. These techniques for tighter estimation of $\mathcal{V}$ do not come easily. In fact, most techniques proposed are quite involved and ingenious and usually require the computation of complex object hierarchies in both 3- and 2-space.

Here again, the discrete nature of the screen, and screen-space coverage tests, play a central role in literally all occlusion-culling algorithms since it paves the way for the use of screen occupancy to cull 3D geometry that projects into already occupied areas. In general, algorithms exploit this fact by 1) projecting $\mathcal{P}_i$ in front-to-back order and 2) keeping screen coverage information. Several efficiency issues are important for occlusion-culling algorithms:

1. They must operate under great time and space constraints since large amounts of geometry must be rendered in fractions of a second for real-time display.
2. It is imperative that primitives that will not be rendered be discarded as early as possible and (hopefully) not be touched at all. Global operations, such as computing a full front-to-back ordering of $\mathcal{P}_i$, should be avoided.
3. The more geometry that gets projected, the less likely the Z-buffer gets changed. In order to effectively use this fact, it must be possible to merge the effect of multiple occluders. That is, it must be possible to account for the case that neither $\mathcal{P}_0$ nor $\mathcal{P}_1$ obscures $\mathcal{P}_2$ by itself, but together they do cover $\mathcal{P}_2$. Algorithms that do not exploit *occluder-fusion* are likely to rely on the presence of large occluders in the scene.

A great amount of work has been done in visibility culling in both computer graphics and computational geometry. For those interested in the computational geometry literature, see [8], [9], [10]. For a survey of computer graphics work, see [28].

We very briefly survey some of the recent work more directly related to our technique. Hierarchical occlusion maps [29] solve the visibility problem by using two hierarchies, an object-space bounding volume hierarchy and another hierarchy of image-space occlusion maps. For each frame, objects from a precomputed database are chosen to be occluders and used to cull geometry that cannot be seen. A closely related technique is the hierarchical Z-buffer [13].

A simple and effective hardware technique for improving the performance of the visibility computations with a Z-buffer has been proposed in [23]. The idea is to add a feedback loop in the hardware which is able to check if changes would have been made to the Z-buffer when scan-converting a given primitive.[1] This hardware makes it possible to check if a complex model is visible by first querying whether an enveloping primitive (often the bounding box of the object, but, in general, one can use any enclosing object, e.g., k-dop [16]), is visible and only rendering the complex object if the enclosing object is actually visible. Using this hardware, simple hierarchical techniques can be used to optimize rendering (see [17]). In [1], another extension of graphics hardware for occlusion-culling queries is proposed.

It is also possible to perform object-space visibility culling. One such technique, described in [26], divides space into cells, which are then preprocessed for potential visibility. This technique works particularly well for architectural models. Additional object-space techniques are described in [6], [7]. These techniques mostly exploit the presence of large occluders and keep track of spatial extents over time. In [4], a technique that precomputes visibility in densely occluded scenes is proposed. They show it is possible to achieve very high occlusion rates in dense environments by precomputing simple ray-shooting checks.

In [12], a constant-frame rendering system is described. This work uses the visibility-culling from [26]. It is related to our approach in the sense that it also uses a (polygon) budget for limiting the overall rendering time. Other notable references include [3], for its level-of-detail management ideas, and [21], where a scalable rendering architecture is proposed.

3 The PLP Algorithm

In this paper, we propose the *Prioritized-Layered Projection* algorithm, a simple and effective technique for optimizing the rendering of geometric primitives. The guts of our algorithm consists of a space-traversal algorithm, which prioritizes the projection of the geometric primitives in such a way as to avoid (actually delay) projecting cells that have a small likelihood of being visible. Instead of conservatively overestimating $\mathcal{V}$, our algorithm works on a budget. At each frame, the user can provide a maximum number of primitives to be rendered, i.e., a polygon budget, and our algorithm, in its single-pass traversal over the data, will deliver what it considers to be the set of primitives which maximizes the image quality, using a solidity-based metric.

Our projection strategy is completely object-space based, and resembles[2] cell-projection algorithms used in volume rendering unstructured grids.

In a nutshell, our algorithm is composed of two parts:

Preprocessing. Here, we tessellate the space that contains the original input geometry with convex cells in the way specified in Section 3.1. During this one-time preprocessing, a collection of cells is generated in such a way as to roughly keep a uniform density of primitives per cell. Our sampling leads to large cells in unpopulated areas and small cells in areas that contain a lot of geometry.

In another similarity to volume rendering, using the number of modeling primitives assigned to a given cell (e.g., tetrahedron) we define its *solidity* value ρ, which is similar to the opacity used in volume rendering. In fact, we use a different name to avoid confusion since the accumulated solidity value used throughout our priority-driven traversal algorithm can be larger than one. Our traversal algorithm prioritizes cells based on their solidity value.

Generating such a space tessellation is not a very expensive step, e.g., taking only a minute or two minutes for a scene composed of one million triangles and, for several large datasets, can even be performed as part of the data input process. Of course, for truly large datasets, we highly recommend generating this view-independent data structure beforehand and saving it with the original data.

Rendering Loop. Our rendering algorithm traverses the cells in roughly front-to-back order. Starting from the seed cell, which, in general contains the eye position, it keeps carving cells out of the tessellation. The basic idea of our algorithm is to carve the tessellation along *layers of polygons*. We define the layering number $\zeta \in \aleph$ of a modeling primitive $\mathcal{P}$ in the following intuitive way: If we order each modeling primitive along each pixel by their positive[3]) distance to the eye point, we define $\zeta(\mathcal{P})$ to be the smallest rank of $\mathcal{P}$ over all of the pixels to which it contributes. Clearly, $\zeta(\mathcal{P}) = 1$ if, and only if, $\mathcal{P}$ is visible.

Finding the rank 1 primitives is equivalent to solving the visibility problem. Instead of solving this hard problem, the PLP algorithm uses simple heuristics. Our traversal algorithm attempts to project the modeling primitives by layers, that is, all primitives of rank 1, then 2, and so on. We do this by always projecting the cell in the front $\mathcal{F}$ (we call *the front*, the collection of cells that are immediate candidates for projection) which is least likely to be occluded according to its solidity values. Initially, the front is empty and, as cells are inserted, we estimate its accumulated solidity value to reflect its position during the traversal. (Cell solidity is defined below in Section 3.2.) Every time a cell in the front is projected, all of the geometry assigned to it is rendered. In Fig. 2, we see a snapshot of our algorithm for each of the spatial tessellations that we have implemented. The cells which have not been projected in the Delaunay

[1] In OpenGL, the technique is implemented by adding a proprietary extension that can be enabled when queries are being performed.

[2] Our cell-projection algorithm is different than the ones used in volume rendering in the following ways: 1) In volume rendering, cells are usually projected in back-to-front order, while, in our case, we project cells in *roughly* front-to-back order; 2) more importantly, we do not keep a strict depth-ordering of the cells during projection. This would be too restrictive, and expensive, for our purposes.

[3] Without loss of generality, assume $\mathcal{P}$ is in the view frustum.

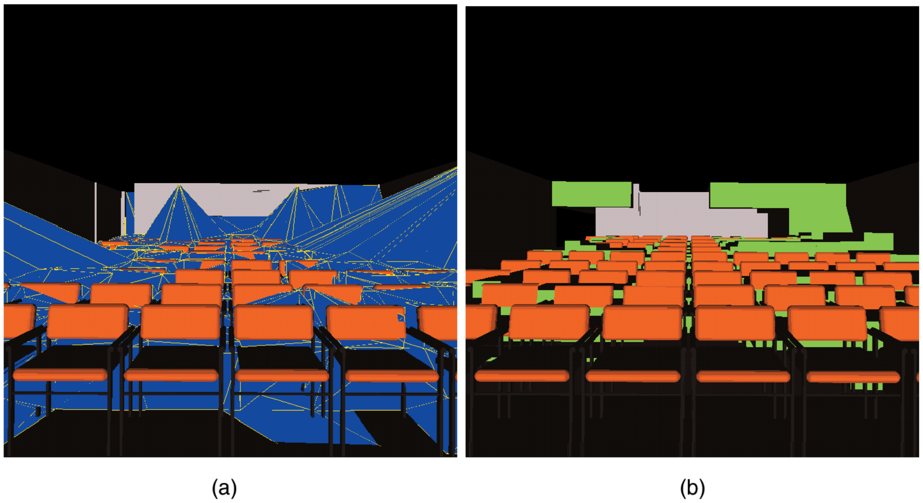


Fig. 2. The input geometry is a model of a seminar room. Snapshots of the PLP algorithm highlight the spatial tessellations that are used. The cells which have not been projected in the (a) Delaunay triangulation and (b) the octree are highlighted in blue and green, respectively. At this point in the algorithm, the geometry associated with the projected cells has been rendered.

triangulation Fig. 2a and the octree Fig. 2b are highlighted in blue and green, respectively.

There are several types of budgeting that can be applied to our technique, for example, a triangle-count budget can be used to make it time-critical. For a given budget of k modeling primitives, let $\mathcal{T}_k$ be the set of primitives our traversal algorithm projects. This set, together with $\mathcal{S}$, the set of all primitives, and $\mathcal{V}$, the set of visible primitives, can be used to define several statistics that measure the overall effectiveness of our technique. One relevant statistic is the *visible coverage ratio* for a budget of k primitives, ε_k. This is the number of primitives in the visible set that we actually render, that is, $\varepsilon_k = \frac{|\mathcal{V} \cap \mathcal{T}_k|}{|\mathcal{V}|}$. If $\varepsilon_k < 1$, we missed rendering some visible primitives.

PLP does not attempt to compute the visible set exactly. Instead, it combines a budget with its solidity-based polygon ordering. For a polygon budget of k, the best case scenario would be to have $\varepsilon_k = 1$. Of course, this would mean that PLP finds all of the visible polygons.

In addition to the visible coverage ratio ε_k, another important statistic is the number of incorrect pixels in the image produced by the PLP technique. This provides a measure as to how closely the PLP image represents the exact image produced by rendering all of the primitives.

3.1 Occupancy-Based Spatial Tessellations

The underlying data structure used in our technique is a decomposition of the 3-space covered by the scene into disjoint cells. The characteristics we required in our spatial decomposition were:

1. **Simple traversal characteristics**—must be easy and computationally inexpensive to walk from cell to cell.
2. **Good projection properties**—depth-orderable from any viewpoint (with efficient, hopefully linear-time projection algorithms available); easy to estimate screen-space coverage.
3. **Efficient space filler**—given an arbitrary set of geometry, it should be possible to sample the geometry adaptively, that is, with large cells in sparse areas, and smaller cells in dense areas.
4. **Easy to build and efficient to store**.

It is possible to use any of a number of different spatial data structures, such as kd-trees, octrees, or Delaunay triangulations. The particular use of one kind of spatial tessellation may be related to the specific dataset characteristics, although our experiments have shown that the technique works with at least two types of tessellations (octrees and Delaunay triangulations).

Overall, it seems that using low-stabbing triangulations, such as those used by Held et al. [14] (see also Mitchell et al. [18], [19] for theoretical properties of such triangulations), which are also depth-sortable (see [27], [25], [5]) are a good choice for occupancy-based tessellations. The main reason for this is that, given any path in space, these triangulations tend to minimize the traversal cost, allowing PLP to efficiently find the visible surfaces.

In order to actually compute a spatial decomposition $\mathcal{M}$ which adaptively samples the scene, we use a very simple procedure, explained in Section 4. After $\mathcal{M}$ is built, we use a naive assignment of the primitives in $\mathcal{S}$ to $\mathcal{M}$ by basically scan-converting the geometry into the mesh. Each cell $c_i \in \mathcal{M}$, has a list of the primitives from $\mathcal{S}$ assigned to it. Each of these primitives is either completely contained in c_i or it intersects one of its boundary faces. We use $|c_i|$, the number of primitives in cell c_i, in the algorithm that determines the solidity values of c_i's neighboring cells. In a final pass over the data during preprocessing, we compute the maximum number of primitives in any cell, $\rho_{max} = \max_{i \in [1 \ldots |\mathcal{M}|]} |c_i|$, to be used later as a scaling factor.

3.2 Priority-Based Traversal Algorithm

Cell-projection algorithms [27], [25], [5] are implemented using queues or stacks, depending on the type of traversal (e.g., depth-first versus breadth-first), and use some form of restrictive dependency among cells to ensure properties of the order of projection (e.g., strict back-to-front).

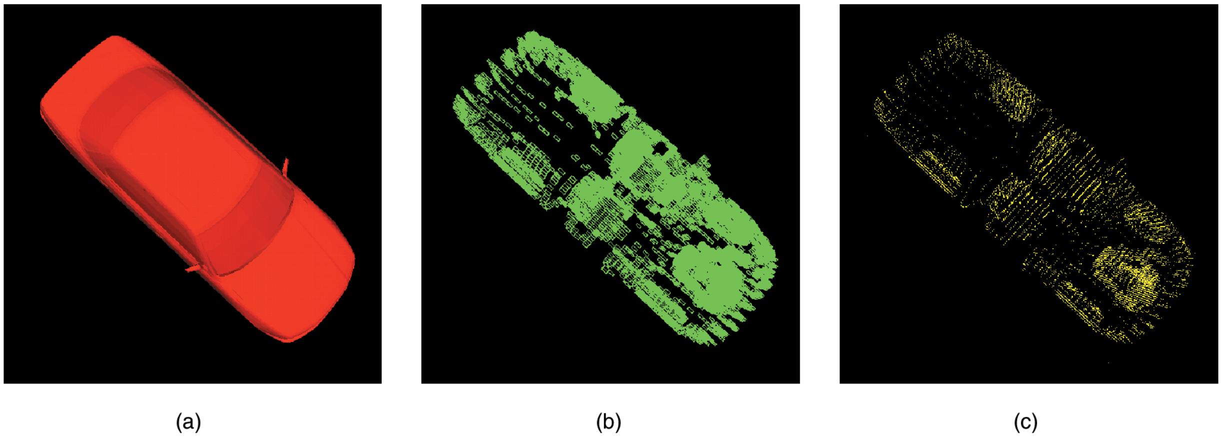


Fig. 3. Occupancy-based spatial tessellation algorithm. The input geometry, a car with an engine composed of over 160K triangles, is shown in (a). Using the vertices of the input geometry, we build an error-bounded octree, shown in (b). The centers of the leaf-nodes of the octree, shown in yellow in (c), are used as the vertices of our Delaunay triangulation.

Unfortunately, such limited and strict projection strategies do not seem general enough to capture the notion of polygon layering, which we are using for visibility culling. In order for this to be feasible, we must be able to selectively stop (or at least delay) cell-projection around some areas, while continuing in others. In effect, we would like to project cells from $\mathcal{M}$ using a layering defined by the primitives in $\mathcal{S}$. The intuitive notion we are trying to capture is as follows: If a cell c_i has been projected, and $|c_i| = \rho_{max}$, then the cells behind should wait until (at least) a corresponding *layer* of polygons in all other cells have been projected. Furthermore, in order to avoid any expensive image-based tests, we would prefer to achieve such a goal using only object-space tests.

```
Algorithm RenderingLoop()
while (empty($\mathcal{F}$) != true)
    c = min($\mathcal{F}$)
    project(c)
    if ((reached_budget() == true)
        break;
    for each n; n = cell_adjacent_to(c)
        if ((projected(n) == true)
            continue;
        ρ = update_solidity(n, c)
        enqueue(n, ρ)
```

Fig. 4. Skeleton of the *RenderingLoop* algorithm. Function $min(\mathcal{F})$ returns the minimum element in the priority queue $\mathcal{F}$. Function $project(c)$ renders all the elements assigned to cell c; it also counts the number of primitives actually rendered. Function *reached_budget*() returns **true** if we have already rendered k primitives. Function *cell_adjacent_to*(c) lists the cells adjacent to c. Function *projected*(n) returns **true** if cell n has already been projected. Function *update_solidity*(n, c) computes the updated solidity of cell n, based on the fact that c is one of its neighbors, and has just been projected. Function *enqueue*(n, ρ) places n in the queue with a solidity ρ. If n was already in the queue, this function will first remove it and reinsert it with the updated solidity value. See text for more details on *update_solidity*().

In order to achieve this goal of capturing global solidity, we extend the cell-projection framework by replacing the fixed insertion/deletion strategy queue with a metric-based queue (i.e., a priority queue) so that we can control how elements get pushed and popped based on a metric we can define. We call this priority queue, $\mathcal{F}$, the front. The complete traversal algorithm is shown in Fig. 4. In order to completely describe it, we need to provide details on solidity metrics and its update strategies.

Solidity. The notion of a cell's solidity is at the heart of our rendering algorithm shown in Fig. 4. At any given moment, cells are removed from the front (i.e., priority queue $\mathcal{F}$) in *solidity order*, that is, the cells with the smallest solidity are projected before the ones with larger solidity. The solidity of a cell B used in the rendering algorithm is not an intrinsic property of the cell by itself. Instead, we use a set of conditions to roughly estimate the visibility likelihood of the cell and make sure that cells more likely to be visible get projected before cells that are less likely to be visible.

The notion of solidity is related to how difficult it is for the viewer to see a particular cell. The actual solidity value of a cell B is defined in terms of the solidity of the cells that intersect the closure of a segment from the cell B to the eye point. The heuristic we have chosen to define the solidity value of our cells is shown in Fig. 5.

We use several parameters in computing the solidity value.

- The normalized number of primitives inside cell A, the neighboring cell (of cell B) that was just projected. This number, which is necessarily between 0 and 1, is $\frac{|A|}{\rho_{max}}$. The rationale is that the more primitives cell A contains, the more likely it is to obscure the cells behind it.
- Its position with respect to the viewpoint. We transfer a cell's solidity to a neighboring cell based

```
float function $update\_solidity(B, A)$
    /* refer to Fig. 6 */
$\rho_B = \frac{|A|}{\rho_{max}} + (\vec{v} \cdot \vec{n_B}) * \rho_A$
if $((star\_shaped(\vec{v}, B) ==$ false$)$
    $\rho_B = apply\_penalty\_factor(\rho_B)$
return $\rho_B$
```

Fig. 5. Function *update_solidity*(). This function works as if transferring accumulated solidity from cell A into cell B. ρ_B is the solidity value to be computed for cell B. $|A|$ is the number of primitives in cell A. ρ_{max} is the maximum number of primitives in any cell. $\vec{n_B}$ is the normal of the face shared by cells A and B. ρ_A is the accumulated solidity value for cell A. The maximum transfer happens if the new cell is well-aligned with the view direction $\vec{v}$ and in star-shaped position. If this is not the case, penalties will be incurred to the transfer.

on how orthogonal the face that is shared between cells is to the view direction $\vec{v}$ (see Fig. 6).

We also give preference to neighboring cells that are star-shaped [8] with respect to the viewpoint and the shared face. That is, we attempt to force the cells in the front to have their interior, e.g., their center point, visible from the viewpoint along a ray that passes through the face shared by the two cells. The reason for this is to avoid projecting cells (with low solidity values) that are occluded by cells in the front (with high solidity values) which have not been projected yet. This is likely to happen as the front expands away from an area in the scene where two densely occupied regions are nearby; we refer to such an area as a bottleneck. Examples of such areas can easily be seen in Fig. 7, which highlights our 2D prototype implementation. Actually, *forcing* the front to be star-shaped at every step of the way is too limiting a rule. This would basically produce a visibility ordering for the cells (such as the one computed in [25], [5]). Instead, we simply *penalize* the cells in the front that do not maintain this star-shaped quality.

4 Implementation Details

We have implemented a system to experiment with the ideas presented in this paper. The code is written in C++, with a mixture of Tcl/Tk and OpenGL for visualization. In order to access OpenGL functionality in a Tcl/Tk application, we use Togl [20]. In all, we have about to 10,000 lines of code. The code is very portable, and the exact same source code compiles and runs under IBM AIX, SGI IRIX, Linux, and Microsoft Windows NT. See Fig. 8 for a screen shot of our graphical user interface.

One of the reasons for the large amount of code actually comes from our flexible benchmarking capabilities. Among other functionality, our system is able to record and replay scene paths; automatically compute several different statistics about each frame as they are rendered (e.g., number of visible triangles, incorrect pixels); compute PLP traversals step-by-step; and "freeze" in the middle of a traversal to allow for the study of the traversal properties from different viewpoints.

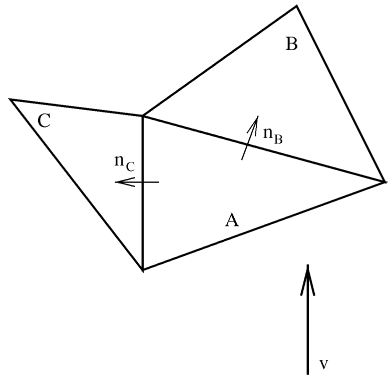


Fig. 6. Solidity Transfer. After projecting cell A, the traversal algorithm will add cells B and C to the front. Based upon the current viewing direction $\vec{v}$, cell B will accumulate more solidity from A than will cell C, however, C will likely incur the non-star-shaped penalty. $\vec{n_B}$ and $\vec{n_C}$ are the (respective) normals of the faces shared by the cell A's neighboring cells. Refer to Fig. 5 for the transfer calculation.

Here is a brief discussion of some of the important aspects of our implementation:

4.1 Rendering Data Structures

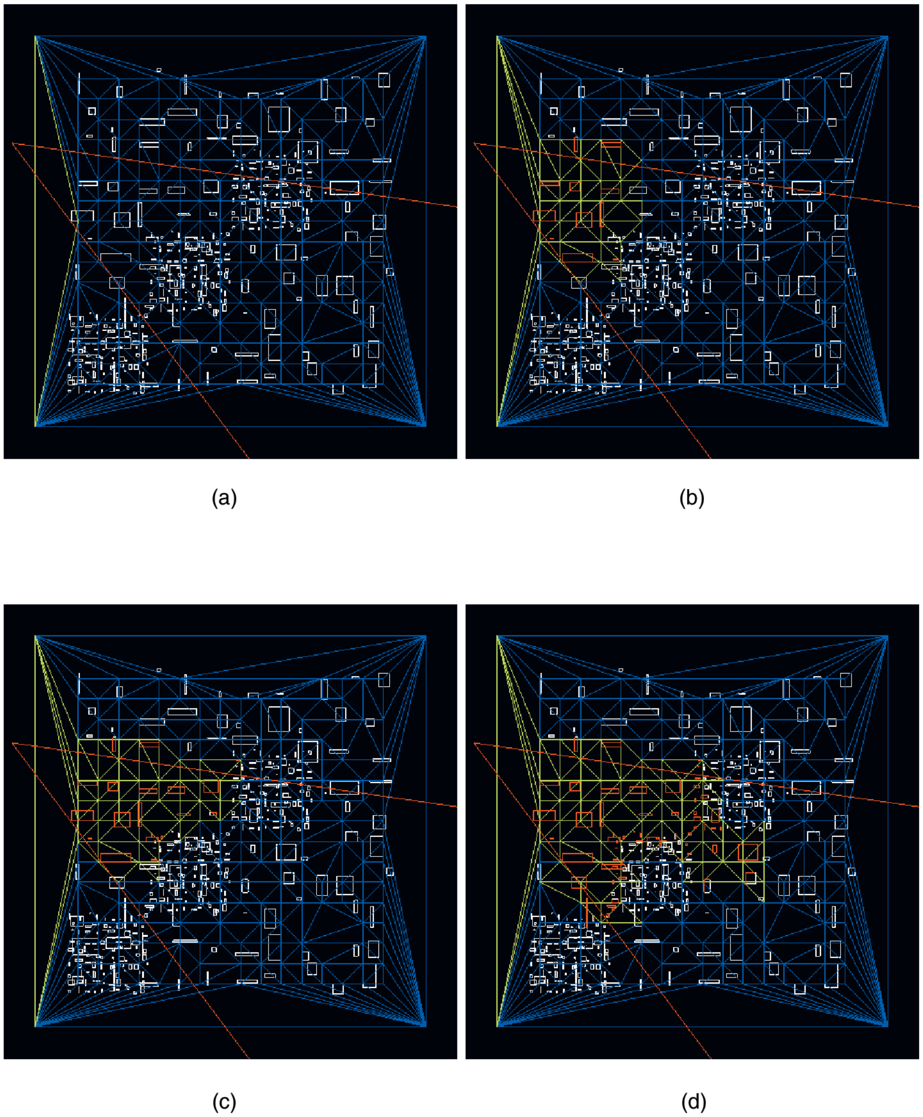
At this time, the main rendering primitive in our system is the triangle. In general, we accept and use "triangle soups" as our input. For each triangle, we save pointers to its vertices (which include color information) and a few flags, one of which is used to mark whether it has been rendered in the current traversal. At this point, we do not make any use of the fact that triangles are part of larger objects. Triangles are assigned to cells and their renderings are triggered by the actual rendering of a cell. Although triangles can (in general) be assigned to more than one cell, they will only be rendered once per frame. A cell might get partially rendered in case the triangle budget is reached while attempting to render that particular cell.

4.2 Traversal Data Structures and Rendering Loop

During rendering, cells need to be kept in a priority queue. In our current implementation, we use an STL `set` to actually implement this data structure. We use an object-oriented design, which makes it easy for the traversal rendering code to support different underlying spatial tessellations. For instance, at this time, we support both octree and Delaunay-based tessellations. Since we are using C++, it is quite simple to do this. The following methods need to be supported by any cell data structure (this list only includes methods needed for the rendering traversal; other methods are needed for initialization and triangle assignment, and also for benchmarking):

- `calculateInitialSolidityValues(int` ρ_{max}`)`—Uses the techniques presented in Section 3.2 for computing the initial solidity.
- `getSolidity()`, `setSolidity()`, `getOriginalSolidity()`—Uses the techniques presented in Section 3.2 for updating the solidity values during traversal. Solidity updates need to be adjusted for different kinds of spatial tessellations.

We use these functions to define a comparator class (`less< >`) that can be used by the STL `set` to



(a) (b)

(c) (d)

Fig. 7. Priority-based traversal algorithm. In (a), the first cell, shown in green, gets projected. The algorithm continues to project cells based upon the solidity values. Note that the traversal, in going from (b) to (c), has delayed projecting those cells with a higher solidity value (i.e., those cells less likely to be visible) in the lower-left region of the view frustum. In (d), as the traversal continues, a higher priority is given to cells likely to have visible geometry, instead of projecting the ones inside of high-depth complexity regions. Note that the star-shaped criterion was not included in our 2D implementation.

sort the different cells. Each cell also has an internal timestamp, which is used to guarantee a first-in first-out behavior when there are ties with respect to the solidity values.

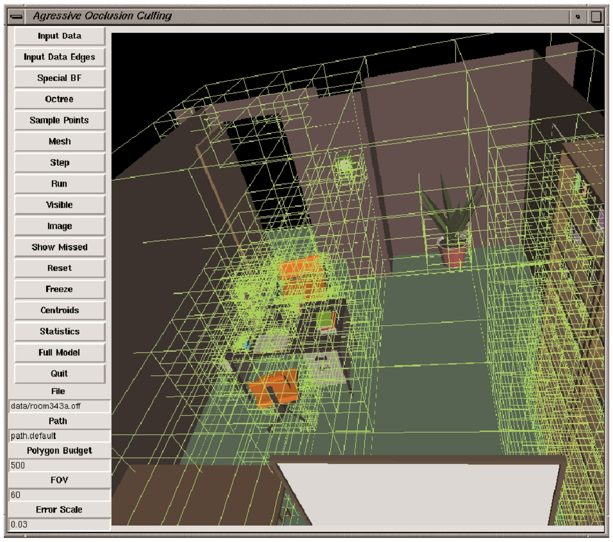
- `findCell(float vp[3])`—Find the cell that contains the viewpoint or returns that the viewpoint is outside the convex hull of the tessellation. (In order to jump start the traversal algorithm when the viewpoint is outside the tessellation, we use the cell that is closest to the viewpoint.)
- `getGeometry_VEC()`—Returns a reference to the list of primitives inside this cell.
- `getNeighbors_VEC()`—Returns a reference to the list of neighbors of this cell. (We also save the direction which identifies the face the two cells share. This is used to perform the solidity update on the neighboring cells.)

Although simple and general, STL can add considerable overhead to an implementation. In our case, the number of cells in the front has been kept relatively small and we have not noticed substantial slowdown due to STL.

The rendering loop is basically a straightforward translation of the code in Fig. 4 into C++. Triangles are rendered very naively, one by one. We mark triangles as they are rendered in order to avoid overdrawing triangles that get mapped to multiple cells. We also perform simple backface culling, as well as view-frustum culling. We take no advantage of triangle-strips, vertex arrays, or other sophisticated OpenGL features.

4.3 Space Tessellation Code

This is quite possibly the most complicated part of our implementation and it consists of two parts, one for each of the two spatial tessellations we support. There is a certain amount of shared code since it is always necessary to first



File: data/room343a.off
Path: path.default
Polygon Budget: 500
FOV: 60
Error Scale: 0.03

Fig. 8. Snapshot of our Tcl/Tk graphical user interface.

compute an adaptive sampling from the scene, for which we use a simple octree.

In more detail, in order to compute a spatial decomposition $\mathcal{M}$, which adaptively samples the scene $\mathcal{S}$, we use a very simple procedure that, in effect, just samples $\mathcal{S}$ with points, then (optionally) constructs $\mathcal{M}$ as the Delaunay triangulation of the sample points, and, finally, assigns individual primitives in $\mathcal{S}$ to $\mathcal{M}$. Fig. 3 shows our overall triangulation algorithm. Instead of accurately sampling the actual primitives (Fig. 3a), such as is done in [15], we simply construct an octree using only the original vertices (Fig. 3b); we limit the size of the octree leaves, which gives us a bound on the maximum complexity of our mesh.[4] Note that, at this point, we do not have a space partitioning where we can run PLP; instead, the octree provides a hierarchical representation of space (i.e., the nodes of the octree overlap and are nested).

Once the octree has been computed with the vertex samples, we can generate two different types of subdivisions:

- **Delaunay triangulation**—We can use the (randomly perturbed) center of the octree leaves as the vertices of our Delaunay triangulation (Fig. 3c).

  For this, we used `qhull`, software written at the Geometry Center, University of Minnesota. Our highly constrained input is bound to have several degeneracies as all the points come from nodes of an octree, therefore we randomly perturbed these points and `qhull` had no problems handling them.

  After $\mathcal{M}$ is built, we use a naive assignment of the primitives in $\mathcal{S}$ to $\mathcal{M}$ by basically scan-converting the geometry into the mesh. Each cell $c_i \in \mathcal{M}$ has a list of the primitives from $\mathcal{S}$ assigned to it. Each of these primitives is either completely contained in c_i, or it intersects one of its boundary faces.

  Each tetrahedron is represented by pointers to its vertices. Adjacency information is also required, as are a few flags for rendering purposes.

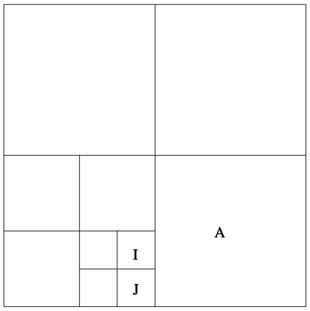


Fig. 9. Finding neighbors within the octree.

- **Octree**—Since we have already built an octree, it is obvious that we can use the same octree to compute a subdivision of space for PLP. Conceptually, this is quite simple since the leaves of the octree are guaranteed to form a subdivision of space. All that is really needed is to compute neighborhood information in the octree, for instance, looking at Fig. 9, we need to find that node A is a "face" neighbor of node I and J and vice-versa.

  Samet [22] describes several techniques for neighbor finding. The basic idea in finding the "face" neighbor of a node is to ascend the octree until the nearest common ancestor is found and to descend the octree in search of the neighbor node. In descending the octree, one needs to reflect the path taken while going up (for details, see Table 3.11 in [22]).

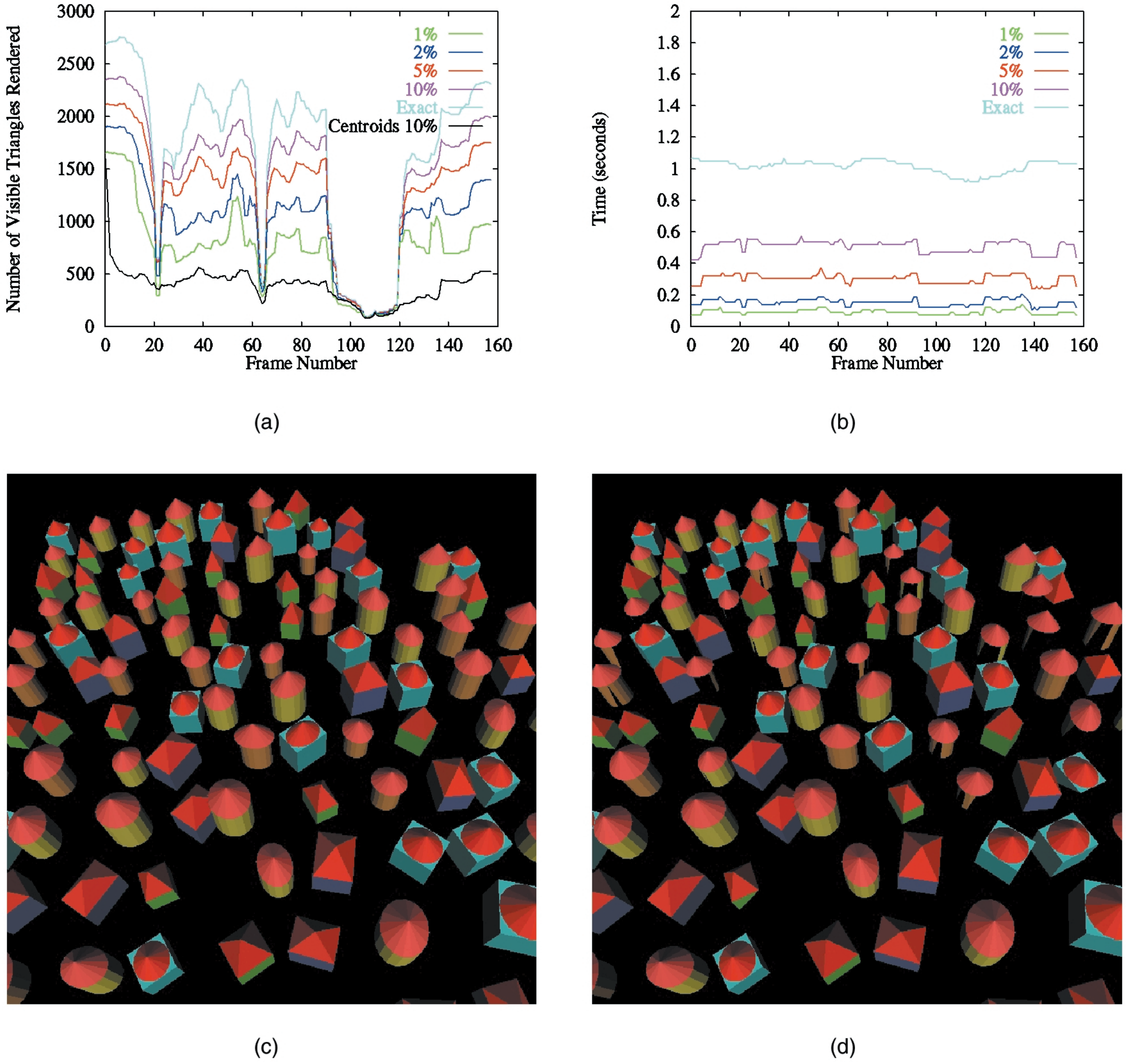
  One shortcoming with the technique as described in [22] is that it is only possible to find a neighbor at the same level or above (that is, it is possible to find that A is the "right" neighbor of I, but it is not possible to go the other way). A simple fix is to traverse the tree from the bottom to the top and allow the deeper nodes (e.g., I) to complete the neighborhood lists of nodes up in the tree (e.g., A).

Regardless of the technique used for subdivision, for the solidity calculations, we use $|c_i|$, the number of primitives in cell c_i, in the algorithm that determines the solidity values of c_i's neighboring cells. In a final pass over the data during preprocessing, we compute the maximum number of primitives in any cell, $\rho_{max} = \max_{i\in[1\ldots|\mathcal{M}|]} |c_i|$, to be used later as a scaling factor.

4.4 Computing the *Exact* Visible Set

A number of benchmarking features are currently included in our implementation. One of the most useful is the computation of the actual *exact* visible set. We estimate $\mathcal{V}$ by using the well-known item buffer technique. In a nutshell, we color all the triangles with different colors, render them, and read the frame buffer back, recording which triangles contributed to the image rendered. After rendering, all the rank-1 triangles have their colors imprinted into the frame buffer.

[4] 1) The resolution of the octree we use is very modest. By default, once an octree node has a side shorter than 5 percent of the length of the bounding box of $\mathcal{S}$, it is considered a leaf node. This has been shown to be quite satisfactory for all the experiments we have performed thus far. 2) Even though primitives might be assigned to multiple cells of $\mathcal{M}$ (we use pointers to the actual primitives), the memory overhead has been negligible. See Section 5.1.



(a)

(b)

(c)

(d)

Fig. 10. CITY results. (a) The top curve, labeled Exact, is the number of visible triangles for each given frame. The next four curves are the number of visible triangles PLP finds with a given budget. From top to bottom, budgets of 10 percent, 5 percent, 2 percent, and 1 percent are reported. The bottom curve is the number of visible triangles that the centroid sorting algorithm finds. (b) Rendering times in seconds for each curve shown in (a), with the exception of the centroid sorting algorithm, which required 4-5 seconds per frame. (c) Image of all the visible triangles. (d) Image of the 10 percent PLP visible set.

4.5 Centroid-Ordered Rendering

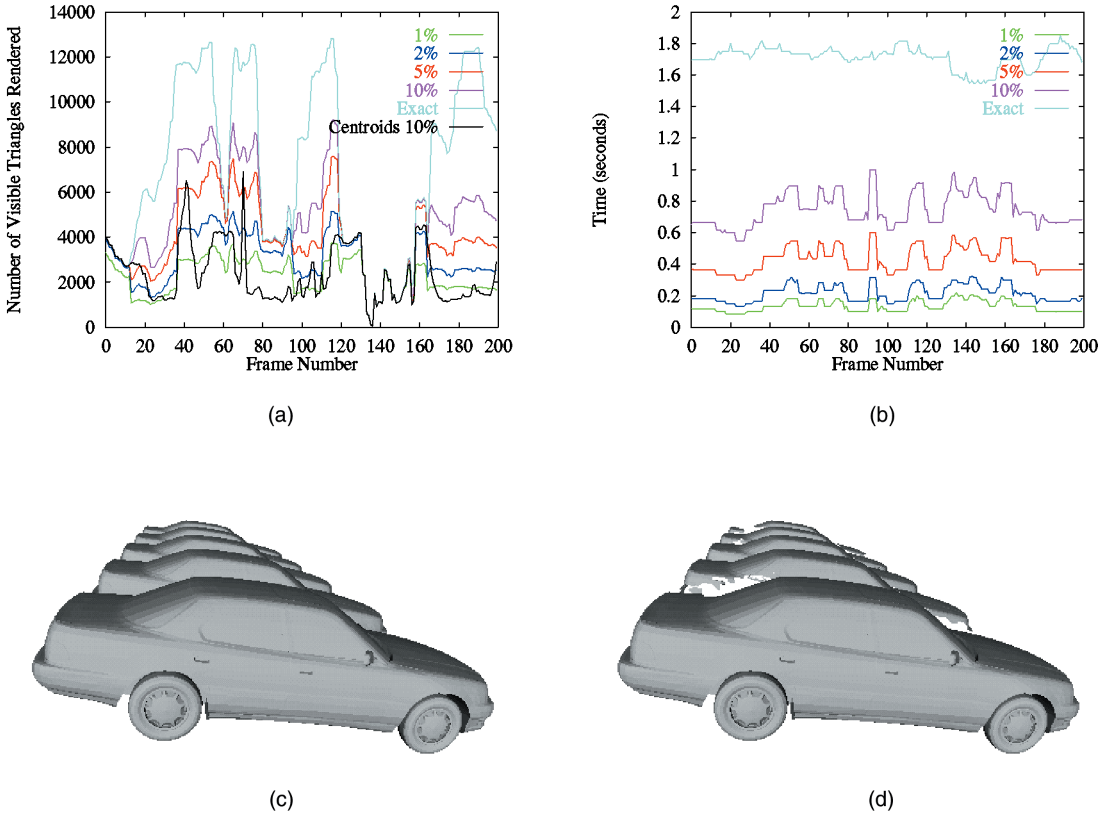
In order to have a basis for comparison, we implemented a simple ordering scheme based on sorting the polygons with respect to their centroid and rendering them in that order up to the specified budget. Our implementation of this feature tends to be slow for large datasets, as it needs to sort all of the triangles in $\mathcal{S}$ at each frame.

5 Experimental Results

We performed a series of experiments in order to determine the effectiveness of PLP's visibility estimation. Our experiments typically consist of recording a flight path consisting of several frames for a given dataset, then playing back the path while varying the rendering algorithm used. We have four different strategies for rendering: 1) rendering every triangle in the scene at each frame, 2) centroid-based budgeting, 3) PLP with octree-based tessellation, and 4) PLP with Delaunay triangulation. During path playback, we also change the parameters when appropriate (e.g., varying the polygon budget for PLP). Our primary benchmark machine is an IBM RS/6000 595 with a GXT800 graphics adapter. In all our experiments, rendering was performed using OpenGL with Z-buffer and lighting calculations turned on. In addition, all three algorithms perform view-frustum and backface culling to avoid rendering those triangles that clearly will not contribute to the final image. Thus, any benefits provided by PLP will be on top of the benefits provided by traditional culling techniques.

We report experimental results on three datasets:

Room 306 of the Berkeley SODA Hall (ROOM). This model has approximately 45K triangles (see Figs. 13 and 14) and consists of a number of chairs in what appears to be a reasonably large seminar room. This is a difficult model to perform visibility culling on since the number of visible triangles along a path varies quite a bit with respect to the total size of the dataset, in fact, in the path



(a)

(b)

(c)

(d)

Fig. 11. 5CBEM results. (a) The top curve, labeled Exact, is the number of visible triangles for each given frame. The next four curves are the number of the visible triangles PLP finds with a given budget. From top to bottom, budgets of 10 percent, 5 percent, 2 percent, and 1 percent are reported. The bottom curve is the number of visible triangles that the centroid sorting algorithm finds. (b) Rendering times in seconds for each curve shown in (a), with the exception of the centroid sorting algorithm, which required 6-7 seconds per frame. (c) Image of all the visible triangles. (d) Image of the 10 percent PLP visible set.

we use, this number ranged from 1 percent to 20 percent of the total number of triangles.

City Model (CITY). The city model is composed of over 500K triangles (Fig. 10c). Each house has furniture inside and, while the number of triangles is large, the actual number of visible triangles per frame is quite small.

5 Car Body/Engine Model (5CBEM). This model has over 810K triangles (Fig. 11c). It is composed of five copies of an automobile body and engine.

5.1 Preprocessing

Preprocessing involves computing an octree of the model, then (optionally) computing a Delaunay triangulation of points defined by the octree (which is performed by calling `qhull`), and, finally, assigning the model geometric primitives to the spatial tessellation generated by `qhull`.

For the CITY model, preprocessing took 70 seconds and generated 25K tetrahedra. Representing each tetrahedron requires less than 100 bytes (assuming the cost of representing the vertices is amortized among several tetrahedra), leading to a memory overhead for the spatial tessellation on the order of 2.5MB. Another source of overhead comes from the fact that some triangles might be multiply assigned to tetrahedra. The average number of times a triangle is referenced is 1.80, costing 3.6 MB of memory (used for triangle pointers). The total memory overhead (on top of the original triangle lists) is 6.1 MB, while storing all the triangles alone (the minimal amount of memory necessary to render them) already costs 50 MB. So, PLP costs an extra 12 percent in memory overhead.

For the 5CBEM model, preprocessing took 135 seconds (also including the `qhull` time) and generated 60K tetrahedra. The average number of tetrahedra that points to a triangle is 2.13, costing 14.7 MB of memory. The total memory overhead is 20 MB and storing the triangles takes approximately 82 MB. So, PLP costs an extra 24 percent in memory overhead.

Since PLP's preprocessing only takes a few minutes, the preprocessing is performed online when the user requests a given dataset. We also support offline preprocessing by simply writing the spatial tessellation and the triangle assignment to a file.

5.2 Rendering

We performed several rendering experiments. During these experiments, the flight path used for the 5CBEM is composed of 200 frames. The flight path for the CITY has 160 frames. The flight path for the ROOM has 235 frames. For each frame of the flight path, we computed the following statistics:

1. The exact number of visible triangles in the frame, estimated using the item-buffer technique.

TABLE 1
Visible Coverage Ratio

Dataset/Budget	1%	2%	5 %	10%
City Model	51%	66%	80%	90%
5 Car Body/Engine Model	44%	55%	67%	76%

The table summarizes ε_k for several budgets on two large models. The city model has 500K polygons and the five car body/engine model has 810K polygons. For a budget of 1 percent, PLP is able to find over 40 percent of the visible polygons in either model.

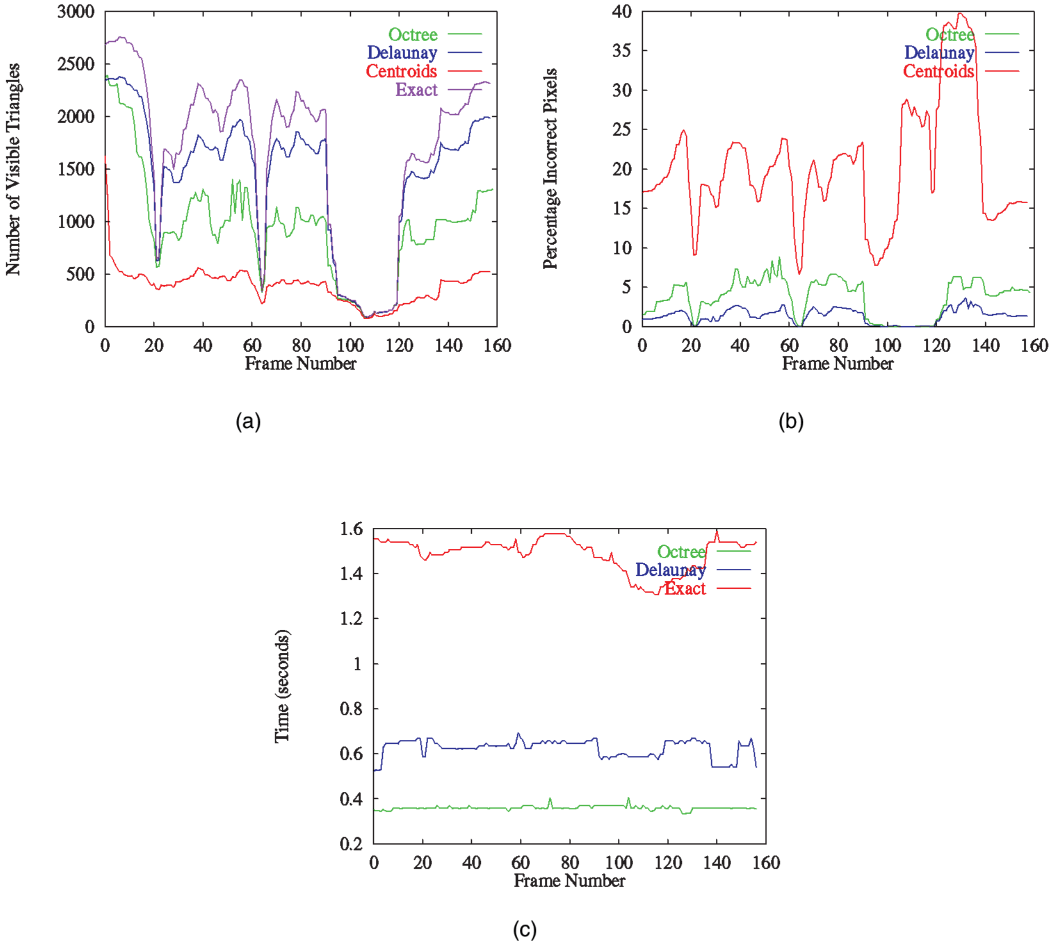
2. The number of visible triangles PLP was able to find for a given triangle budget. We varied the budget as follows: 1 percent, 2 percent, 5 percent, and 10 percent of the number of triangles in the dataset.
3. The number of visible triangles the centroid-based budgeting was able to find under a 10 percent budget.
4. The number of wrong pixels generated by PLP.
5. Time (all times are reported in seconds) to render the whole scene.
6. Time PLP took to render a given frame.
7. Time the centroid-based budgeting took to render a given frame.

Several of the results (in particular, 1, 2, 3, 5, and 6) are shown in Table 1 and Figs. 10 and 11, which show PLP's overall performance and how it compares to the centroid-sorting based approach. The centroid rendering time (7) is mostly frame-independent since the time is dominated by the sorting, which takes 6-7 seconds for the 5CBEM model, and 4-5 seconds for the CITY model. We collected the number of wrong pixels (4) on a frame-by-frame basis. We report worst-case numbers. For the CITY model, PLP gets as many as 4 percent of the pixels wrong; for the 5CBEM model, this number goes up and PLP misses as many as 12 percent of the pixels in any given frame.

The other figures focus on highlighting specific features of our technique, and compare the octree and Delaunay-based tessellations.

5.2.1 Speed and Accuracy Comparisons on the CITY Model

Fig. 12c shows the rendering times of the different algorithms and compares them with the rendering of the entire model geometry. For a budget of 10 percent, the Delaunay triangulation was over two times faster, while the



(a)

(b)

(c)

Fig. 12. This figure illustrates the quantitative differences among the different rendering techniques for each frame of the CITY path. In each plot, we report results for each rendering technique (centroid, octree-based PLP, and Delaunay-based PLP, respectively). In (a), we show the percentage of the visible polygons that each technique was able to find. In (b), we show the number of incorrect pixels in the images computed with each technique.

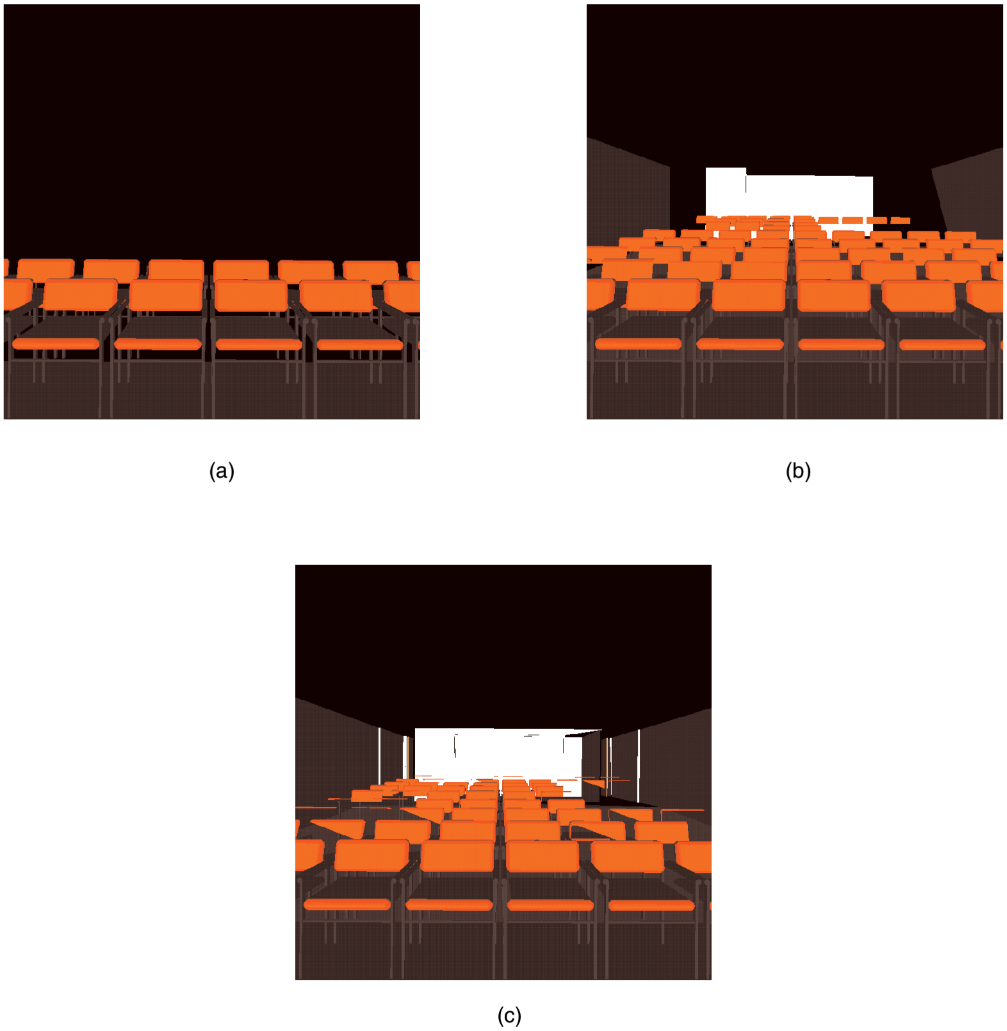


Fig. 13. This figure illustrates the qualitative differences among the different rendering techniques on each frame of the ROOM path. The three images show the actual rendered picture achieved with each rendering technique (centroid, octree-based PLP, and Delaunay-based PLP respectively).

octree approach was about four times faster. We have not included the timings for the centroid-sorting method as our implementation was straightforward and naively sorted all of the triangles for each of the frames. Fig. 12a highlights the effectiveness of our various methods for a budget of 10 percent of the total number of triangles, showing the number of visible triangles that were found. The magenta curve shows the exact number of visible triangles for each frame of this path. In comparison, the Delaunay triangulation was very successful, finding an average of over 90 percent of the visible triangles. The octree was not as good in this case and averaged only 64 percent. However, this was still considerably better than the centroid-sorting approach, which averaged only 30 percent. Fig. 12b highlights the effectiveness of the PLP approaches. In the worst case, the Delaunay triangulation version produced an image with 4 percent of the pixels incorrect with respect to the actual image. The octree version of PLP was a little less effective, generating images with at most 9 percent of the pixels incorrect. However, in comparison with the centroid-sorting method, which rendered images with between 7-40 percent of the pixels incorrect, PLP has done very well.

5.2.2 Visual and Quantitative Quality on the ROOM Model

Figs. 13 and 14 show one of the viewpoints for the path in the Seminar Room dataset. Most of the geometry is made up of the large number of chairs, with relatively few triangles being contributed by the walls and floor. From a viewpoint on the outside of this room, the walls would be very good occluders and would help make visibility culling much easier. However, once the viewpoint is in the interior sections of this room, all of these occluders are invalidated (except with respect to geometry outside of the room), and the problem becomes much more complicated. For a budget of 10 percent of the triangles, we provide figures to illustrate the effectiveness of our PLP approaches, as well as the centroid-sorting algorithm. Fig. 13 shows the images rendered by the centroid method, the octree method, and the Delaunay triangulation method, respectively. The image produced by the octree in this case is the best overall, while the centroid-sorting image clearly

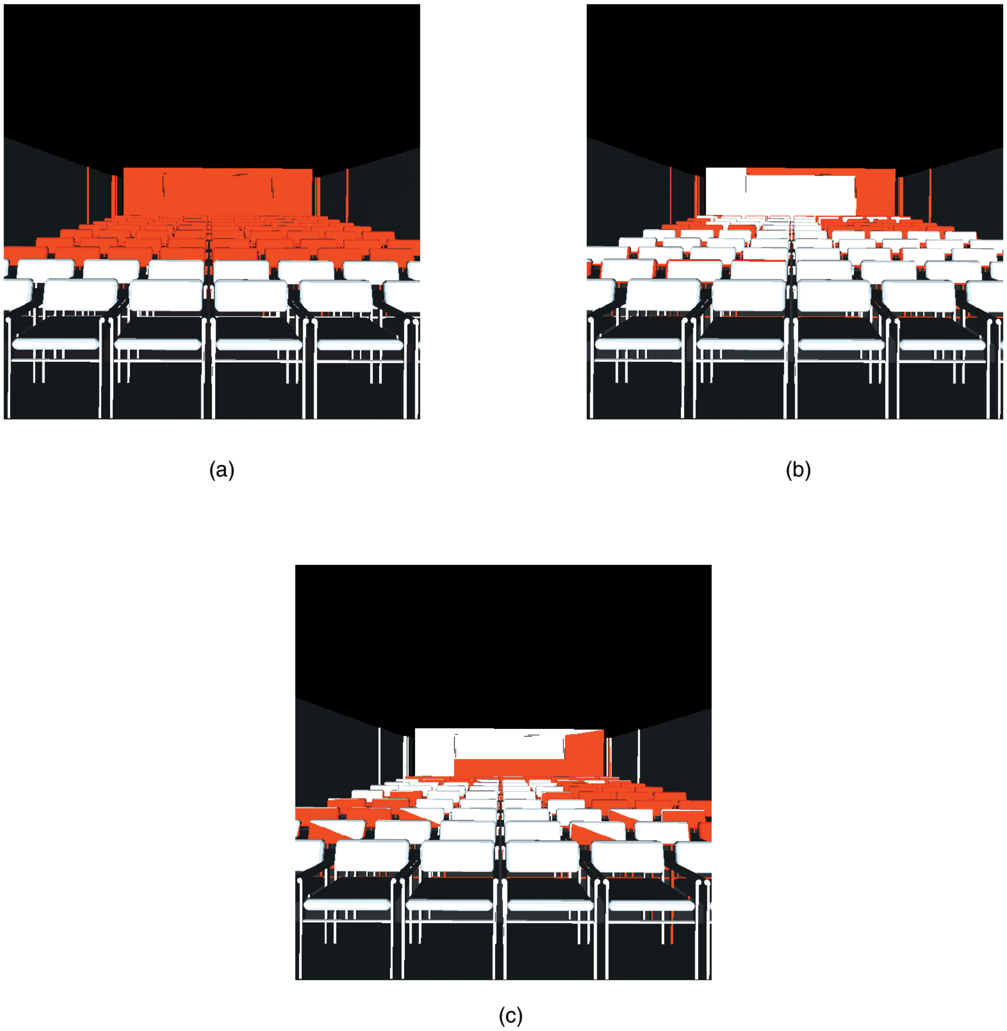


Fig. 14. This figure illustrates the qualitative differences among the different rendering techniques on a single frame of the ROOM. The three images show the missed polygons, rendered in red, to highlight which portion of the image a given technique rendered incorrectly.

demonstrates the drawback of using such an approach. To better illustrate where the algorithms are failing, Fig. 14 shows exactly the pixels which were drawn correctly, in white, and those drawn incorrectly, in red. Further quantitative information can be seen in Fig. 15. In fact, it is quite interesting that, in terms of the overall number of visible primitives, the centroid technique actually does quite well. On the other hand, it keeps rendering a large number of incorrect pixels.

5.2.3 Summary of Results

PLP seems to do quite a good job at finding visible triangles. In fact, looking at Figs. 10a and 11a, we see a remarkable resemblance between the shape of the curve plotting the exact visible set and PLP's estimations. In fact, as the budget increases, the PLP curves seem to smoothly converge to the exact visible set curve. It is important to see that this is not a random phenomena. Notice how the centroid-based budgeting curve does not resemble the visible set curves. Clearly, there seems to be some relation between our heuristic visibility measure (captured by the solidity-based traversal) and actual visibility, which cannot be captured by a technique that relies on distance alone.

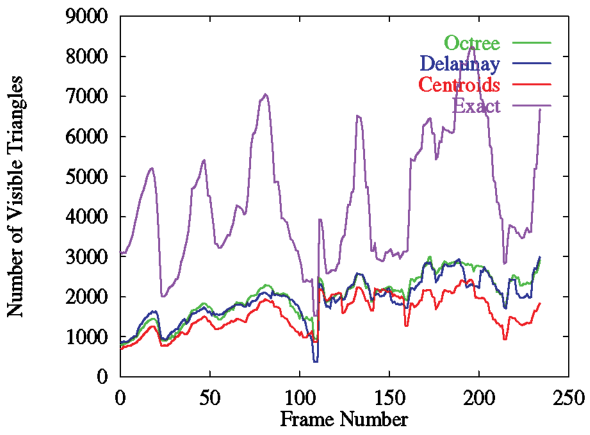
Still, we would like PLP to do a better job at approximating the visible set. For this, it is interesting to see where it fails. In Figs. 10d and 11d, we have 10 percent-budget images. Notice how PLP loses triangles in the back of the cars (in Fig. 11d) since it estimates them to be occluded.

With respect to speed, PLP has very low overhead. For 5CBEM, at 1 percent, we can render useful images at over 10 times the rate of the completely correct image and, for CITY, at 5 percent, we can get 80 percent of the visible set and still have four times faster rendering times.

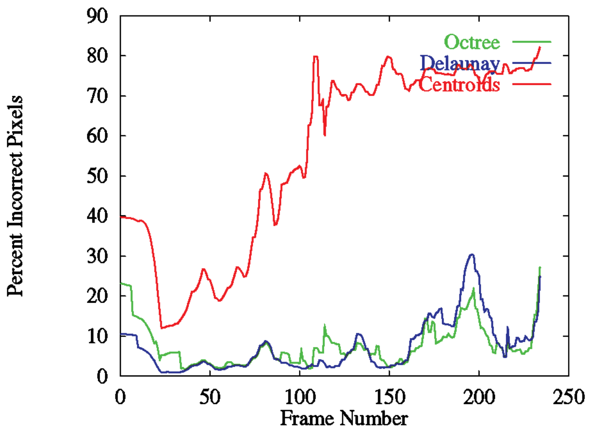
Overall our experiments have shown that: 1) PLP can be applied to large data, without requiring large amounts of preprocessing; 2) PLP is able to find a large amount of visible geometry with a very low budget; 3) PLP is useful in practice, making it easier to inspect large objects and in culling geometry that cannot be seen.

6 Algorithm Extensions and Future Work

In this section, we mention some of the possible extensions of this work:



(a)



(b)

Fig. 15. This figure illustrates the quantitative differences among the different rendering techniques on a single frame of the ROOM. In each plot, we report results for each rendering technique (centroid, octree-based PLP, and Delaunay-based PLP, respectively). In (a), we show the percentage of the visible polygons that each technique was able to find. In (b), we show the number of incorrect pixels in the images computed with each technique.

1. Occlusion-culling techniques which rely on being able to use the z-buffer values to cull geometry, e.g., HOM [29], HP's occlusion-culling hardware [23], can potentially be sped up considerably with PLP.

   Take, for instance, the HP fx6 graphics accelerator. Severson [24] estimates that performing an occlusion-query with a bounding box of an object on the fx6 is equivalent to rendering about 190 25-pixel triangles. This indicates that a naive approach, where objects are constantly checked for being occluded, might actually hurt performance and not achieve the full potential of the graphics board. In fact, it is possible to slow down the fx6 considerably if one is unlucky enough to project the polygons in a back to front order (because none of the primitives would be occluded).

   Since PLP is able to determine a large number of the visible polygons at low cost in terms of projected triangles (e.g., PLP can find over 40 percent of the visible polygons while only projecting 1 percent of the original geometry). An obvious approach would be to use PLP's traversal for rendering a first "chunk" of geometry, then use the hardware to cull away unprojected geometry. Assuming PLP does its job, the z-buffer should be relatively complete, and a much larger percentage of the tests should lead to culling.

   A similar argument is valid for using PLP with HOM [29]. In this case, PLP can be used to replace the occluder selection piece of the algorithm, which is time consuming, and involves a nontrivial "occlusion preserving simplification" procedure.
2. Another potential use of the PLP technique is in level-of-detail (LOD) selection. The PLP traversal algorithm can estimate the proportion of a model that is currently visible, which would allow us to couple visibility with the LOD selection process, as opposed to relying only on screen-space coverage tests.
3. Related to 1 and 2, it would be interesting to explore techniques which automatically can adjust the PLP budget to the optimum amount to increase the quality of the images and, at the same time, decrease the rendering cost. Possibly, ideas from [12] could be adapted to our framework.

Besides the extensions cited above, we would like to better understand the relation of the solidity measure to the actual set of rendered polygons. Changing our solidity value computation could possibly lead to even better performance, for example, accounting for front facing triangles in a given cell by considering their normals with respect to the view direction. The same is true for the mesh generation. Another class of open problems are related to further extensions in the front-update strategies. At this time, a single cell is placed in the front, after which the PLP traversal generates an ordering for all cells. We cut this tree by using a budget. It would be interesting to exploit the use of multiple initial seeds. Clearly, the better the initial guess of what's visible, the easier it is to continue projecting visible polygons.

7 Conclusions

In this paper, we proposed the Prioritized-Layered Projection algorithm. PLP renders geometry by carving out space along layers while keeping track of the solidity of these layers as it goes along. PLP is very simple, requiring only a suitable tessellation of space where solidity can be computed (and is meaningful). The PLP rendering loop is a priority-based extension of the traversal used in depth-ordering cell projection algorithms developed originally for volume rendering.

As shown in this paper, PLP can be used with many different spatial tessellations, for example, octrees or Delaunay triangulations. In our experiments, we have found that the octree method is typically faster than the

Delaunay method due to its simple structure. However, it does not appear to perform as well as the Delaunay triangulation in terms of capturing our notion of polygon layering.

We use PLP as our primary visibility-culling algorithm. Two things are most important to us. First, there is no offline preprocessing involved, that is, no need to simplify objects, pregenerate occluders, and so on. Second, its flexibility to adapt to multiple machines with varying rendering capabilities. In essence, in our application, we were mostly interested in obtaining good image accuracy across a large number of machines with minimal time and space overheads. For several datasets, we can use PLP to render only 5 percent of a scene and still be able to visualize over 80 percent of the visible polygons. If this is not accurate enough, it is simple to adjust the budget for the desired accuracy. A nice feature of PLP is that the visible set is stable, that is, the algorithm does not have major popping artifacts as it estimates the visible set from nearby viewpoints.

Acknowledgments

The authors would like to thank Dirk Bartz and Michael Meißner for the city model, Bengt-Olaf Schneider for suggesting adding star-shape constraints to the front, Fausto Bernardini, Paul Borrel, João Comba, William Horn, and Gabriel Taubin for suggestions and help throughout the project. We thank Craig Wittenbrink for help with the occlusion-culling capabilities of the HP fx series accelerators, the Geometry Center of the University of Minnesota for `qhull` and `geomview`, and the University of California, Berkeley for the SODA Hall model used in some of our experiments.

References

[1] D. Bartz, M. Meißner, and T. Hüttner, "Extending Graphics Hardware for Occlusion Queries in Opengl," *Proc. Workshop Graphics Hardware '98*, pp. 97-104, 1998.

[2] E.E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," PhD thesis, Dept. of Computer Science, Univ. of Utah, Dec. 1974.

[3] J.H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Comm. ACM*, vol. 19, no. 10, pp. 547-554, Oct. 1976.

[4] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario, "Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes," *Computer Graphics Forum*, vol. 17, no. 3, pp. 243-253, 1998.

[5] J. Comba, J.T. Klosowski, N. Max, J.S.B. Mitchell, C.T. Silva, and P.L. Williams, "Fast Polyhedral Cell Sorting for Interactive Rendering of Unstructured Grids," *Computer Graphics Forum*, vol. 18, no. 3, pp. 369-376, Sept. 1999.

[6] S. Coorg and S. Teller, "Real-Time Occlusion Culling for Models with Large Occluders," *Proc. 1997 Symp. Interactive 3D Graphics*, 1997.

[7] S. Coorg and S. Teller, "Temporally Coherent Conservative Visibility," *Proc. 12th Ann. ACM Symp. Computational Geometry*, pp. 78-87, 1996.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications.* Berlin: Springer-Verlag, 1997.

[9] D.P. Dobkin and S. Teller, "Computer Graphics," *Handbook of Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke, eds., chapter 42, pp. 779-796, Boca Raton, Fla.: CRC Press LLC, 1997.

[10] S.E. Dorward, "A Survey of Object-Space Hidden Surface Removal," *Int'l J. Computational Geometry Applications*, vol. 4, pp. 325-362, 1994.

[11] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics, Principles and Practice*, second ed. Reading, Mass.: Addison-Wesley, 1990.

[12] T.A. Funkhouser and C.H. Séquin, "Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments," *Computer Graphics (SIGGRAPH '93 Proc.)*, J.T. Kajiya, ed., vol. 27, pp. 247-254, Aug. 1993.

[13] N. Greene, M. Kass, and G. Miller, "Hierarchical Z-Buffer Visibility," *Computer Graphics Proc., Ann. Conf. Series*, pp. 231-240, 1993.

[14] M. Held, J.T. Klosowski, and J.S.B. Mitchell, "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs," *Proc. Seventh Canadian Conf. Computational Geometry*, pp. 205-210, 1995.

[15] P.M. Hubbard, "Approximating Polyhedra with Spheres for Time-Critical Collision Detection," *ACM Trans. Graphics*, vol. 15, no. 3, pp. 179-210, July 1996.

[16] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan, "Efficient Collision Detection Using Bounding Volume Hierarchies of k-dops," *IEEE Trans. Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21-36, Jan.-Mar. 1998.

[17] M. Meißner, D. Bartz, T. Hüttner, G. Müller, and J. Einighammer, "Generation of Subdivision Hierarchies for Efficient Occlusion Culling of Large Polygonal Models," Technical Report WSI-99-13, ISSN 0946-3852, 1999.

[18] J.S.B. Mitchell, D.M. Mount, and S. Suri, "Query-Sensitive Ray Shooting," *Proc. 10th Ann. ACM Symp. Computational Geometry*, pp. 359-368, 1994.

[19] J.S.B. Mitchell, D.M. Mount, and S. Suri, "Query-Sensitive Ray Shooting," *Int'l J. Computational Geometry Applications*, vol. 7, no. 4, pp. 317-347, Aug. 1997,

[20] B. Paul and B. Bederson, "Togl—A Tk OpenGL Widget," http://Togl.sourceforge.net.

[21] J. Rohlf and J. Helman, "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics," *Proc. SIGGRAPH '94, Computer Graphics Proc., Ann. Conf. Series*, A. Glassner, ed., pp. 381-395, July 1994.

[22] H. Samet, *Applications of Spatial Data Structures.* Reading, Mass.: Addison-Wesley, 1990.

[23] N. Scott, D. Olsen, and E. Gannet, "An Overview of the Visualize fx Graphics Accelerator Hardware," *The Hewlett-Packard J.*, pp. 28-34, May 1998.

[24] K. Severson, "VISUALIZE Workstation Graphics for Windows NT," HP product literature.

[25] C.T. Silva, J.S.B. Mitchell, and P. Williams, "An Interactive Time Visibility Ordering Algorithm for Polyhedral Cell Complexes," *Proc. ACM/IEEE Volume Visualization Symp. '98*, pp. 87-94, Nov. 1998.

[26] S.J. Teller and C.H. Séquin, "Visibility Preprocessing for Interactive Walkthroughs," *Computer Graphics (SIGGRAPH '91 Proc.)*, T.W. Sederberg, ed., vol. 25, p. 61-69, July 1991.

[27] P.L. Williams, "Visibility Ordering Meshes Polyhedra," *ACM Trans. Graphics*, vol. 11, no. 2, 1992.

[28] H. Zhang, "Effective Occlusion Culling for the Interactive Display of Arbitrary Models," PhD thesis, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, 1998.

[29] H. Zhang, D. Manocha, T. Hudson, and K.E. Hoff III, "Visibility Culling Using Hierarchical Occlusion Maps," *SIGGRAPH 97 Conf. Proc., Ann. Conf. Series*, T.Whitted, ed., pp. 77-88, Aug. 1997.

James T. Klosowski received a BS in computer science and mathematics from Fairfield University in 1992, and an MS and a PhD in applied mathematics from the State University of New York at Stony Brook in 1994 and 1998, respectively. He is a research staff member at the IBM Thomas J. Watson Research Center. His main research interests are in computer graphics, visualization and applied computational geometry. While a graduate student, his research focused on applied computational geometry and computer graphics. He received the Catacosinos Fellowship for Excellence in Computer Science for his work in real-time collision detection. His research interests in computer graphics include interactive visualization of large datasets, collision detection, volume rendering, and adaptive network graphics. Recently, his research has focused on visibility culling and simplification of complex geometric models.

Cláudio Silva has a Bachelor's degree in mathematics from the Federal University of Ceara (Brazil), and MS and PhD degrees in computer science from the State University of New York at Stony Brook. He is a senior member of the technical staff in the Information Visualization Research Department at AT&T Labs-Research. His current research focuses on architectures and algorithms for building scalable displays, rendering techniques for large datasets, and algorithms for graphics hardware. Before joining AT&T, he was a research staff member at the graphics group at IBM T.J. Watson Research Center. There, he worked on 3D compression (as part of the MPEG-4 standardization committee), 3D scanning, visibility culling, and volume rendering. While a student and, later, as a U.S. National Science Foundation postdoctoral researcher, he worked at Sandia National Labs, where he developed large-scale scientific visualization algorithms and tools for handling massive datasets. His main research interests are in graphics, visualization, applied computational geometry, and high-performance computing. He has published more than 30 papers in international conferences and journals and presented courses at Eurographics and ACM Siggraph conferences. He serves on the committee for the IEEE Visualization 2000 Conference and the IEEE Volume Visualization and Graphics Symposium 2000. He is a member of the ACM, ACM Siggraph, IEEE, and IEEE Computer Society.

Efficient Conservative Visibility Culling Using The Prioritized-Layered Projection Algorithm

James T. Klosowski* Cláudio T. Silva†

Abstract

We propose a novel conservative visibility culling technique based on the Prioritized-Layered Projection (PLP) algorithm. PLP is a time-critical rendering technique that computes, for a given viewpoint, a partially correct image by rendering only a subset of the geometric primitives, those that PLP determines to be most likely visible. Our new algorithm builds on PLP and provides an efficient way of finding the remaining visible primitives. We do this by adding a second phase to PLP which uses image-space techniques for determining the visibility status of the remaining geometry. Another contribution of our work is to show how to efficiently implement such image-space visibility queries using currently available OpenGL hardware and extensions. We report on the implementation of our techniques on several graphics architectures, analyze their complexity, and discuss a possible hardware extension that has the potential to further increase performance.

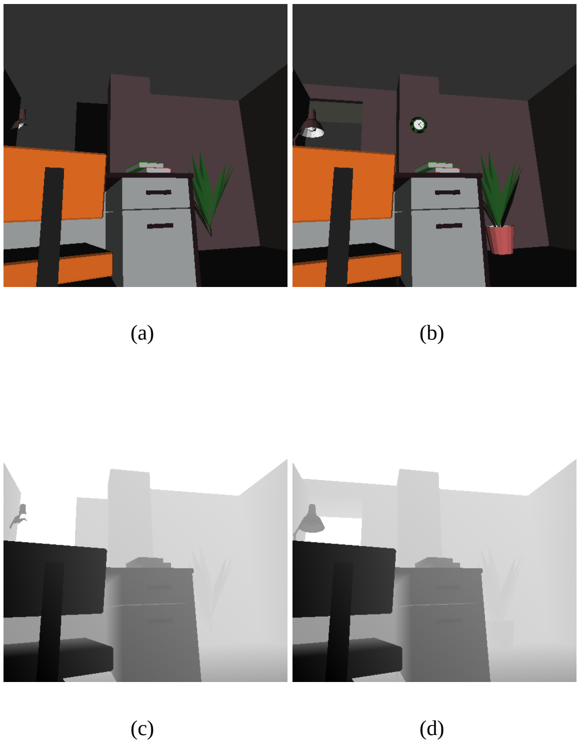
Index Terms Conservative visibility, occlusion culling, interactive rendering

1 Introduction

Interactive rendering of very large data sets is a fundamental problem in computer graphics. Although graphics processing power is increasing every day, its performance has not been able to keep up with the rapid increase in data set complexity. To address this shortcoming, techniques are being developed to reduce the amount of geometry that is required to be rendered, while still preserving image accuracy.

Occlusion culling is one such technique whose goal is to determine which geometry is hidden from the viewer by other geometry. Such occluded geometry need not be processed by the graphics hardware since it will not contribute to the final image produced on the screen. Occlusion culling, also known as visibility culling[1], is especially effective for scenes with high depth complexity, due to the large amount of occlusion that occurs. In such situations, much geometry can often be eliminated from the rendering process. Occlusion culling techniques are usually conservative, producing images that are identical to those that would result from rendering all of the geometry. However, they can also be approximate techniques that produce images that are mostly correct, in exchange for even greater levels of interactivity. The approximate approaches are more effective when only a few pixels are rendered incorrectly, limiting any artifacts that are perceivable to the viewer.

The Prioritized-Layered Projection (PLP) algorithm, introduced by Klosowski and Silva [16, 17], is one such example of an approximate occlusion culling technique. Rather than performing (expensive) conservative visibility determinations, PLP is an aggressive culling algorithm that estimates the visible primitives for a given viewpoint, and only renders those primitives that it determines to be most likely visible, up to a user-specified budget. Consequently, PLP is suitable for generating partially correct images for use in a time-critical rendering system. To illustrate this approach, consider the images of the office model shown in Fig. 1. The image generated by PLP for this viewpoint is shown in Fig. 1(a), while the correctly rendered image is in Fig. 1(b). We can see that the image rendered by PLP is fairly accurate, although portions of the model are missing, including the plant stand, clock, door jam, and parts of the desk lamp.


(a) (b)

(c) (d)

Figure 1: Office model: (a) This image was computed using PLP and is missing several triangles. (b) The correct image showing all the visible triangles rendered with cPLP. (c) The current z-buffer, rendered as luminance, for the image in (a). Black/white represents near/far objects. (d) Final z-buffer for the correct image in (b).

PLP works by initially creating a partition of the space occupied by the geometric primitives. Each cell in the partition is then assigned, during the rendering loop, a probabilistic value indicating how likely it is that the cell is visible, given the current viewpoint, view direction, and geometry in the neighboring cells. The intuitive idea behind the algorithm is that a cell containing much geometry is

*IBM T. J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598; jklosow@us.ibm.com.

†AT&T Labs-Research, 180 Park Ave., PO Box 971, Florham Park, NJ 07932; csilva@research.att.com.

[1] Visibility culling is also used in a more general context to refer to all algorithms that cull geometry based on visibility, such as back-face culling, view frustum culling, and occlusion culling.

likely to occlude the cells behind it. At each point of the algorithm, PLP maintains a priority queue, also called the *front*, which determines which cell is most likely to be visible and therefore projected next by the algorithm. As cells are projected, the geometry associated with those cells is rendered, until the algorithm runs out of time or reaches its limit of rendered primitives. At the same time, the neighboring cells of the rendered cell are inserted into the front with appropriate probabilistic values. It is by scheduling the projection of cells as they are inserted in the front that PLP is able to perform effective visibility estimation.

In [16, 17], PLP was shown to be effective at finding visible primitives for reasonably small budgets. For example, for a city model containing 500K triangles, PLP was able to find (on average) 90% of the visible triangles while rendering only 10% of the total geometry. This number alone does not guarantee the quality of the resulting images, since the missing 10% of the visible triangles could occupy a very large percentage of the screen or may be centrally located so that the incorrect pixels are very evident to the viewer. To address this concern, the authors reported the number of incorrect pixels generated by the PLP algorithm. In the worst case, for the same model and viewpoints discussed above, PLP only generated 4% of the pixels incorrectly. These two statistics support the claim that PLP is effective in finding visible geometry.

As mentioned previously, approximate occlusion culling techniques will sacrifice image accuracy for greater rendering interactivity. While this tradeoff may be acceptable in some applications (especially those that demand time-critical rendering), there are many others (such as manufacturing, medical, and scientific visualization applications) that cannot tolerate such artifacts. The users of these applications require that all of the images generated to be completely accurate. To address the requirements of these applications, we describe an efficient conservative occlusion culling algorithm based upon PLP. Essentially, our new algorithm works by filling in the holes in the image where PLP made the mistake of not rendering the complete set of visible geometry.

An interesting fact is that after rendering PLP's estimation of the visible set, as shown in Fig. 1(a), most of the z-buffer gets initialized to some non-default value, as illustrated by Fig. 1(c). This figure corresponds to the z-buffer rendered as luminance, where black represents near objects, and white represents far objects. If we were to render the cells in the front (see Fig. 3), the visible cells would protrude through the rendered geometry. The techniques we present in this paper are based on incrementally computing which cells in PLP's front are occluded (that is, can not be "seen" through the current z-buffer), and eliminating them from the front until the front is empty. When this condition holds, we know we have the correct image (1(b)) and z-buffer (1(d)).

The use of (two-dimensional) depth information to avoid rendering occluded geometry is not a new idea. The Hierarchical Z-Buffer technique of Greene et al. [14] is probably the best known example of a technique that effectively uses such information. However, even before this seminal paper, Kubota Pacific already had hardware support on their graphics subsystem for visibility queries based on the current status of the depth buffer. In Section 5, we will put our new techniques into context with respect to the relevant related work.

The main contributions of our work are:

- We propose cPLP, an efficient interactive rendering algorithm that works as an extension to the PLP algorithm by adding a second phase which uses image-space visibility queries.

- We show how to efficiently implement such image-space visibility queries using available OpenGL hardware and extensions. Our implementation techniques can potentially be used in conjunction with other algorithms.

- We discuss the performance and limitations of current graphics hardware, and we propose a simple hardware extension that could provide further performance improvements.

The remainder of our paper has been organized as follows. In Section 2, after a brief overview of PLP and some aspects of its implementation, we detail our new cPLP algorithm. We present several techniques for the implementation of our image-space visibility queries using available OpenGL hardware and extensions in Section 3. We also propose a simple hardware extension to further improve rendering performance. In Section 4 we report on the overall performance of the various techniques on several graphics architectures. In Section 5, we provide a brief overview of the previous work on occlusion culling, followed by a more thorough comparison of our current algorithm with the most relevant prior techniques. Finally, we end the presentation with some concluding remarks.

2 The Conservative PLP Algorithm

The conservative PLP algorithm (cPLP) is an extension to PLP which efficiently uses image-space visibility queries to develop a conservative occlusion culling algorithm on top of PLP's time-critical framework. In this section, we briefly review the original PLP algorithm and then present our cPLP algorithm. Our image-space visibility queries, a crucial part of the implementation of cPLP, are discussed in Section 3.

2.1 Overview of PLP

Prioritized-Layered Projection is a technique for fast rendering of high depth complexity scenes. It works by estimating the visible primitives in a scene from a given viewpoint incrementally. At the heart of the PLP algorithm is a space-traversal algorithm, which prioritizes the projection of the geometric primitives in such a way as to delay rendering primitives that have a small likelihood of being visible. Instead of explicitly overestimating the visible set of primitives, as is done in conservative techniques, the algorithm works on a budget. For each viewpoint, the viewer can provide a maximum number of primitives to be rendered and the algorithm will deliver what it considers to be the set of primitives which maximizes the image quality, based upon a visibility estimation metric. PLP consists of an efficient preprocessing step followed by a time-critical rendering algorithm as the data is being visualized.

PLP partitions the space that contains the original input geometry into convex cells. During this one-time preprocessing, the collection of cells is generated in such a way as to roughly keep a uniform density of primitives per cell. This sampling leads to large cells in unpopulated areas and small cells in densely occupied areas. Originally, the spatial partitioning used was a Delaunay Triangulation [16]; however, an octree has recently been shown in [17] to be a more effective data structure, both in terms of efficiency and ease of use. Since an octree is actually a hierarchy of spatial nodes as opposed to a disjoint partition, we only utilize the set of all leaf nodes of the octree, since these do provide such a partition.

Using the number of geometric primitives contained in a given cell, a *solidity* value ρ is defined, which represents the intrinsic occlusion that this cell will generate. During the space traversal algorithm, solidity values are accumulated by cells based upon the current viewing parameters (viewpoint and view direction), as well as the normal of the face shared by neighboring cells. Using these accumulated values, the traversal algorithm prioritizes which cells are most likely to be visible and therefore should be projected. For a complete treatment of these calculations, please refer to [16, 17].

Starting from the initial cell which contains the viewpoint, PLP attempts to carve cells out of the tessellation. It does this by always projecting the cell in the front $\mathcal{F}$ (*the front* is the collection of cells that are immediate candidates for projection) that is least likely to be occluded according to its solidity value. For each new viewpoint, the front is initially empty, and we insert the cell containing (or closest to) the viewpoint. This cell is then immediately projected (since it is the only candidate currently in the front) and as its neighboring cells are inserted into the front, their accumulated solidity values are estimated to reflect their position during the traversal. At the next iteration, the cell in the front most likely to be visible is projected, and its neighboring cells are inserted into the front with appropriate solidity values. If a cell has already been inserted into the front, its solidity values are updated accordingly. Every time a cell in the front is projected, all of the geometry assigned to it is (scheduled to be) rendered.

2.2 The cPLP Algorithm

As previously mentioned, the cPLP algorithm is built on top of PLP. The basic idea is to first run PLP to render an initial, approximate image. As a side effect of rendering this image, two further structures will be generated that we can exploit in cPLP: (i) the depth buffer corresponding to the approximate image, and (ii) PLP's priority queue (front), which corresponds to the cells of the spatial partition that would be rendered next by PLP if it had more time. In cPLP, we will iteratively use the depth buffer to effectively cull the cells in the front until all of the visible geometry has been rendered. The general idea can be summarized as follows:

(1) Run PLP using a small budget of geometric primitives.

This step generates a partially correct image with "holes" (regions of incorrect pixels), the corresponding depth buffer, and the priority queue (front) containing the cells that would be projected next.

(2) While the front is not empty, perform the following steps:

(2a) Given the current front, determine which cells are occluded, using image-space visibility queries, and remove them from the front.

(2b) Continue running PLP, so that each cell in the current front gets projected, since we know that they are all visible.

During this phase, new cells that neighbor the projected cells are inserted into the front as before, although they not candidates for projection during this iteration. We terminate this iteration after each of the original cells (i.e. those in the front after step (2a)) have been projected.

As cells are rendered in step (2b), the holes (and the depth buffer) get filled in, until the image is complete. A nice feature of cPLP is that we know we are done exactly when the front is empty.

One advantage of the formulation given above is that cPLP is able to perform several visibility queries during each iteration. At the same time, the main complication in implementing cPLP comes from the visibility queries in step (2a). This is further discussed in Section 3.

2.3 Challenges

There are primarily three obstacles that cPLP must overcome to be a conservative, interactive rendering algorithm. It must start with a good estimation of the correct image, determine which regions of the estimation are incorrect, and find the remaining visible geometry. Of course, to be truly interactive, each of the solutions to these challenges must be performed very efficiently. This can be done thanks to the way PLP was designed. We discuss each of these issues below.

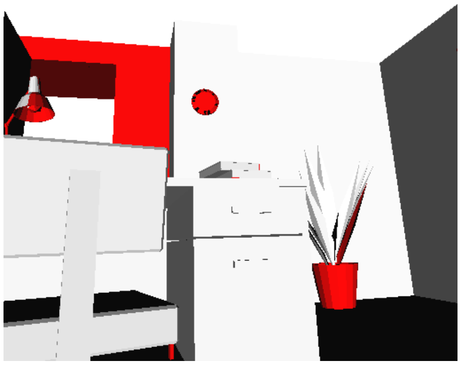


Figure 2: Illustration of the accuracy of PLP: For the same viewpoint and model as shown in Fig. 1, the visible geometry that PLP rendered is shown in white, and the visible geometry that PLP did not render is shown in red.

Estimating the image As demonstrated in [16, 17], PLP is very effective in finding the visible polygons and correctly rendering the vast majority of pixels, even when using relatively small budgets. To illustrate this point, Figs. 1(a) and 1(b) show images of an office model for the PLP and cPLP algorithms. PLP was fairly successful in finding most of the visible geometry for this viewpoint. To better visualize the accuracy of PLP, Fig. 2 highlights the visible geometry that PLP rendered in white, and the visible geometry that PLP did not render in red. By taking full advantage of the accuracy of PLP, our conservative algorithm can quickly obtain a good estimation of the correct image.

This feature can also be used to potentially speed-up other occlusion culling techniques (such as those in [25, 33]), which rely on using the z-buffer values to cull geometry.

Finding the holes As PLP projects cells (and renders the geometry inside these cells), it maintains the collection of cells that are immediate candidates for projection in a priority queue, called the front. Clearly, as the primitives in the scene are rendered, parts of the front get occluded by the rendered geometry. In Fig. 3, we illustrate this exact effect. If no "green" (the color that we used for the front) were present, the image would be correct. In general, the image will be completed, and rendering can be stopped, after all of the cells in the front are occluded by the rendered primitives. Thus, to find the holes in the estimated image, we need only consider the cells in the front.

Filling the holes The final piece that we need to build cPLP is how to complete the rendering once we know what parts of the front are still visible. For this, it is easier to first consider the current occluded part of the front. Basically, we can think of the occluded front as a single occluder (see Fig. 4) that has a few holes (corresponding to the green patches in Fig. 3). Thinking analogously to the work of Luebke and Georges [19], the holes can be thought of as "portals", or reduced viewing frusta, through which all of the remaining visible geometry can be seen. An equivalent formulation is

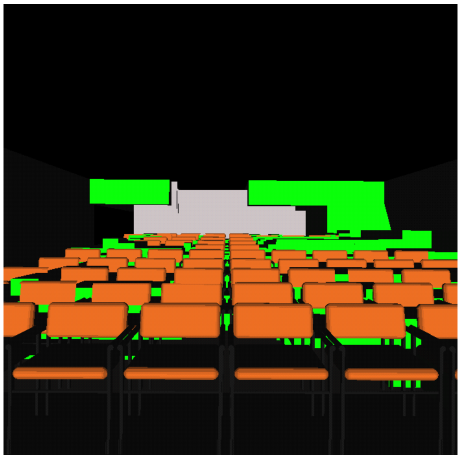


Figure 3: The current front is highlighted in green. By determining where the front is still visible, it is possible to localize the remaining work to be done by our rendering algorithm.

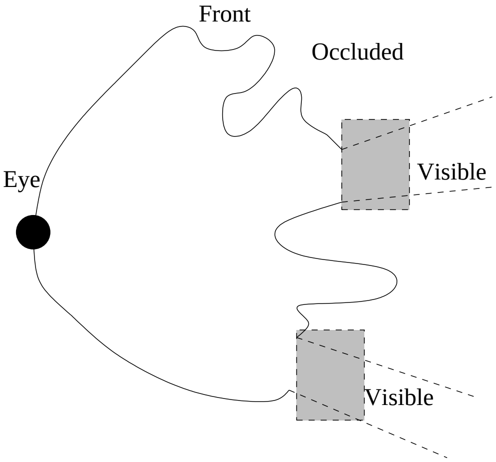


Figure 4: This figure illustrates the technique used in finding the remaining visible cells in cPLP. These cells are found by limiting the remaining work done by the algorithm to only the visible regions.

to incrementally determine what cells belong to these smaller view frusta by using an efficient visibility query (discussed below).

3 Implementing Visibility Queries

As previously discussed, to extend PLP into a conservative algorithm, we need to efficiently determine which cells in the front are visible. The visibility queries will take place in image-space and will utilize the current depth buffer. In this section, we first describe three techniques for implementing these queries using available OpenGL hardware and extensions. These include using a hardware feature available on some graphics architectures (such as some Hewlett-Packard (HP) and Silicon Graphics (SGI) graphics adapters), an item-buffer technique that requires only the capability of reading back the color buffer, and an alternative approach that uses an extension of OpenGL 1.2. Then, we discuss some further optimization techniques. Finally, we end this section by proposing a new hardware extension that has the potential to speed up visibility queries even further.

3.1 Counting Fragments After Depth Test

One technique for performing the visibility queries of cPLP is to use the HP occlusion culling extension, which is implemented in their fx series of graphics accelerators. This proprietary feature, which actually seems quite similar to the capabilities of the Kubota Pacific Titan 3000 reported by Greene et al. [14], makes it possible to determine the visibility of objects as compared to the current values in the z-buffer. The idea is to add a feedback loop in the hardware which is able to check if changes *would* have been made to the z-buffer when scan-converting geometric primitives. The actual hardware feature as implemented on the fx series graphics accelerators is explained in further detail in [25, 26]. Though not well-known, several other vendors provide the same functionality. Basically, by simply adding instrumentation capabilities to the hardware which are able to count the fragments which pass the depth test, any architecture can be efficiently augmented with such occlusion culling capabilities. This is the case for the SGI Visual Workstation series which have defined an extension called `GL_SGIX_depth_pass_instrument` [27, pages 72–75]. Several new graphics boards, such as the SGI InfiniteReality 3 and the Diamond FireGL have such functionality. Even low-cost PC cards such as the 3Dfx Voodoo graphics boards have had similar functionality in their Glide library (basically by supporting queries into the hardware registers). Since the functionality proposed by the different vendors is similar, in the rest of this paper, we concentrate on the HP implementation of such occlusion culling tests.

One possible use of this hardware feature is to avoid rendering a very complex object by first checking if it is potentially visible. This can be done by checking whether a bounding volume *bv*, usually the bounding box of the object, is visible and only rendering the actual object if *bv* is visible. This can be done using the following fragment of C++ code:

```
glEnable(GL_OCCLUSION_TEST_HP);
glDepthMask(GL_FALSE);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
DrawBoundingBoxOfObject();
bool isVisible;
glGetBooleanv(GL_OCCLUSION_RESULT_HP, &isVisible);
glDisable(GL_OCCLUSION_TEST_HP);
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
if (isVisible)
  DrawGeometryofObject();
```

This capability is exactly what is required by our cPLP visibility queries. Given the current z-buffer, we need to determine what cells in the front are visible. It is a simple task to use the HP hardware to query the visibility status of each cell.

The HP occlusion culling feature is implemented in several of their graphics accelerators, for example, the fx6 boards. Although performing our visibility queries using the HP hardware is very easy, the HP occlusion culling test is not cheap. In an HP white paper [26], it is estimated that performing an occlusion query with a bounding box of an object on the fx6 is equivalent to rendering about 190 25-pixel triangles. Our own experiments on an HP Kayak with an fx6 estimates the cost of each query being higher. Depending upon the size of the bounding box, it could require anywhere between 0.1 milliseconds (ms) to 1 ms. This indicates that a naive approach to visibility culling, where objects are constantly checked for being occluded, might actually hurt performance, and not achieve the full potential of the graphics board. In fact, it is possible to slow down the fx6 considerably if one is unlucky enough to project the polygons in a back-to-front order, since none of the bounding boxes would be occluded. In their most recent offerings, HP has improved their occlusion culling features. The fx5 and fx10 accelerators can perform several occlusion culling queries in parallel [9]. Also, HP reports that their OpenGL implementation has been changed to use the occlusion culling features automatically whenever feasible. For example, prior to rendering a large display list, their software would actually perform an occlusion query before rendering all of the geometry.

Utilizing the HP occlusion culling feature has proven to be the simplest and most efficient of our three techniques for performing the visibility queries needed by cPLP. Unfortunately, at this time, this hardware feature is not widely available in other graphics boards (for instance, neither of market leaders Nvidia or ATI support this feature). Because of this, we next describe a simple item-buffer technique, whose only requirement is the capability to read back the color buffer. In Section 3.6, we propose a simple extension of the OpenGL functionality which extends the fragment-counting idea, by adding components of the techniques described next.

3.2 An Item Buffer Technique

It is possible to implement visibility queries similar to the ones provided by the HP occlusion test on generic OpenGL hardware. The basic idea is to use the color buffer to determine the visibility of geometric primitives. For example, if one would like to determine if a given primitive is visible, one could clear the color buffer, disable changes to the z-buffer (but not the actual z test), and then render the (bounding box of the) primitive with a well-known color. If that color appears during a scan of the color buffer, we know that some portion of the primitive passed the z test, which means the (bounding box of the) primitive is actually visible.

There are two main costs associated with the item-buffer technique: transferring the color buffer from the graphics adapter to the host computer's main memory and the time it takes the CPU to scan the color buffer. The transfer cost can be substantial in comparison to the scanning cost (see Table 2). Consequently, it is much more efficient to do many visibility queries at once. By coloring each of the cells in the front with a different color, it is possible to perform many queries at the same time.

An unwanted side effect of checking multiple cells is that a cell, *C*, in the front may be occluded by other cells in the front, as opposed to the current z-buffer which contains depth information for the *previously rendered geometry*. This is a problem because although cell *C* is occluded by the other cells in the front, the geometry contained within cell *C* may not be occluded by the geometry within the other cells. A multi-pass algorithm is therefore required to guarantee that a cell is properly marked as occluded. Initially, all cells in the front are marked as "potentially visible". We also disable writing to the z-buffer, so that it remains accurate with respect to the geometry previously rendered by PLP. To retain the color buffer information for this geometry, we save the initial image generated by PLP during step (1) (see Sections 2.2 and 3.5). Each pass of the algorithm then clears the color buffer and renders the boundary of each of the cells in the front that is potentially visible using a distinct color. We then transfer and scan the color buffer to determine which cells are actually visible and mark them. Iterating in this fashion, we can determine exactly which cells are visible with respect to the previously rendered geometry. The remaining cells are determined to be occluded by the previously rendered geometry and need not be considered further. The multi-pass algorithm terminates once the color buffer scan indicates that none of the rendered cells, for the current pass, were determined to be visible. That is, the color buffer is completely empty of all colors. Note that potentially visible cells will need to be rendered multiple times, however, once a cell is found to be visible in one pass, it is marked appropriately and not rendered again. Pseudo-code for the item-buffer technique is included below.

```
glDepthMask(GL_FALSE);
for each cell c in front {
  markCellPotentiallyVisible(c);
}
bool done = false;
while (!done) {
  glClear(GL_COLOR_BUFFER_BIT);
  for each cell c in front {
    if (potentiallyVisible(c))
      renderCell(c);
  }
  glReadPixels(0, 0, width, height,
    GL_RGBA, GL_UNSIGNED_BYTE, visible_colors);
  int cnt = 0;
  for each cell c that appears in visible_colors {
    markCellVisible(c);
    cnt++;
  }
  if (cnt == 0)
    done = true;
}
```

3.3 The OpenGL Histogram Extension

The item-buffer technique just proposed performs a lot of data movement between the graphics accelerator's memory and the host computer's main memory. On most architectures, this is still a very expensive operation, since the data must flow through some shared bus with all of the other components in the computer. We propose a different technique which uses intrinsic OpenGL operations to perform all the computations on the graphics accelerators, and only move a very small amount of data back to the host CPU.

Our new technique shares some similarity to the previous item-buffer technique. For instance, it also needs to render the potentially visible cells multiple times, until no visible cell is found. However, the new method uses OpenGL's histogramming facility, available in the ARB_imaging extension of OpenGL 1.2, to actually compute the visible cells (see [1]). After rendering the potentially visible cells in this case, rather than transferring the color buffer to the host's CPU and scanning it for the visible cells, we simply enable the histogramming facility and transfer the color buffer into texture memory (still on the graphics accelerator). During this transfer, OpenGL will compute the number of times a particular color appears. A short array with the accumulated values can then be fetched by the host CPU with a single call. A fragment of our C++ code illustrates this approach.

```
glEnable(GL_TEXTURE_2D);
glEnable(GL_HISTOGRAM_EXT);
glHistogramEXT(GL_HISTOGRAM_EXT,  256,
    GL_LUMINANCE, GL_TRUE );
glCopyTexSubImage2D(GL_TEXTURE_2D, 0,
    0, 0, WIDTH, HEIGHT, WIDTH, HEIGHT);
GLuint histogram_values[256];
glGetHistogramEXT(GL_HISTOGRAM_EXT,  GL_FALSE,
    GL_LUMINANCE, GL_UNSIGNED_INT,
    histogram_values);
glResetHistogramEXT ( GL_HISTOGRAM_EXT );
glDisable(GL_TEXTURE_2D);
glDisable(GL_HISTOGRAM_EXT);
```

After this code is executed, the array histogram_values contains the number of times each color (here uniquely identified by an integer between 0 to 255) appeared. With this technique, the graphics board does all the work, and only transfers the results to the host CPU. The same termination criterion exists for this multi-pass algorithm as for the item-buffer technique, although we can more easily test for this condition in this case. For instance, if histogram_values[0] is equal to WIDTH × HEIGHT, meaning all pixels are the same (background) color, then no cells are visible and we terminate the algorithm.

3.4 Improving Visibility Query Performance

It is possible to improve the performance of our visibility query techniques by implementing several optimizations. The previous two techniques need to perform operations that touch all the pixels in the image, possibly multiple times. To avoid computations in areas of the screen that have already been completely covered, we have implemented a simple tiling scheme that greatly reduces the amount of transfers and scans required. The basic idea is to simply divide the screen into fixed tiles. For a 512x512 pixel image, we could break the screen up into 64 tiles, each containing a block of 64x64 pixels. During the multi-pass algorithm, we need to keep track of the active tiles, those that in the previous iteration contained visible primitives. After each iteration, tiles get completed, and the number of tiles which need to be rendered to and scanned decreases.

Another simple optimization for the item-buffer technique was to minimize the number of color channels to transfer to the host computer's main memory. For example, if we have r bits to represent the red color component on our machine, and we have fewer than 2^r cells to check in the front, we can uniquely color these cells using only the red color component. Consequently, we would only need to transfer and scan the GL_RED component for each pixel in the image, as opposed to transferring and scanning the entire GL_RGBA component.

We have implemented and are currently using these two optimizations. A non-conservative optimization for our techniques would be to compute visibility in a lower resolution than the actual rendering window [33]. Although a quite effective optimization, this might lead to undesirable artifacts. This is one of the reasons we do not use it in our system.

3.5 Integration with cPLP

The techniques presented so far essentially solve step (2a) of cPLP. Both the item-buffer technique as well as the histogramming technique need to have access to the color buffer of the machine being used for its computations. For each pass, they require that the color buffer be cleared, which conflicts with the image computation which is performed in steps (1) and (2b). Naively, it would be necessary to save the complete color buffer (or at least the active tiles) before each call to step (2a) and restore it before the call to step (2b).

Instead, since we expect that after step (1) most of the visible triangles have been rendered, we simply save the image step (1) generated, and ignore the changes to the color buffer from then on (we re-rendered the extra geometry in the end to recover the correct image). The important thing is to correctly account for the z-buffer changes that are triggered by the rendering of the geometry inside the cells. To do this, before step (2b), we change the masks on the z-buffer so that it gets updated as geometry is rendered in (2b). When the front becomes empty, we know the z-buffer was completed. At that point, we perform a single image restore (with the image we saved in step (1)), and we re-render all the geometry that was found to be visible since that point.

Fig. 10 provides an overview of our cPLP algorithm as described. For a sample view of an office model, snapshots were taken at several iterations (step 2) of our algorithm. Figs. 10(a)-(c) illustrate the current color buffer and front (in blue) at each iteration. The remaining visible geometry will come from within the visible front cells. (d)-(f) illustrate the tiles of the screen that have been completed and therefore do not need to be scanned during subsequent iterations. (c) and (f) correspond to the final (correct) image, since all of the tiles have been completely covered. Note that in (b), the front cells, which are barely visible, are in the upper left corner and near the two desks in the middle of the screen. As expected, the tiles that represent these areas are not marked as completed.

3.6 Extending the OpenGL Histogram

Here we propose a modification to OpenGL that has the potential to greatly improve performance. In particular, it would make it possible to avoid the costly multi-pass visibility computations that we are currently forced to use, and it can be seen as a generalization of the HP occlusion culling test.

OpenGL background Before we go into details, it helps to understand a bit more on how OpenGL works. The graphics pipeline is the term used for the path a particular primitive takes in the graphics hardware from the time the user defines it in 3D to the time it actually contributes to the color of a particular pixel on the screen. At a very high level, a primitive must undergo several operations before it is drawn on the screen. A flowchart of the operations is shown in Fig. 5.

The user has several options for specifying vertices that are grouped into primitives, e.g., triangles or quads. Primitives go through several stages (not shown), and eventually, get to the rasterization phase. It is at rasterization that the colors and other properties of each pixel are computed. During rasterization, primitives get broken into what we usually refer to as "fragments". Modern graphics architectures have several per-fragment operations that can be performed on each fragment as they are generated. As fragments are computed, they are further processed, and the hardware incrementally fills the framebuffer with an image.

Per-Fragment Histogramming The OpenGL histogramming facility, part of the pixel transfer operations shown in Fig. 5, operates on images, which can potentially come from the framebuffer. The OpenGL histogram works by counting the number of times a color appears in a given image.

The reason we need to perform multiple passes to determine when cells are visible at this time is that we are using the color buffer to find which of the primitives passed the z-test. With the standard pipeline, we only get the "top layer" of visible cells, since one of the per-fragment operations that occurs before a pixel is written to the color buffer is the depth-test. If a per-fragment histogramming facility is added to the pipeline and it could be used to perform the same exact operation on *fragments* (which pass the z-test), it would be possible to count how many fragments of a given primitive passed the z-test. If this number is zero, the primitive would be occluded, otherwise, the histogram value would not only tell us that it is visible, but actually provide an upper bound on the number of its pixels that are visible. With the proposed change in the OpenGL pipeline, we would still be able to perform several queries at the same time, but we would not be required to perform multiple passes over the framebuffer.

The per-fragment histogramming functionality we are proposing is a clean way to extend the (already useful) techniques based on counting the number of fragments which pass the z-test (such as the HP occlusion culling test), so that it is able to handle multiple and more general tests with better performance. We would like to point out that the hardware cost (in component cost or chip area) would likely be non-trivial, since high-performance graphics hardware is highly parallel (for instance, Nvidia's GeForce can compute four fragments simultaneously), and the extra hardware for the per-fragment histogramming would have to be replicated for each fragment generator. Of course, this is already the case for several other extensions, including the existing fragment counting hardware. We believe the actual cost (in time) of our augmented test would be similar to the cost of a single HP test, while we would be able to perform several tests concurrently.

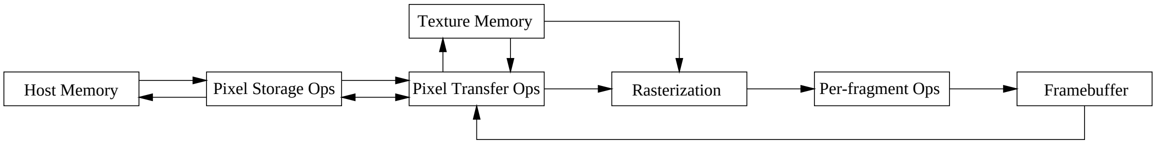


Figure 5: OpenGL imaging pipeline

Machine	CPU(s)	Graphics	RAM
SGI Octane	1 X R12000, 300MHz	MXE	512MB
SGI Onyx	12 X R10000, 195MHz	Infinite Reality	2GB
HP Kayak	2 X Pentium II, 450MHz	fx6	384MB

Table 1: The configurations of the machines used in our experiments. The number of processors *P* per machine is listed in the CPU(s) column, in the form: P X cpu-type, cpu-speed.

4 Experimental Results

We performed a series of experiments to determine the effectiveness of our new cPLP algorithm. We report results for each of the three implementations of our visibility queries presented in Section 3, as well as several alternatives for benchmarking:

cPLP-HP: cPLP, using the HP occlusion culling extension,

cPLP-IB: cPLP, using the item-buffer technique,

cPLP-HG: cPLP, using the OpenGL histogram extension,

cPLP-EXT: cPLP, using our hardware extension proposed in Section 3.6,

PLP: the original PLP,

VF-BF: view frustum and back-face culling only,

HP: using the HP hardware to perform the visibility queries without the benefit of running PLP to preload the color and depth buffers.

Test model The primary model that we report results on is shown in Fig. 9(a) and consists of three copies, placed side by side, of the third floor of the Berkeley SODA Hall. Arranging the copies in such a way helps us better understand how the different occlusion culling techniques function in a high depth complexity environment, since they have their greatest opportunity where there is significant occlusion. Each room in the model has various pieces of furniture and in total, the three replicas contain over one million triangles.

We generated a 500-frame path that travels right-to-left, starting from the upper right corner of Fig. 9(a). In Fig. 9(b)–(e), we show a few representative frames of the path. The number of visible polygons in each frame varies considerably, especially when moving from room to room.

Machine architectures Our experiments were performed on a three different architectures: an SGI Octane, an SGI Onyx, and an HP Kayak. The configurations of the machines are listed in Table 1.

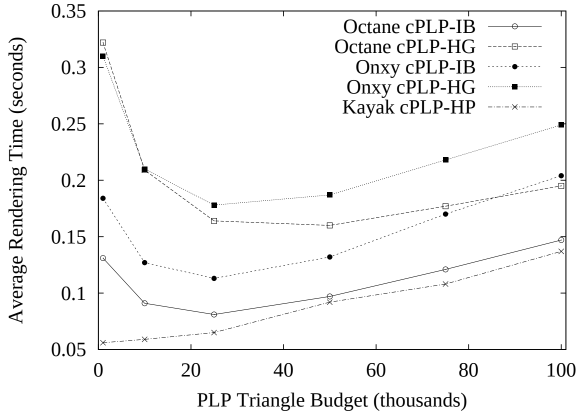


Figure 6: Average rendering times per frame for the implementations of the cPLP algorithm. The PLP budget, reported in thousands of triangles, determines the number of triangles initially rendered to fill-in the depth buffer.

Preprocessing As discussed in Section 2, the preprocessing step of cPLP, which is identical to the preprocessing step of the original PLP algorithm, is very efficient. The preprocessing includes reading the input geometry from a file, building the octree, determining which geometry each cell contains, and computing the initial solidity values. The total preprocessing times for the one million triangle model mentioned above was 76 seconds, 128 seconds, and 90 seconds, for the Octane, Onyx, and Kayak, respectively. While these times are actually quite modest, we have an additional opportunity to reduce the preprocessing requirement. For portability purposes, we are currently using an ASCII format to store the model. For each of the three machines being used, at least half (42, 64, and 56 seconds, respectively) of the preprocessing time listed above was spent simply reading in the model. If we were to store the model in a compact binary format, the input portion of the preprocessing would likely be reduced considerably. The octree construction, geometry assignment, and initial solidity computation only required 34, 64, and 34 seconds, respectively, on each of the three machines, and could likely be reduced by carefully optimizing our code. For the experiments reported here, we subdivided the octree until each leaf contained fewer than 5000 triangles. This resulted in 1429 octree leaf cells being created.

Rendering results We present our main rendering results for the various cPLP implementations in Fig. 6. The vertical axis represents the average rendering time for each of the 500 steps in the path generated for the test model. The horizontal axis represents the initial budget used by PLP to render what it determined to be the most likely visible geometry, thereby preloading the color and depth buffers.

If we compare the item-buffer and histogram techniques, we see that the item-buffer is considerably faster on each of the SGI machines. All of these runs[2] tended to reach their minimum values for an initial PLP budget of 25K triangles, or roughly 2.5% of the total number in the model. For this budget, the rendering times for the item-buffer technique on the Octane and Onyx were 0.081 and 0.113 seconds on average per frame. This is equivalent to rendering 12.35 and 8.85 frames per second, respectively. In comparison, the histogram approach took 0.164 and 0.178 seconds on average per frame, or the equivalent of 6.10 and 5.62 rendered frames per second.

We did not run cPLP-HG on the Kayak since the OpenGL histogram extension is not available on that machine. Also, the cPLP-IB technique on the Kayak was very slow, requiring 0.864 seconds on average per frame. We explain why this is the case when we discuss the costs of the primitive operations for each of the techniques below. The HP hardware occlusion culling extension was clearly not available on the SGIs, and so we can only report on this technique on the Kayak.

cPLP-HP was the most efficient algorithm but we were a little surprised by the fact that it increased in running time as we increased the PLP budget. We anticipated that we would see a parabolic curve similar to the runs on the two SGI machines. Initially, we considered that running PLP followed by our cPLP-HP visibility queries was not benefitting us at all on the Kayak. To test this hypothesis, we implemented another technique, HP, that used the hardware occlusion culling extension without the benefit of running PLP first to preload the depth buffer. Given the set of leaves in our octree, we first discarded those nodes that were outside the view frustum, and then sorted the remaining nodes according to their distance from the viewpoint. We then performed visibility queries for the nodes in this order. On average, the HP technique required 0.157 seconds per frame, which is considerably slower than our cPLP-HP algorithm.

While sorting the nodes according to distance appeared to be a good technique, it clearly cannot capture any occlusion information as did cPLP. In addition, this HP technique does not have a mechanism for determining which nodes are still visible and which sections of the screen are yet incomplete. Consequently, this method cannot easily determine when it is finished, and therefore must perform many more visibility queries than the cPLP-HP technique. One could think of modifying this HP approach so that the queries are performed in a hierarchical fashion since we have the octree constructed anyway. However, while in some cases this could reduce the overall rendering time, in many others the times will increase due to the increase in the number of visibility queries. We shall discuss shortly the times required for the HP visibility queries. Thus, although the benefit gained from PLP was not exactly as we anticipated, it still plays a crucial role in achieving interactive rendering times.

To quantify how well our conservative culling algorithm is working, we implemented a simple rendering algorithm, VF-BF, that performed only view frustum and back-face culling. These traditional culling approaches were also used within cPLP. The VF-BF algorithm is considerably slower than all of the cPLP implementations. For example, on the Octane, VF-BF took 0.975 seconds to render each frame on average. Thus, our cPLP-IB and cPLP-HG methods render frames 12 and 6 times faster than the VF-BF technique. Our cPLP-HP method provides even better comparisons. Such improvements in rendering speeds, which were similar on all of the architectures, are crucial for any application requiring interactivity.

Of the time spent by our cPLP approaches, a good portion of that time was actually spent running the initial PLP algorithm. For example, on the Octane, out of the 0.081 seconds it takes to render a frame on average, 0.064 seconds were occupied by the initial PLP algorithm, and 0.017 seconds used by the iterative visibility queries to complete the rendered image. For the item-buffer and histogram techniques, the average number of iterative visibility queries per frame ranged from 4.7 iterations, for an initial PLP budget of only 1000 triangles, to 1.5 iterations, for an initial budget of 100000 triangles.

Machine	SGI Octane		SGI Onyx		HP Kayak	
Image Size	64^2	512^2	64^2	512^2	64^2	512^2
Transfer	217	4483	564	7733	375	11250
Scan	30	2300	20	1000	47	3430
Total	247	6783	584	8733	422	14680

Table 2: Times for the primitive operations of the item-buffer technique. An image size of 64^2 refers to an image that is 64x64 pixels in size. The transfer time is the dominant cost of this method. All times are reported in microseconds.

Primitive Operation Costs To better understand the rendering times reported in Fig. 6, we analyzed the cost of performing the underlying primitive operations for each of the methods. By looking at these results, we can offer additional insight into why each of the methods works as well, or as poorly, as it does.

For the cPLP-HP technique, the visibility queries involve enabling the HP culling extension, rendering a cell, and reading back the flag to indicate whether the z-buffer would have changed if we had actually rendered the cell. We timed the visibility queries on the HP Kayak and found that the time ranged between 100 microseconds (μs) and 1000μs. In addition to these costs, the HP visibility query can also interrupt the rendering pipeline, thereby reducing the overall throughput. Consequently, it is imperative when using these queries to do so with some caution. It is especially advantageous when you are very likely to find significant occlusion. Otherwise, many queries may be wasted and the overall rendering performance will be reduced.

The primitive operation for the item-buffer technique is the transferring of the color buffer from the graphics accelerators memory to the main memory of the host computer. This is done in OpenGL using a single call to glReadPixels. The other main cost associated with this technique is the time it takes the CPU to scan the color buffer to determine which cells have actually contributed to the image. We report these numbers for each of our machines in Table 2. It is immediately apparent why the cPLP-IB technique on the Kayak is so slow. The transfer and scan times are considerably slower (for the 512x512 image) than on the SGIs. Another interesting observation, which also helps justify our tiling optimization in Section 3.4, is the substantial increase in time that is required to transfer and scan a 512x512 pixel image, as opposed to only a 64x64 pixel (sub)image.

For those machines that support the OpenGL histogram extension, the underlying operations include copying an image, or subimage in the case of our tiles, from the framebuffer to texture memory. We have timed this operation with the histogram extension enabled to see how much time is required for the copy with the histogram calculations. The histogram calculation also includes the time to retrieve and scan the histogram results. On the Octane it takes 800μs for a 64x64 pixel image, and 34000μs for a 512x512 image. On the Onyx, it takes 690μs for a 64x64 pixel image, and 13500μs for a 512x512 image. (We should note that it is quite difficult to perform such measurements, but we have done our best to report accurate results.) We were surprised by the amount of time required to copy the image to texture memory and perform the histogram computations. Our initial belief was that by using the actual hardware to perform our visibility queries, our render-

[2] The only exception being the Octane cPLP-HG method, which reached a minimum at a PLP budget of 50K triangles.

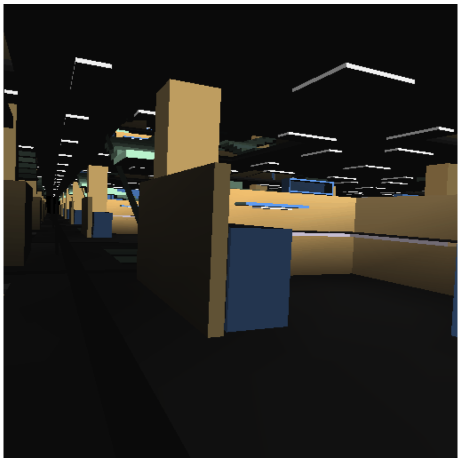


Figure 7: Interior view of a skyscraper model. cPLP reduced the depth complexity of this rendered image from 26 to 8.

ing times would decrease. Unfortunately, this is not the case at this point in time. While the Onyx appears to be more advanced than the (newer) Octane in its histogramming features, neither machine performs well enough to be faster than the item-buffer techniques.

Depth Complexity To further test our cPLP algorithms, we considered another model with extremely high depth complexity. Fig. 7 shows an interior view of a skyscraper model which consists of over one million triangles. The model, courtesy of Ned Greene, consists of 54 copies of a module, each with almost 20K triangles.

The purpose of this experiment was to determine the depth complexity of this model when rendering it using the various techniques. By depth complexity, we refer here to the average number of times a z-test is performed for each pixel in the image. If our cPLP techniques are effective at determining occlusion, our methods should reduce the depth complexity considerably in comparison to a standard rendering algorithm. Using one such technique, VF-BF, we determined the depth complexity of this model (for this viewpoint) to be 26.70 on average, for all of the pixels in the image. Using cPLP, we were able to reduce this value to only 7.97. We emphasize that these numbers refer to the number of z-tests per pixel, as opposed to the number of z-tests that pass (i.e., resulting in the pixel's color being overwritten by a fragment that is closer to the viewer), which has been reported in other approaches. We opted for this number since the number of z-tests more accurately reflect the work that is done during the rendering algorithm.

cPLP-EXT Since we do not actually have hardware which implements our proposed extension, here we extrapolate on its performance based on the results we have, assuming we were to add such an extension to the HP Kayak fx6. Using cPLP-IB, it is possible to determine the number of tests that can be performed in parallel for each triangle budget in Fig. 6. Assuming our extension is properly implemented, we believe it should take no more time than the fragment counting technique already available on several architectures. While measuring on HP machines, we found that in the worst case, an occlusion test costs 1 ms. But since we have to bring more data from the graphics hardware for our extension, we will assume that each query is twice as expensive, or 2 ms, to account for the extra data transfer. (Since only extremely small arrays of 256 values are being transfered, we don't actually believe it would have such an impact.)

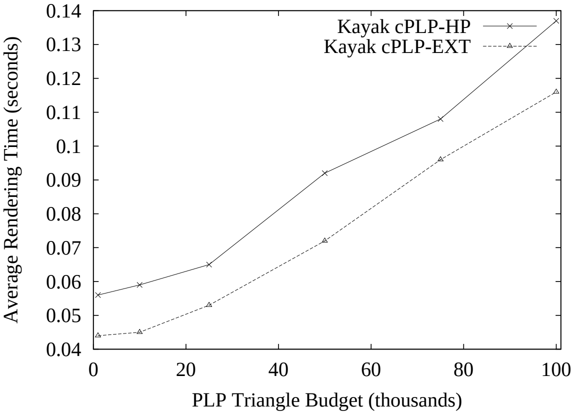


Figure 8: Average rendering times per frame for cPLP-HP and our proposed hardware extension method cPLP-EXT. The PLP budget, reported in thousands of triangles, determines the number of triangles initially rendered to fill-in the depth buffer.

Table 3 summarizes our findings. Basically, we are computing the time for cPLP-EXT as a sum of the initial PLP cost (initialize its per-frame data structures, such as zeroing the solidity of each cell; and rendering the first batch of triangles for all frames), plus the total number of parallel EXT tests (which we assume take 2 ms each), plus the time to rendering the extra triangles (at a rate of approximately 1 million triangles/sec) which are found as visibility tests are performed.

With these assumptions, we can see that our frame rates get considerably better (see Fig. 8), and we could potentially achieve a frame rate of 23 Hz (versus 18 Hz for cPLP-HP; an improvement of 28%) if we had a hardware implementation of our extension. We would like to point out that the advantage would be even greater if the cost of initializing PLP's per-frame data structures was made lower. Our current PLP implementation uses an STL set, which is not particularly optimized for linear traversals which are necessary during initialization. If necessary, it would be possible to optimize this code further.

5 Related Work

There has been a substantial amount of recent work on occlusion culling (see, for instance, [5, 6, 8, 11, 18, 23, 24, 30, 31]). The purpose of this section is not to do an extensive review of all occlusion culling algorithms. For that, we refer the interested reader to the recent surveys by Cohen-Or et al. [7] and Durand [10]. Instead, we focus on reviewing work that is more closely related to our own, so that we can indicate the similarities and differences with our current work.

Closely related to our work are techniques that use two-dimensional depth information to avoid rendering occluded geometry. An early example of this is a technique by Meagher [20] that stores the scene in an octree, and the framebuffer in a quadtree. Meagher renders the octree in a strict front-to-back order, while keeping track of which parts of the quadtree get filled, in order to avoid touching parts of the octree that can not be seen. Naylor [22] proposes another version of this idea, where instead of using an octree and a quadtree, he uses two binary-space partitioning trees [12], one in 3D, the other in 2D, to efficiently keep both the scene and the image respectively. The 3D BSP can be used to traverse the scene in a strict front-to-back order, and the 2D BSP is used to keep the areas of the screen which get filled. Our current approaches differ

PLP Budget (triangles)	PLP Time (s)	# EXT Tests	Avg. Extra Triangles	Average Time (s)	Frame Rate (Hz)
1,000	0.019	4.688	16844	0.044	22.7
10,000	0.028	3.376	10978	0.045	22.2
25,000	0.043	2.426	5641	0.053	18.9
50,000	0.066	1.908	2796	0.072	13.9
75,000	0.091	1.630	1770	0.096	10.4
100,000	0.112	1.372	1247	0.116	8.6

Table 3: Performance of cPLP-EXT on a "hypothetical" HP Kayak fx6. All times are reported in seconds. The average extra triangles are the number of triangles that get rendered in addition to the PLP budget. See text for further details.

from these methods in that they do not render in a strict front-to-back order (which was shown to be less effective), but rather allow PLP to determine the order in which to visit (and render) the cells.

The Hierarchical Z-Buffer (HZB) technique of Greene et al. [14] is probably the best known example of a technique that efficiently uses depth information for occlusion culling. Their technique is related to Meagher [20] in that it also uses an octree for managing the scene, which is rendered in front-to-back order. Another similarity is that they also use a quadtree, but not for the actual framebuffer (as in [20]). Instead, they use the quadtree to store the z-buffer values, which allow for fast rejection of occluded geometry. The HZB technique also explores temporal coherency by initializing the depth buffer with the contents of the visible geometry in the previous frame.

The Hierarchical Z-Buffer has several similarities to cPLP. Their use of the visible geometry from the previous frame for the purpose of estimating the visible geometry is similar to our approach, although in our case, we use the visibility estimation properties of PLP to estimate the current frame. One advantage of doing it this way is that (as we have shown earlier) the front intrinsically tells us where to continue rendering to fill-up the z-buffer. HZB has no such information; it renders the remaining geometry in front-to-back order. The fact that we employ a spatial partitioning instead of a hierarchy in object-space is only a minor difference. Depending upon the scene properties, this may or may not be an advantage. The flat data structure we use seems more efficient for a hardware implementation, since we do not need to stop the pipeline as often to determine the visibility of objects. In [13], Greene introduces an optimized variation of the HZB technique, including a non-conservative mode.

A closely related technique is the Hierarchical Occlusion Maps of Zhang et al. [33]. For each frame, objects from a precomputed database are chosen to be occluders, and are rendered (possibly) in lower resolution to get a coverage footprint of the potential occluders. Using this image, OpenGL's texture mapping functionality generates a hierarchy of image-space occlusion maps, which are then used to determine the possible occlusion of objects in the scene. Note that in this technique, the depth component is considered after it is determined that an object can potentially be occluded. One of the main differences between HOM and cPLP is that HOM relies on preprocessing the input to find its occluders, while cPLP uses PLP for that purpose. HOM also utilizes a strict front-to-back traversal of the object-space hierarchy.

The work by Bartz et al. [3, 4] addresses several of the same questions we do in this paper. They provide an efficient technique for implementing occlusion culling using core OpenGL functionality, and then propose a hardware extension which has the potential to improve performance. Similar to the previous methods, Bartz et al. use a hierarchy for the 3D scene. In order to determine the visible nodes, they first perform view-frustum culling, which is optimized by using the OpenGL selection mode capabilities. For the actual occlusion tests, which are performed top-down in the hierarchy nodes, they propose to use a *virtual occlusion buffer*, which is implemented using the stencil buffer to save the results of when a given fragment has passed the z-test. In their technique, they need to scan the stencil buffer to perform each visibility test. Since this has to be performed several times when determining the visible nodes of a hierarchy, this is the most time consuming part of their technique, and they propose an optimization based on sampling the virtual occlusion buffer (thus making the results only approximate). In their paper, they also propose an extension of the HP occlusion culling test [25] (see [3] for details). At this time, the HP occlusion test simply tells whether a primitive is visible or not. Bartz et al. propose an extension to include more detail, such as number of visible pixels, closest z-value, minimal-screen space bounding box, etc. There are several differences between their work and our own. First and foremost, our techniques are designed to exploit multiple occlusion queries at one time, which tend to generate a smaller number of pipeline stalls in the hardware. Also, our hardware extension is more conservative in its core functionality, but has the extra feature that it would support multiple queries. One additional difference is that, similar to Greene et al. [14], cPLP incorporates an effective technique for filling up the depth buffer so as to minimize the number of queries. We do not believe that it would be difficult to incorporate this feature within the framework of Bartz et al.

The technique by Luebke and Georges [19] describe a screen-based technique for exploiting "visibility portals", that is, regions between cells which can potentially limit visibility from one region of space to another. Their technique can be seen as a dynamic way to compute information similar to that in [28]. One can think of cPLP's obscured front as a single occluder, which has a few holes. If we think of the holes as "portals", this is in certain respects analogous to the work of Luebke and Georges. In the context of their colonoscopy work, Hong et al. [15] propose a technique which merges Luebke and Georges's portals with a depth-buffer based technique similar to ours. However, in their work, they exploit the special properties of the colon being a tube-like structure.

HyperZ [21] is an interesting hardware feature that has been implemented by ATI. HyperZ has three different optimizations that improve the performance of 3D applications. The main thrust of the optimizations is to lower the memory bandwidth required for updating the z-buffer, which they report is the single largest user of bandwidth on their graphics cards. One optimization is a technique for lossless compression of z-values. Another is a fast z-buffer clear, which performs a lazy clear of the depth values. ATI also reports on an implementation of the hierarchical z-buffer in hardware. Details on the actual features are only sketchy and ATI has not yet exposed any of the functionality of their hardware to applications. Consequently, it is not possible at this point to exploit their hardware functionality for occlusion culling.

Another recent technique related to the hierarchical Z-buffer is described by Xie and Shantz [32]. They propose the Adaptive Hierarchical Visibility (AHV) algorithm as a simplification of HZB for tile architectures.

Alonso and Holzschuch [2] propose a technique which exploits the graphics hardware for speeding up visibility queries in the context of global illumination techniques. Their technique is similar to our item-buffer technique. Westermann et al. [29] propose a different technique for using the OpenGL histogram functionality for occlusion culling. Their work involves histogramming the stencil buffer, instead of the color buffer as done in our work.

6 Conclusions

In this paper we presented a novel conservative visibility algorithm based on the non-conservative PLP algorithm. Our approach exploits several features of PLP to quickly estimate the correct image (and depth buffer) and to determine which portions of this estimation were incorrect. To complete our conservative approach, we required an efficient means of performing visibility queries with respect to the current estimation image. We showed how to implement these visibility queries using either hardware or software. If fragment-counting hardware is available (such as on HP fx, Diamond FireGL, SGI IR3), this is clearly the best choice. Otherwise, the item-buffer technique is the next best option. As graphics hardware continues to improve, and if the OpenGL histogramming features are further optimized, this approach may offer the highest levels of interactive rendering.

Our cPLP approach has several nice features. It provides a much higher level of interactivity than traditional rendering algorithms, such as view frustum culling. As opposed to PLP, cPLP provides a conservative visibility culling algorithm. The preprocessing required by our algorithm is very modest, and we are not required to store significant occlusion information, such as view-dependent occluders or potentially visible sets. We are also able to run our algorithm on all (polygonal) data sets since we do not require any underlying structure or format, such as connectivity information.

Further investigation is necessary to study the feasibility (cost) of adding our hardware extension proposed in Section 3.6 to current architectures. As we show in this paper, it can further improve the performance substantially over techniques that provide a single counter of the fragments that pass the depth-test, such as the HP occlusion-culling extension, since it is able to perform several test in parallel.

Acknowledgements

We thank Craig Wittenbrink for help with the occlusion culling capabilities of the HP fx series accelerators. Craig provided us with much needed material, including documentation and a sample implementation that showed us how to use the HP occlusion test. Many thanks to Prof. Carlo Sequin and his students at the University of California, Berkeley for the SODA Hall model used in our experiments. Thanks to Ned Greene for providing us with the Skyscraper dataset, and for helping in tracking down hard to find references. Thanks to David Kirk of Nvidia for discussions about the complexity of adding the per-fragment histogramming extension. Finally, we thank Dirk Bartz for comments on an earlier version of this paper.

References

[1] OpenGL histogram documentation. http://www.opengl.org/developers/documentation/Version1.2/1.2specs/-histogram.txt.

[2] L. Alonso and N. Holzschuch. Using graphics hardware to speed-up your visibility queries. *Journal of Graphics Tools*, to appear.

[3] D. Bartz, M. Meißner, and T. Hüttner. Extending graphics hardware for occlusion queries in OpenGL. *1998 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 97–104, August, 1998.

[4] D. Bartz, M. Meißner, and T. Hüttner. OpenGL-assisted occlusion culling for large polygonal models. *Computers & Graphics*, 23(5):667–679, October, 1999.

[5] F. Bernardini, J. T. Klosowski, and J. El-Sana. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum*, 19(3):507–516, August, 2000.

[6] Y. Chrysanthou, D. Cohen-Or, and D. Lischinski. Fast approximate quantitative visibility for complex scenes. *Computer Graphics International '98*, pages 220-229, June, 1998.

[7] D. Cohen-Or, Y. Chrysanthou, and C. Silva. A Survey of Visibility for Walkthrough Applications. *Submitted for publication, 2000.* Also in "Visibility, problems, techniques, and applications", ACM SIGGRAPH 2000 Course #4, 2000.

[8] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[9] R. Cunniff. Visualize fx graphics scalable architecture. In *Hot 3D Proceedings*, Graphics Hardware Workshop 2000, Interlaken, Switzerland, August, 2000.

[10] F. Durand. *3D Visibility: Analytical study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France, July, 1999.

[11] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July, 2000.

[12] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Proceedings of SIGGRAPH 1980*, pages 124–133, 1980.

[13] N. Greene. Occlusion Culling with Optimized Hierarchical Buffering. In *Proc. ACM SIGGRAPH'99 Sketches and Applications*, page 261, August, 1999.

[14] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

[15] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. *Proceedings of SIGGRAPH 97*, pages 27–34, 1997.

[16] J. T. Klosowski and C. T. Silva. Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99*, pages 115–122, October, 1999.

[17] J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, April - June, 2000.

[18] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, June, 2000.

[19] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *1995 ACM Symposium on Interactive 3D Graphics*, pages 105–106, 1995.

[20] D. Meagher. Efficient synthetic image generation of arbitrary 3-d objects. In *Proceedings of IEEE Conference on Pattern Recognition and Image Processing*, pages 473–478, June, 1982.

[21] S. Morein. ATI Radeon Hyper-Z technology. In *Hot 3D Proceedings*, Graphics Hardware Workshop 2000, Interlaken, Switzerland, August, 2000.

[22] B. F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, May, 1992.

[23] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. *Computers and Graphics*, 23(5):635–643, 1999.

[24] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July, 2000.

[25] N. Scott, D. Olsen, and E. Gannet. An overview of the visualize fx graphics accelerator hardware. *The Hewlett-Packard Journal*, May:28–34, 1998.

[26] K. Severson. VISUALIZE Workstation Graphics for Windows NT. HP product literature.

[27] Silicon Graphics, Inc. SGI Visual Workstation OpenGL Programming Guide for Windows NT. Document Number 007-3876-001. https://www.sgi.com/developers/nt/sdk/files/-OpenGLEXT.pdf

[28] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July, 1991.

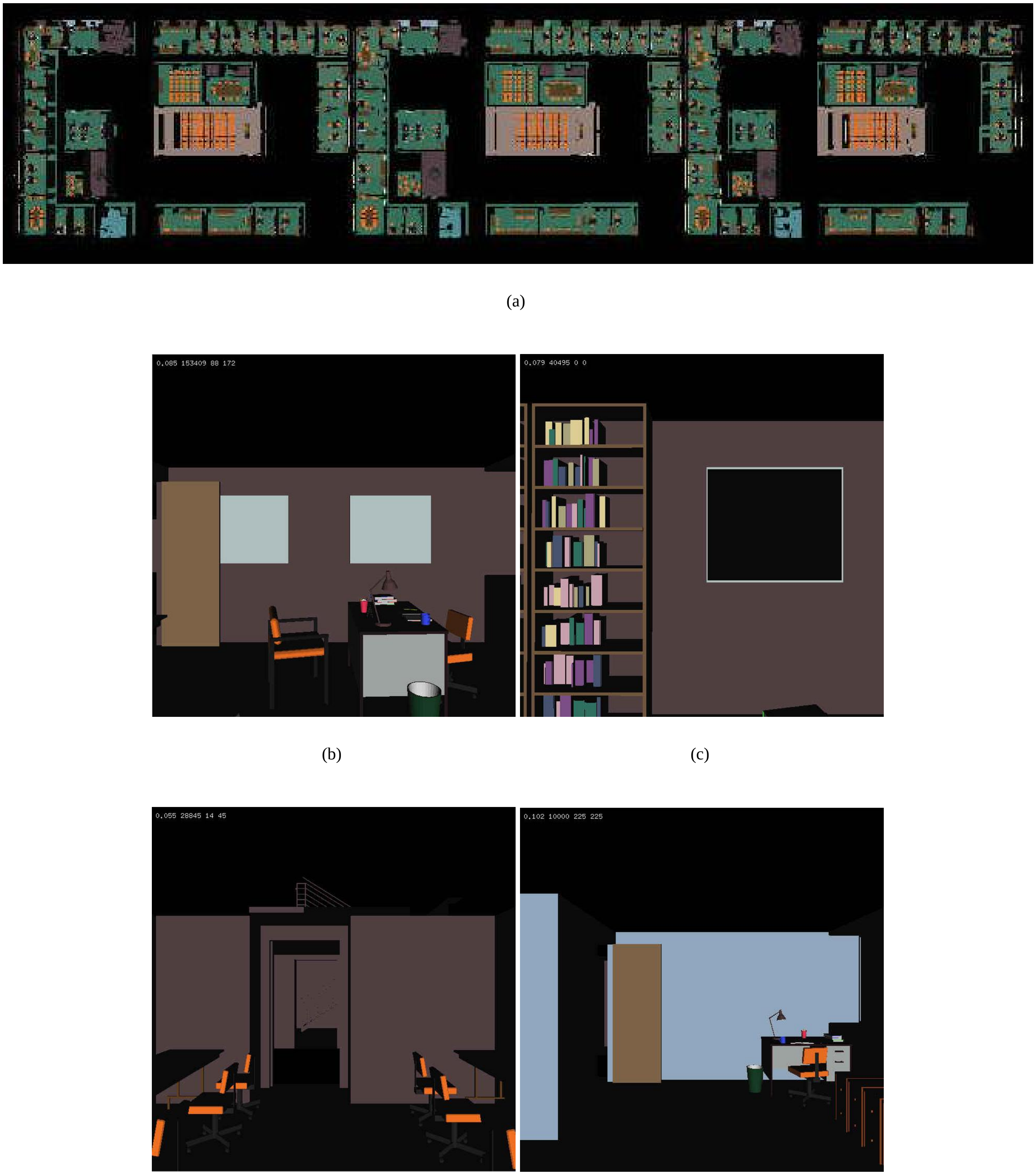
[29] R. Westermann, O. Sommer, and T. Ertl. Decoupling Polygon Rendering from Geometry using Rasterization Hardware. *Unpublished manuscript, 1999.*

[30] P. Wonka and D. Schmalsteig. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, September, 1999.

[31] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 71–82, June, 2000.

[32] F. Xie and M. Shantz. Adaptive hierarchical visibility in a tiled architecture. *1999 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 75–84, August, 1998.

[33] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, pages 77–88, 1997.



(a)

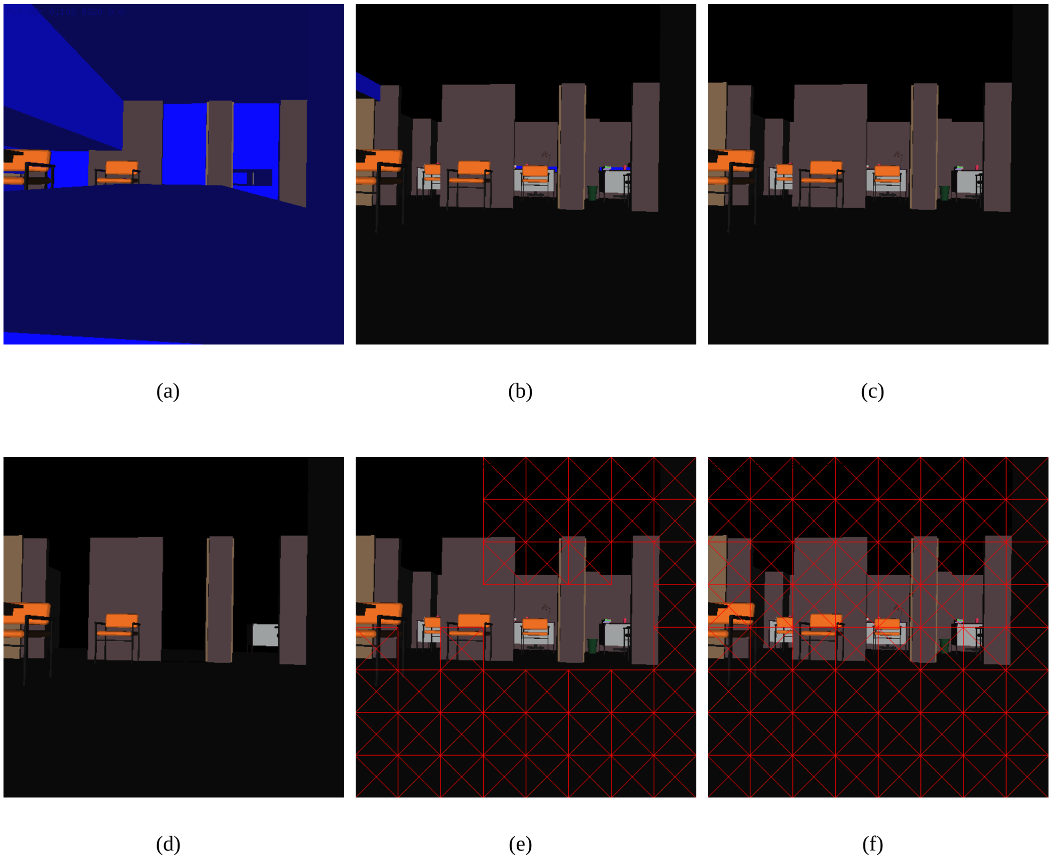
(b)

(c)

(d)

(e)

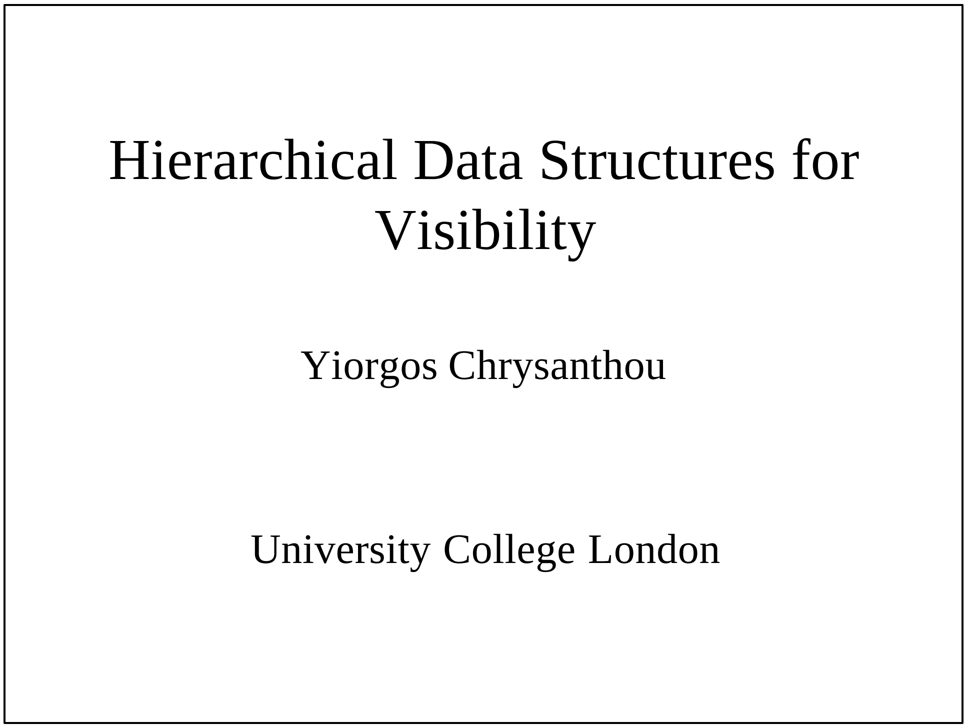
Figure 9: (a) A top-down view of our dataset. (b)–(e) Sample views of the recorded path.



(a) (b) (c)

(d) (e) (f)

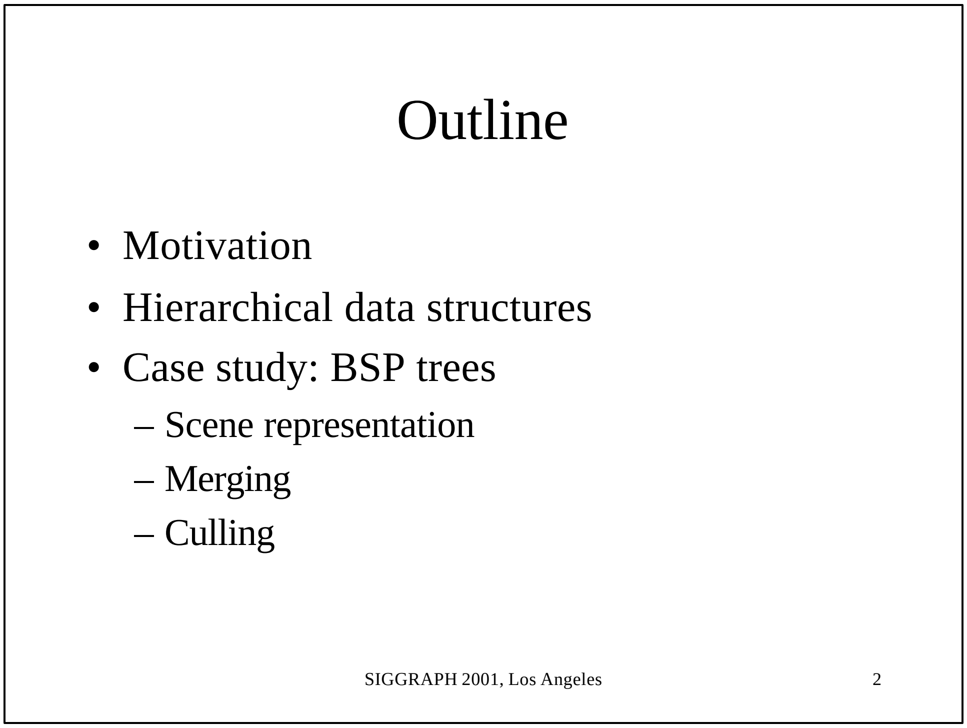
Figure 10: Snapshots during three iterations of our cPLP algorithm. The current front (blue) and completed tiles (red) are highlighted for iteration 1 in (a) and (d), iteration 2 in (b) and (e), and iteration 3 in (c) and (f). The final rendered image is (c).



Hierarchical Data Structures for Visibility

Yiorgos Chrysanthou

University College London



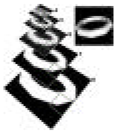
Outline

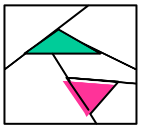
- Motivation
- Hierarchical data structures
- Case study: BSP trees
  - Scene representation
  - Merging
  - Culling

Where Are They Used?

- On the scene for the hierarchical classification

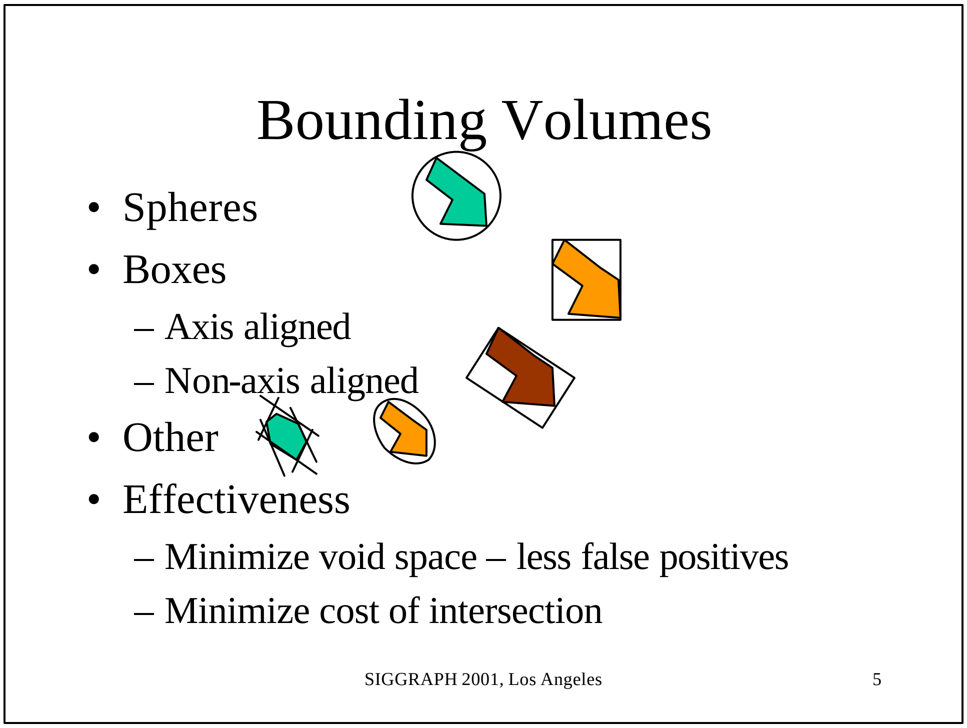
- For the viewspace to store pre-computed occluders and other information
- To store the occlusion representatione occlusion information





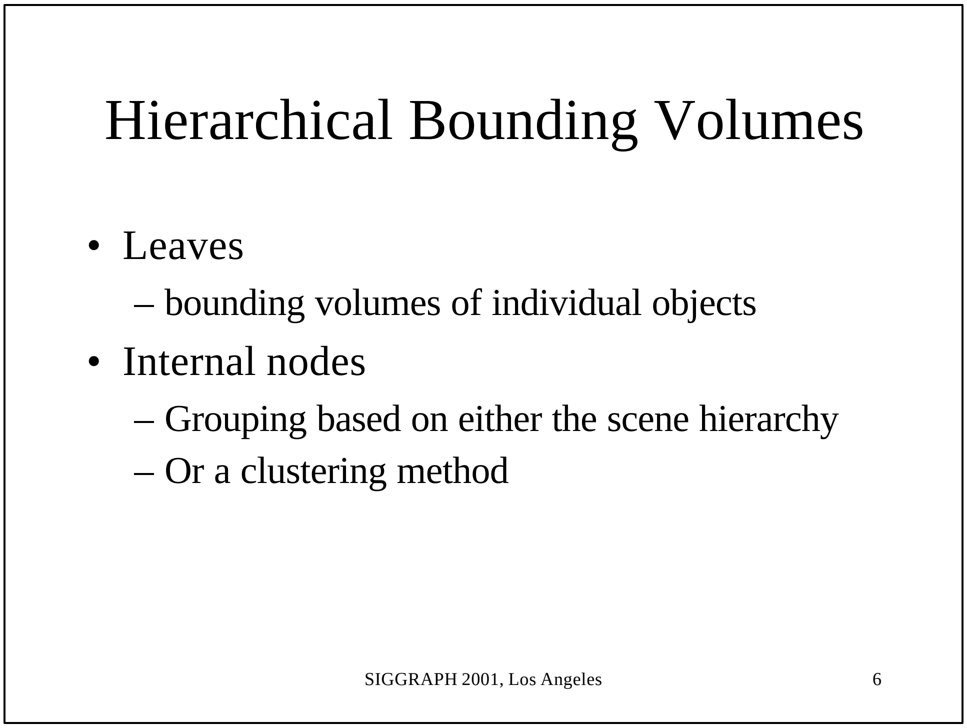
Types of Data Structures Used for Visibility

- Hierarchical bounding volumes
  - Object based
  - Possibly overlapping
- Hierarchical space partitioning
  - Space based
  - Convex cells
  - Disjoint cells



Bounding Volumes

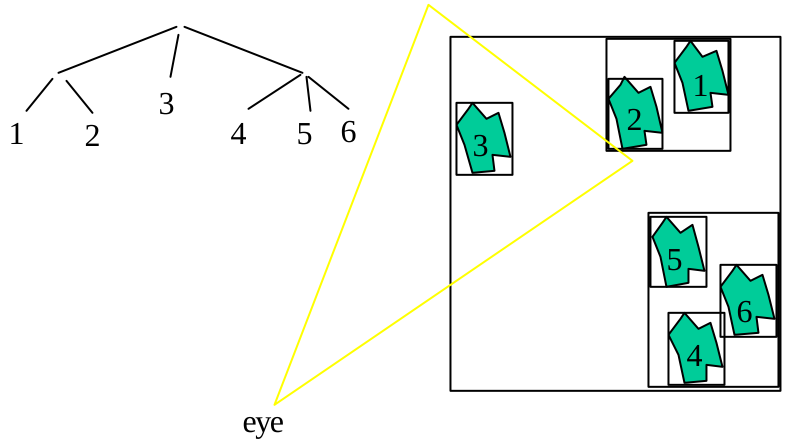
- Spheres
- Boxes
  - Axis aligned
  - Non-axis aligned
- Other
- Effectiveness
  - Minimize void space – less false positives
  - Minimize cost of intersection



Hierarchical Bounding Volumes

- Leaves
  - bounding volumes of individual objects
- Internal nodes
  - Grouping based on either the scene hierarchy
  - Or a clustering method

Example, Hierarchical View Volume Culling



Hierarchical Spatial Partitioning

- Oc-trees

  At each node split space half way along each of x, y and z using axis aligned planes

- Kd-trees

  At each node split space along one of the 3 dimensions using an axis aligned plane

- BSP trees

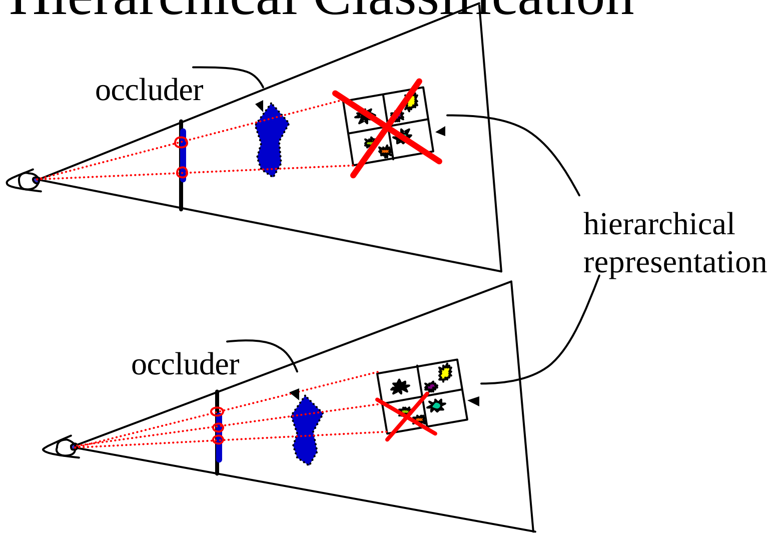
  At each node split space along one of the 3 dimensions using a NON axis aligned plane

Hierarchical Space Partitioning

- Ease of implementation
  - – Oc-trees, kd-trees, bsp trees
- Speed of intersection
  - – Oc-trees, kd-trees, bsp trees
- Flexibility / functionality
  - – bsp trees, kd-trees, Oc-trees

SIGGRAPH 2001, Los Angeles 9

Hierarchical Classification

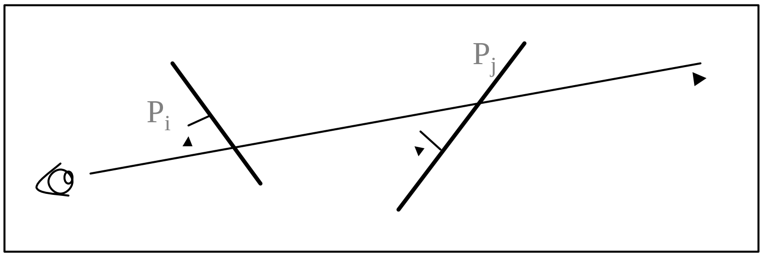


SIGGRAPH 2001, Los Angeles 10

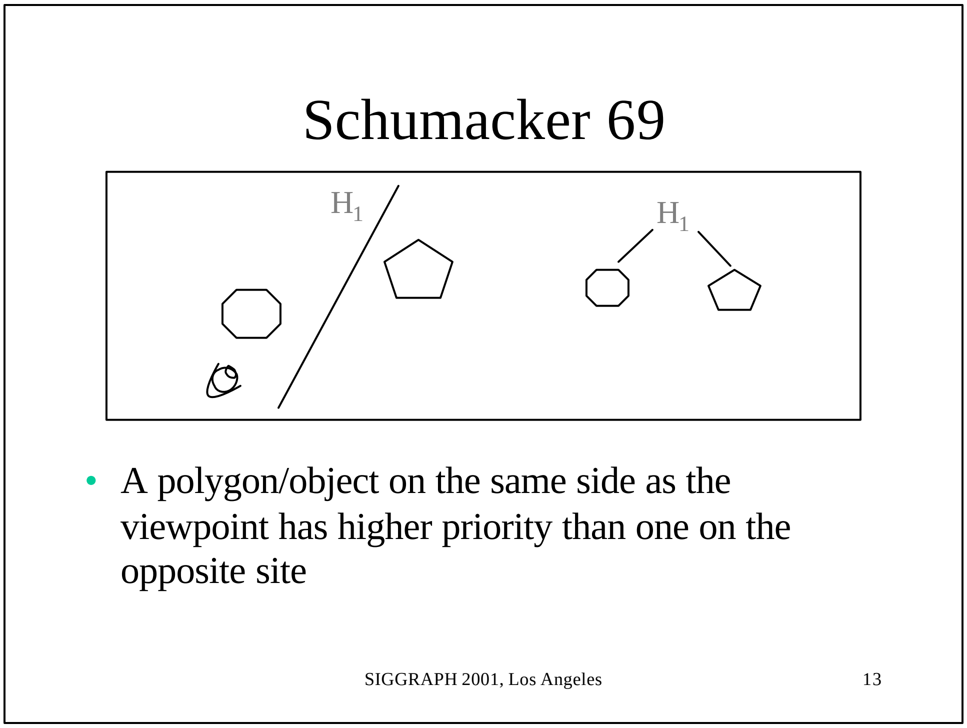
Case Study: BSP trees

- Visibility ordering
- BSP trees as a hierarchy of volumes
- Hierarchical visibility culling
- Tree merging
- Visibility culling using merging

Visibility (Priority) Ordering



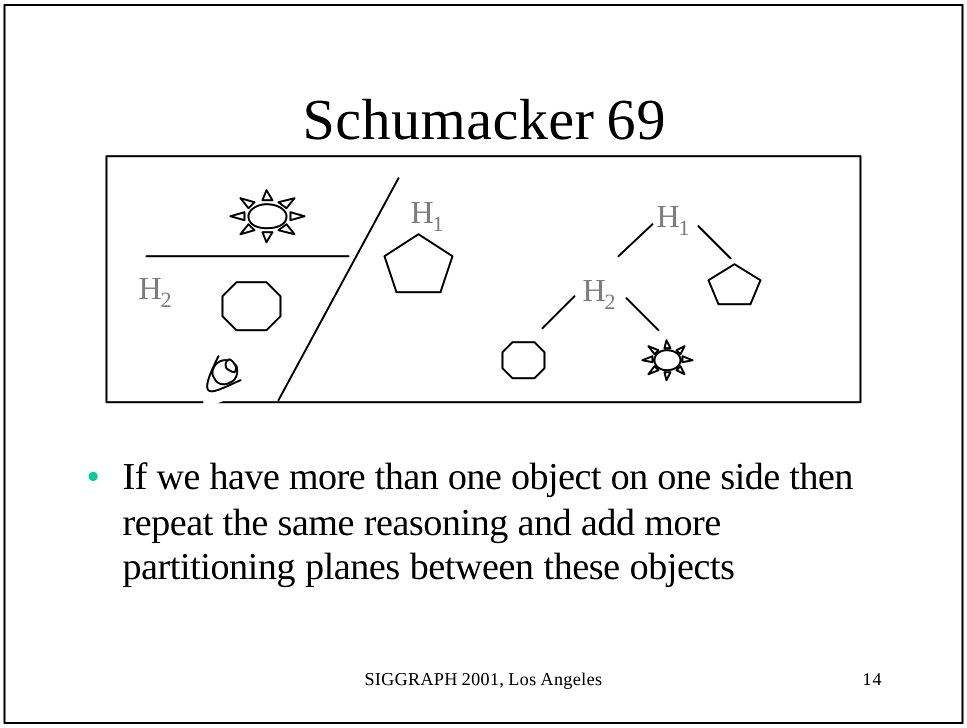
- Given a set of polygons *S* and a viewpoint *vp*, find an ordering on *S* st for any 2 polygons intersected by a ray through *vp* P_i has higher priority than P_j



Schumacker 69

- A polygon/object on the same side as the viewpoint has higher priority than one on the opposite site

SIGGRAPH 2001, Los Angeles 13



Schumacker 69

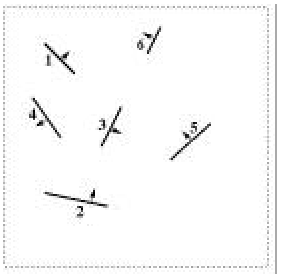
- If we have more than one object on one side then repeat the same reasoning and add more partitioning planes between these objects

SIGGRAPH 2001, Los Angeles 14

Binary Space Partitioning Trees
(Fuchs, Kedem and Naylor `80)

- More general, can deal with inseparable objects
- Automatic, uses as partitions planes defined by the scene polygons
- Method has two steps:
  - building of the tree independently of viewpoint
  - traversing the tree from a given viewpoint to get visibility ordering

Building a BSP Tree (Recursive)

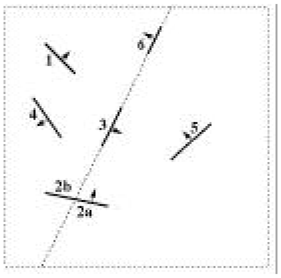


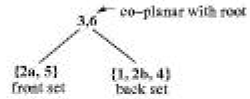
{1, 2, 3, 4, 5, 6}

A set of polygons

The tree

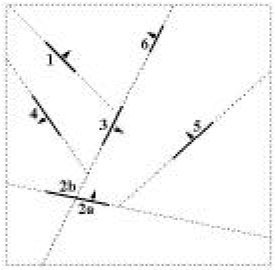
Building a BSP Tree (Recursive)

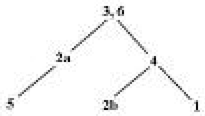




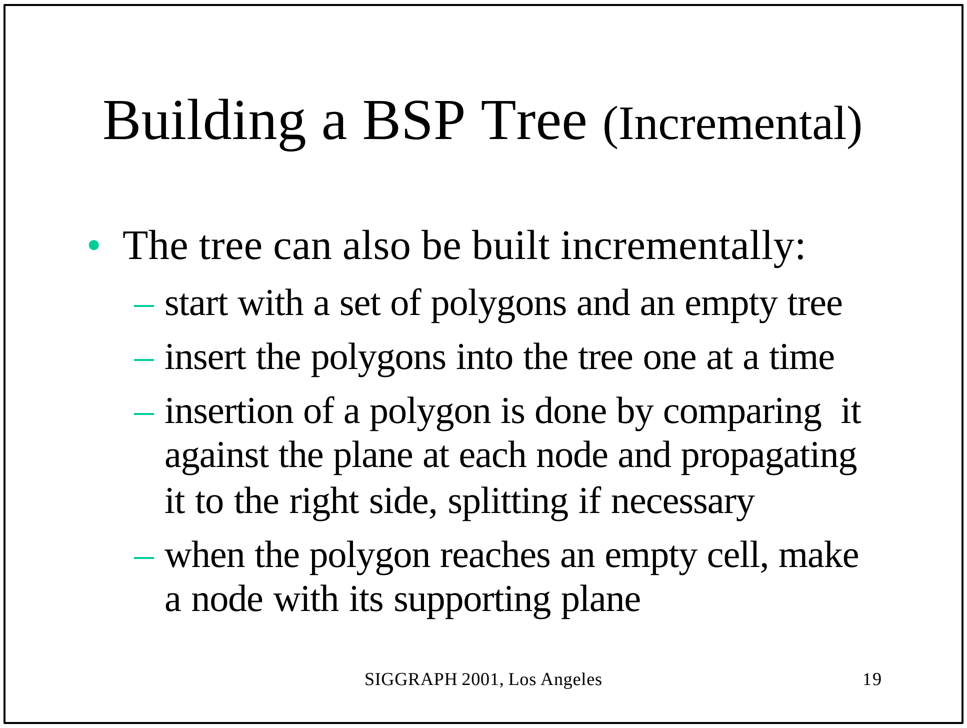
Select one polygon and partition the space and the polygons

Building a BSP Tree (Recursive)





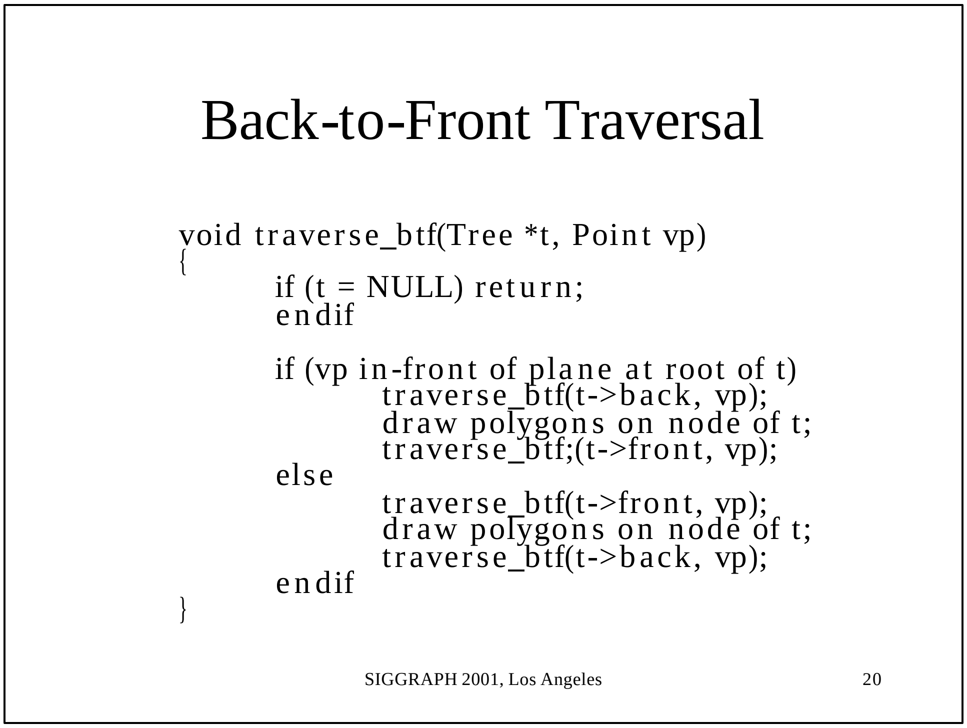
Recursively partition each sub-tree until all polygons are used up



Building a BSP Tree (Incremental)

- The tree can also be built incrementally:
  - start with a set of polygons and an empty tree
  - insert the polygons into the tree one at a time
  - insertion of a polygon is done by comparing it against the plane at each node and propagating it to the right side, splitting if necessary
  - when the polygon reaches an empty cell, make a node with its supporting plane

SIGGRAPH 2001, Los Angeles 19



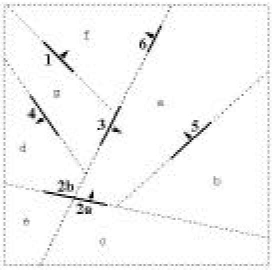
Back-to-Front Traversal

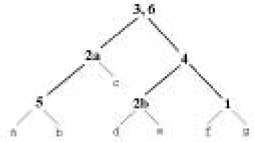
```
void traverse_btf(Tree *t, Point vp)
{
        if (t = NULL) return;
        endif

        if (vp in-front of plane at root of t)
                traverse_btf(t->back, vp);
                draw polygons on node of t;
                traverse_btf;(t->front, vp);
        else
                traverse_btf(t->front, vp);
                draw polygons on node of t;
                traverse_btf(t->back, vp);
        endif
}
```

SIGGRAPH 2001, Los Angeles 20

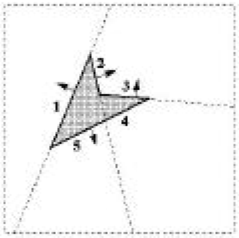
The BSP as a Hierarchy of Spaces

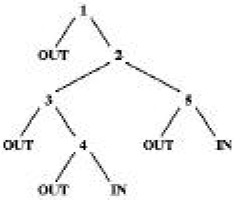


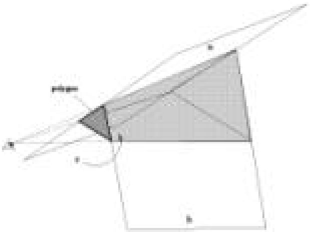


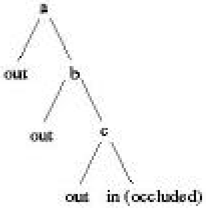
- Each node corresponds to a region of space
  - the root is the whole of R^n
  - the leaves are homogeneous regions

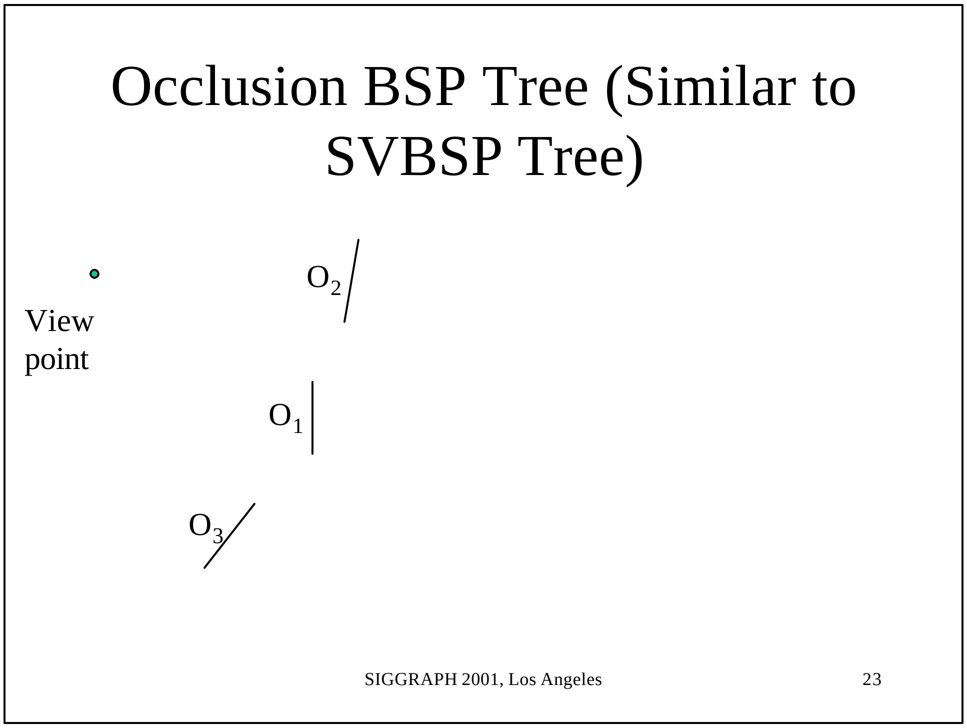
BSP Representation of Polyhedra





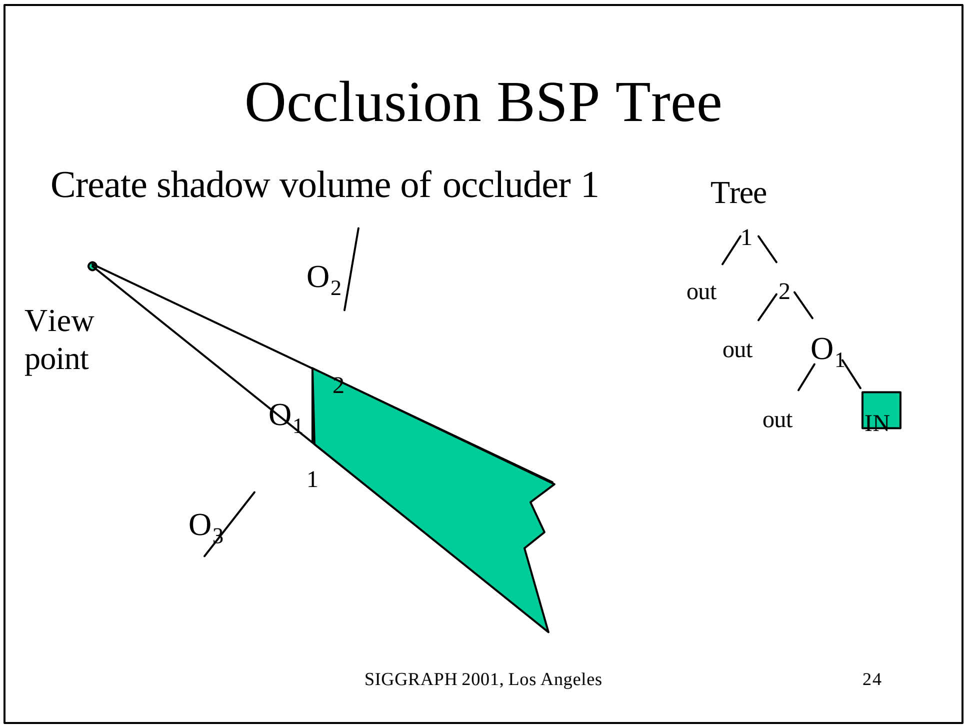






Occlusion BSP Tree (Similar to SVBSP Tree)

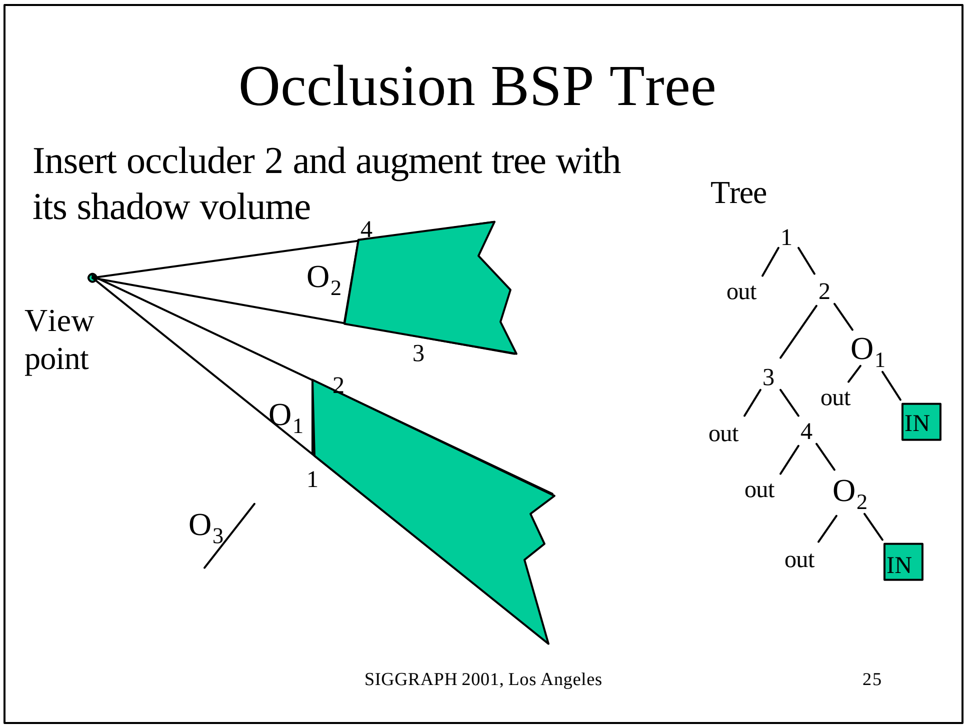
SIGGRAPH 2001, Los Angeles 23



Occlusion BSP Tree

Create shadow volume of occluder 1

SIGGRAPH 2001, Los Angeles 24

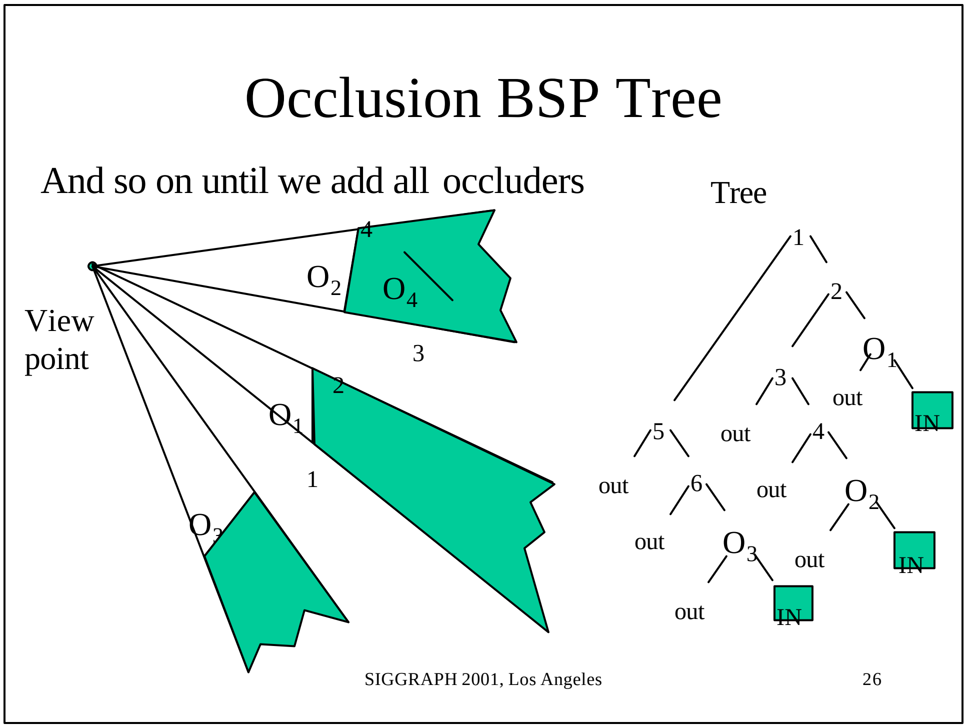


Occlusion BSP Tree

Insert occluder 2 and augment tree with its shadow volume

SIGGRAPH 2001, Los Angeles

25



Occlusion BSP Tree

And so on until we add all occluders

SIGGRAPH 2001, Los Angeles

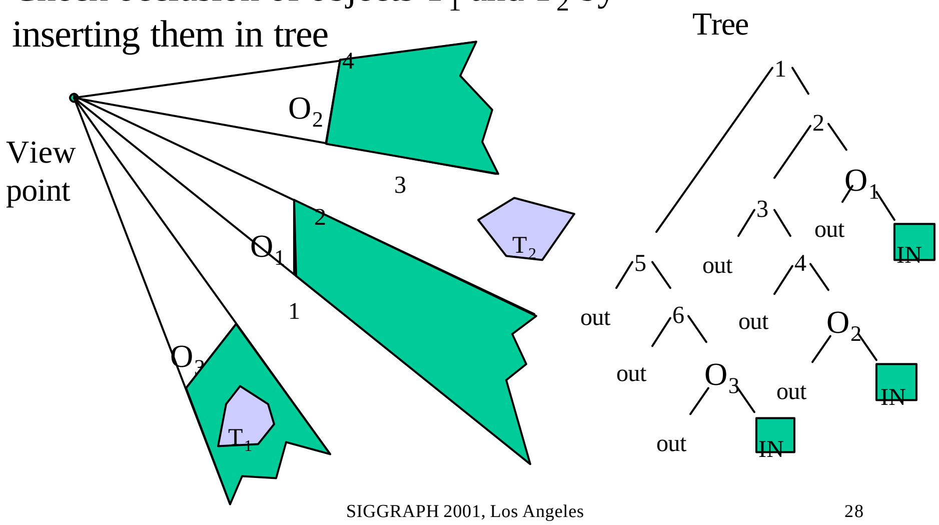
26

Example of Using BSP Trees for Visibility

- Extended Hudson method
  - – instead of constructing a shadow volume for each occluder, construct an occlusion BSP tree and compare the objects against the aggregate occlusion.
  - – If we have *N* occluders then we need *O (logN)* comparisons for each object

Occlusion BSP Tree

Check occlusion of objects T_1 and T_2 by inserting them in tree

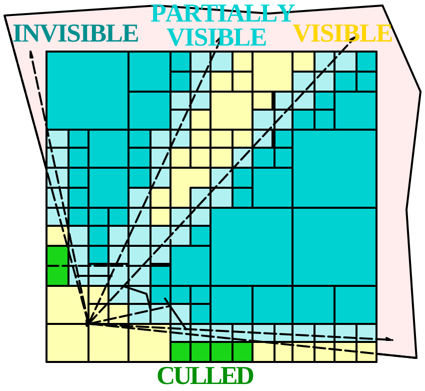


Hierarchical Occlusion Culling Using the Occlusion Tree (Bittner 98)

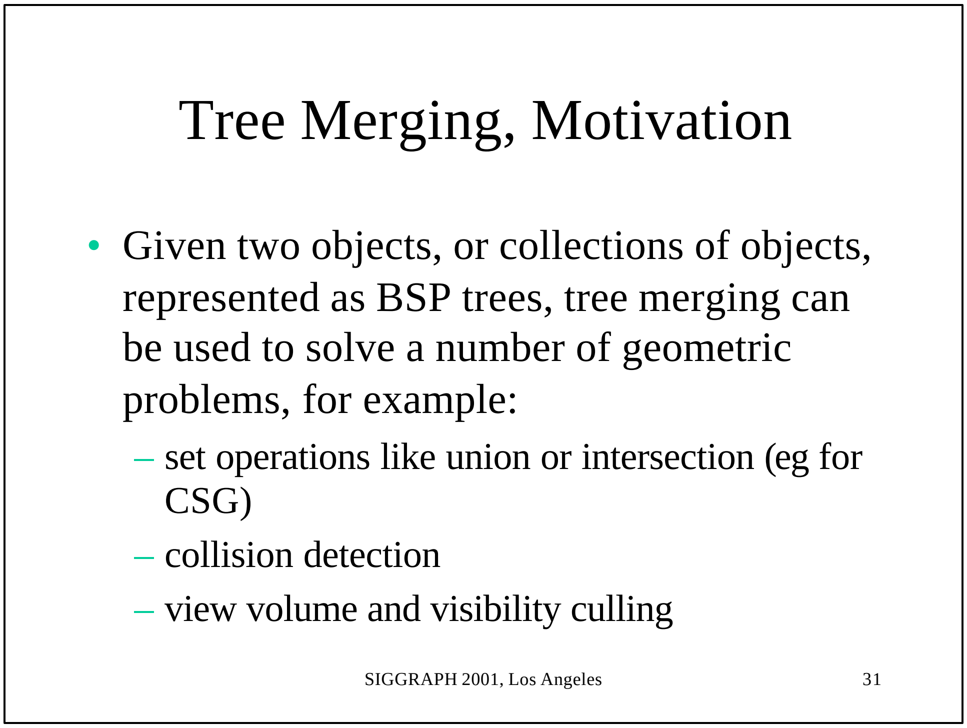
- Scene is represented by a k-d tree
- For a given viewpoint:
  - select a set of potential occluders
  - build an occlusion tree from these occluders
  - hierarchically compare the k-d nodes against the occlusion tree

Hierarchical Visibility Algorithm

- Viewpoint-to-region visibility
  - visible
  - invisible
  - partially visible

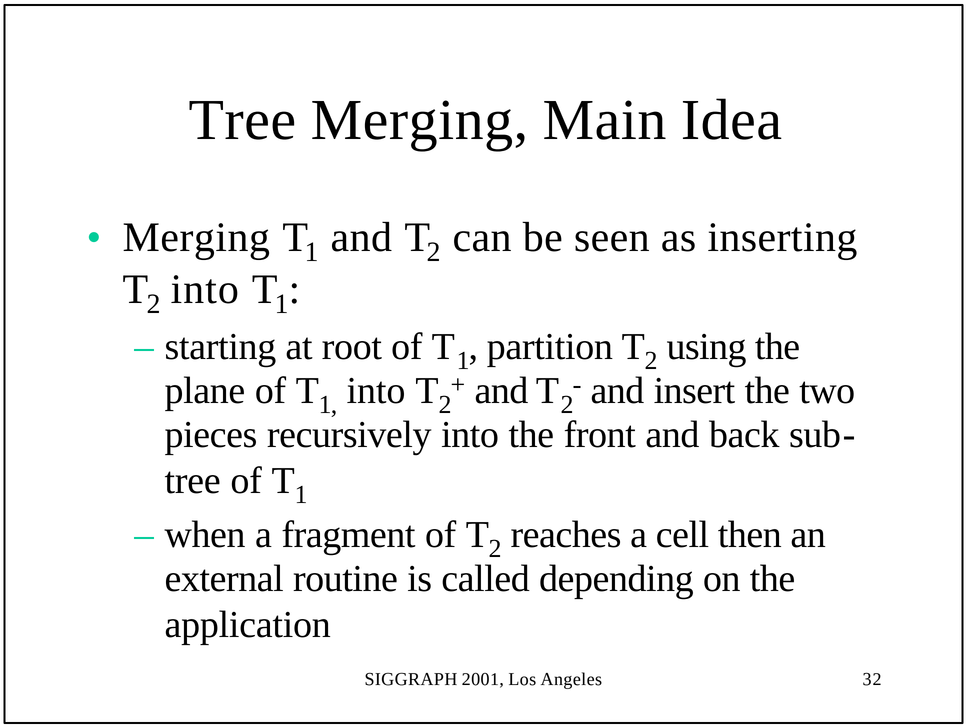


- Refinement of partially visible regions



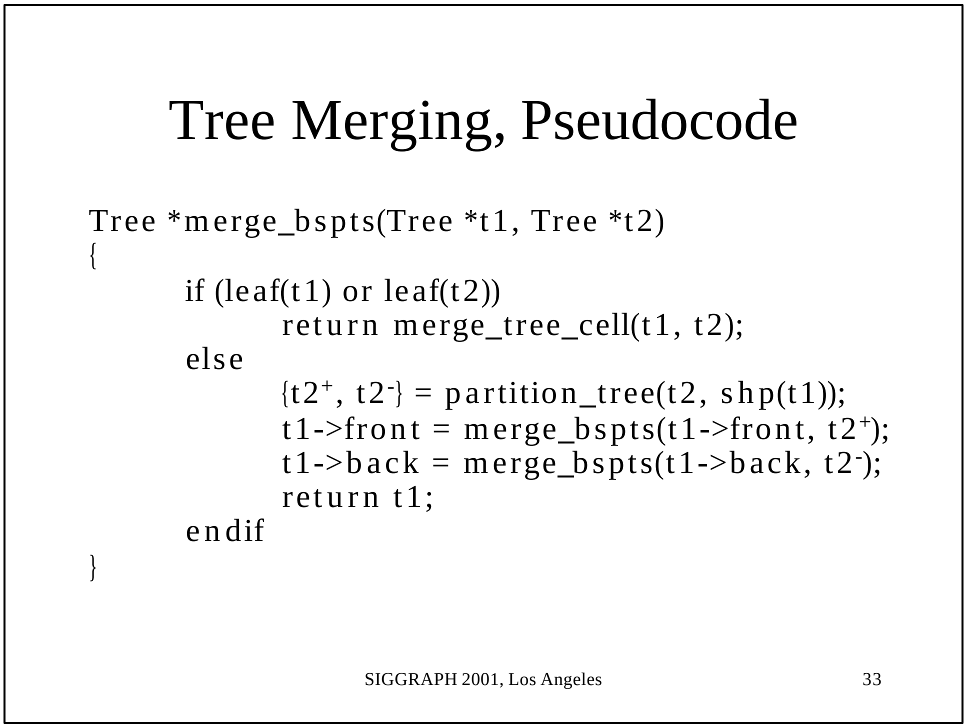
Tree Merging, Motivation

- Given two objects, or collections of objects, represented as BSP trees, tree merging can be used to solve a number of geometric problems, for example:
  - set operations like union or intersection (eg for CSG)
  - collision detection
  - view volume and visibility culling



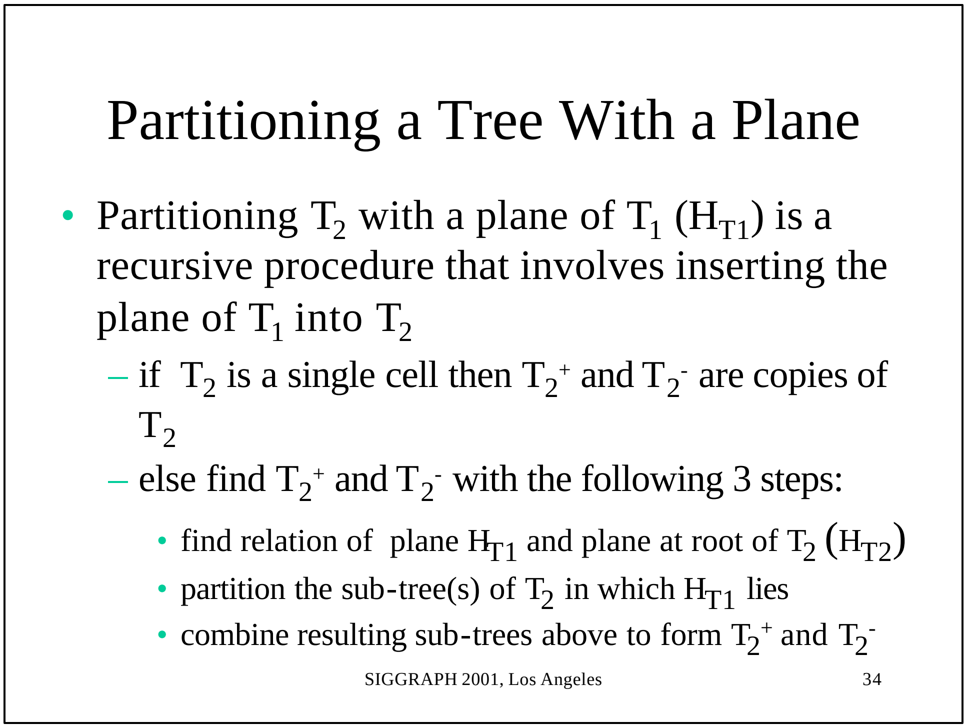
Tree Merging, Main Idea

- Merging T_1 and T_2 can be seen as inserting T_2 into T_1:
  - starting at root of T_1, partition T_2 using the plane of $T_{1,}$ into T_2^+ and T_2^- and insert the two pieces recursively into the front and back subtree of T_1
  - when a fragment of T_2 reaches a cell then an external routine is called depending on the application



Tree Merging, Pseudocode

```
Tree *merge_bspts(Tree *t1, Tree *t2)
{
        if (leaf(t1) or leaf(t2))
                return merge_tree_cell(t1, t2);
        else
                {$t2^+$, $t2^-$} = partition_tree(t2, shp(t1));
                t1->front = merge_bspts(t1->front, $t2^+$);
                t1->back = merge_bspts(t1->back, $t2^-$);
                return t1;
        endif
}
```



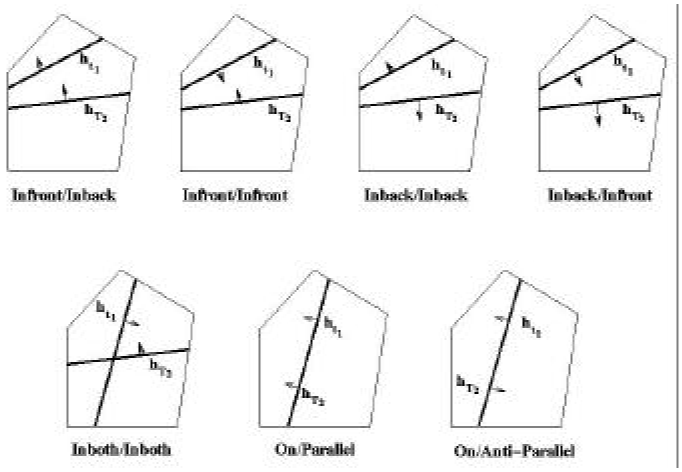
Partitioning a Tree With a Plane

- Partitioning T_2 with a plane of T_1 (H_{T1}) is a recursive procedure that involves inserting the plane of T_1 into T_2
  - if T_2 is a single cell then T_2^+ and T_2^- are copies of T_2
  - else find T_2^+ and T_2^- with the following 3 steps:
    - find relation of plane H_{T1} and plane at root of T_2 (H_{T2})
    - partition the sub-tree(s) of T_2 in which H_{T1} lies
    - combine resulting sub-trees above to form T_2^+ and T_2^-

Find Relation of H_{T1} and H_{T2}

- Note that we are interested only in the relation of the two planes within the space that T2 is defined (we should be talking of sub-hyperplanes)
- There are seven possible classifications which can be grouped into 3 sets: in one sub-tree, in both, coplanar
- For each classification we do two comparisons

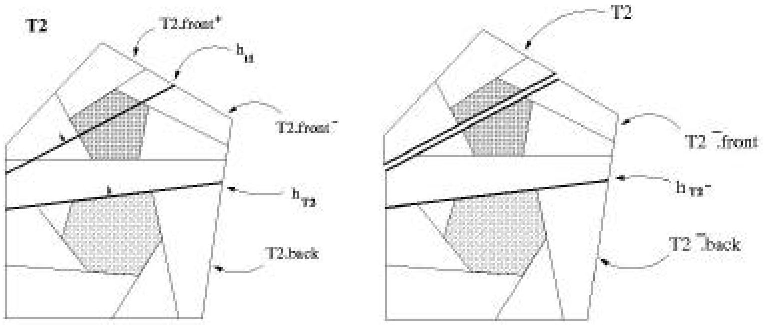
The Seven Classifications



Infront/Inback Infront/Infront Inback/Inback Inback/Infront

Inboth/Inboth On/Parallel On/Anti-Parallel

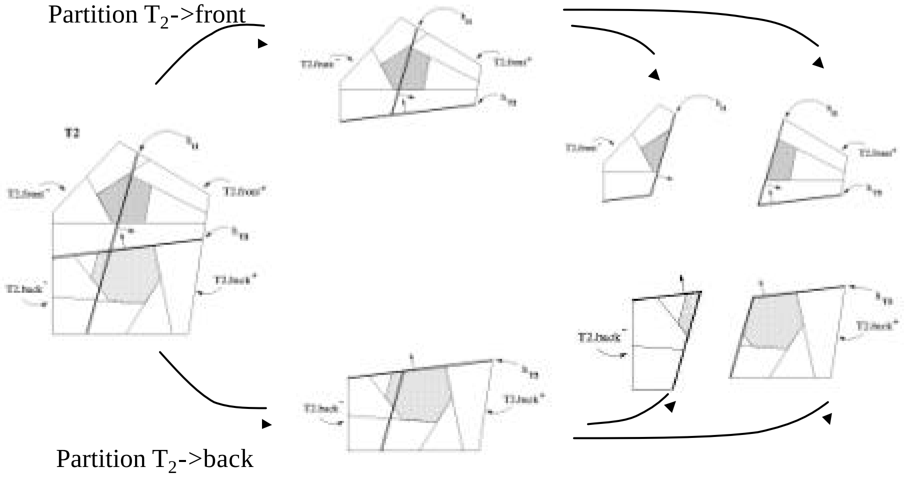
The Case Infront/Inback



T_2^-->front = (T_2->front)$^-$
T_2^-->root = T_2->root
T_2^-->back = T_2->back

T_2^+ = (T_2->front)$^+$

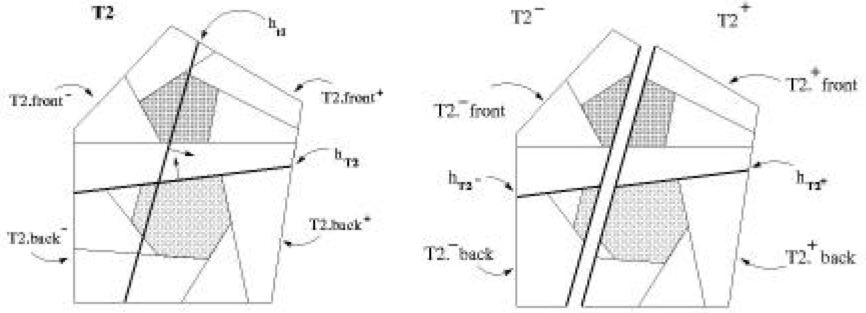
The Case Inboth/Inboth



Partition T_2->front

Partition T_2->back

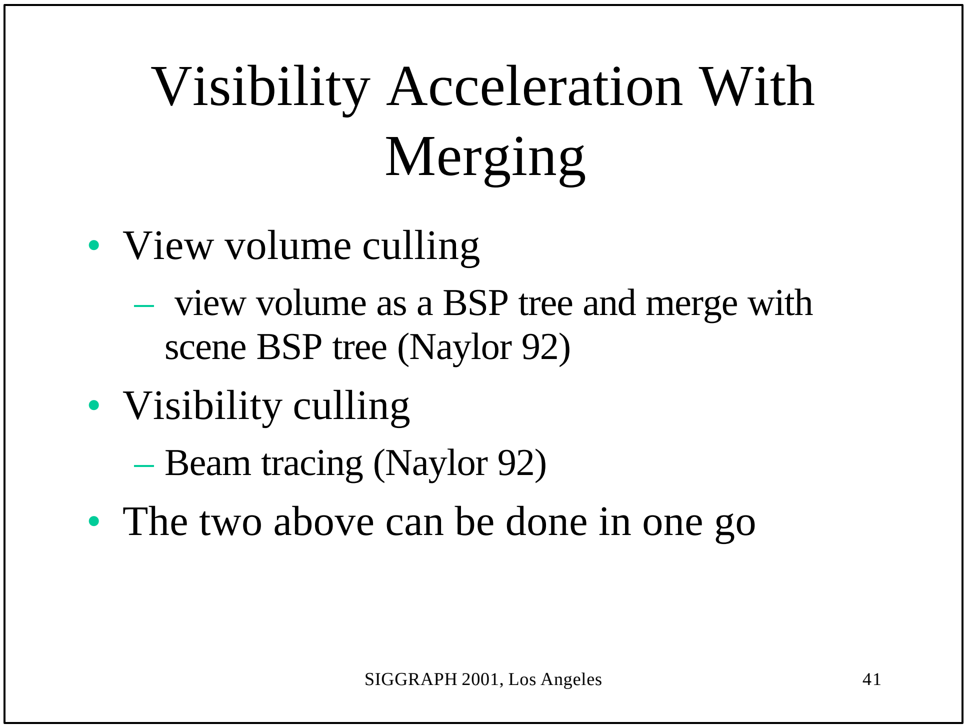
The Case Inboth/Inboth (cont.)



T_2^+->front = $(T_2$->front$)^+$	T_2^-->front = $(T_2$->front$)^-$
T_2^+->root = T_2->root	T_2^-->root = T_2->root
T_2^+->back = $(T_2$->back$)^+$	T_2^-->back = $(T_2$->back$)^-$

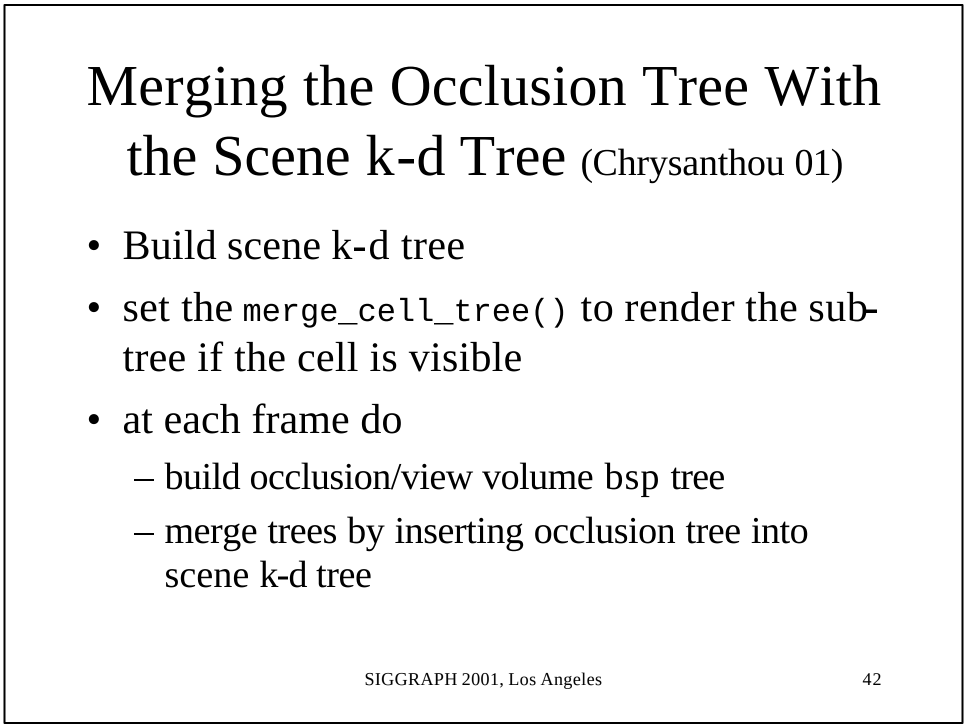
Example Uses of Tree Merging

- Constructive Solid Geometry (CSG)
- Collision detection
- Shadows from area light sources
- Discontinuity meshing



Visibility Acceleration With Merging

- View volume culling
  - view volume as a BSP tree and merge with scene BSP tree (Naylor 92)
- Visibility culling
  - Beam tracing (Naylor 92)
- The two above can be done in one go



Merging the Occlusion Tree With the Scene k-d Tree (Chrysanthou 01)

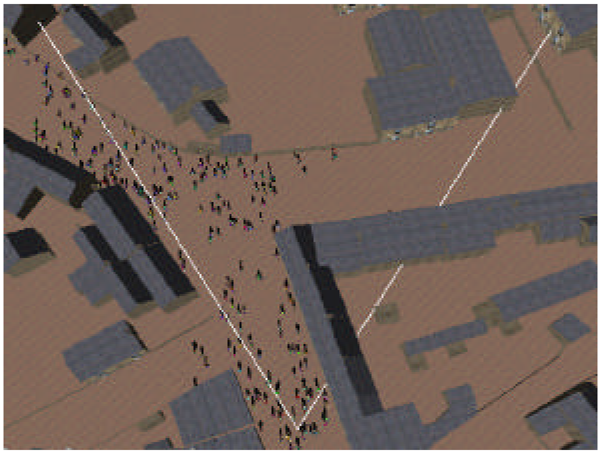
- Build scene k-d tree
- set the `merge_cell_tree()` to render the sub-tree if the cell is visible
- at each frame do
  - build occlusion/view volume bsp tree
  - merge trees by inserting occlusion tree into scene k-d tree

Culling Using Tree Merging

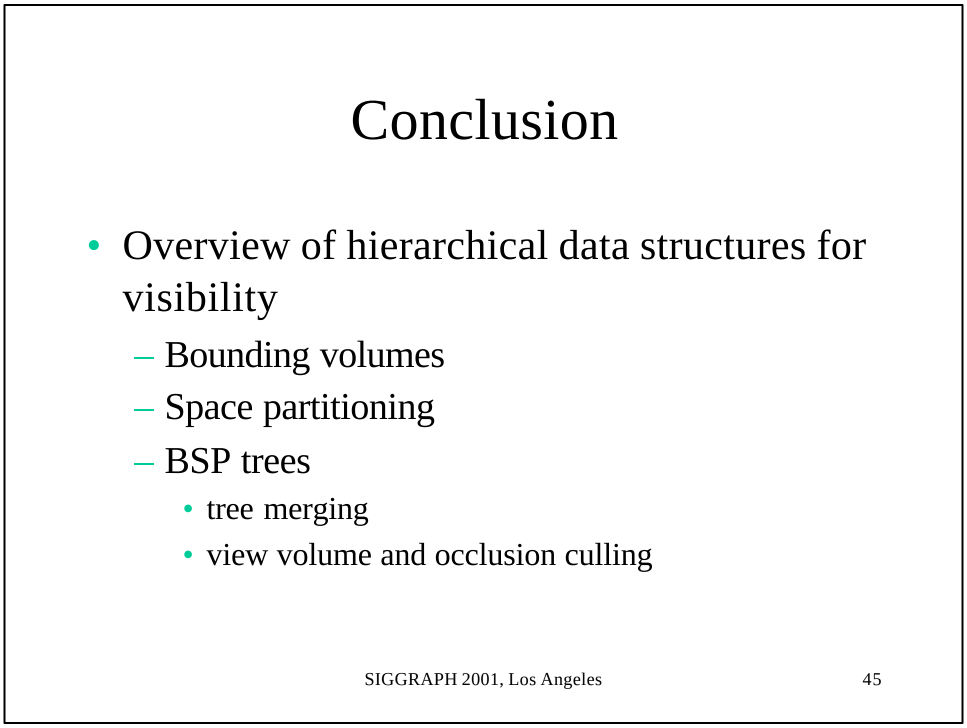
- We get a very efficient occlusion/view volume comparison of the scene
- The traversal is done in order (we use this to help our image based rendering technique)
- The occluders are selected in the same traversal

SIGGRAPH 2001, Los Angeles 43

Culling Populated Environments



SIGGRAPH 2001, Los Angeles 44

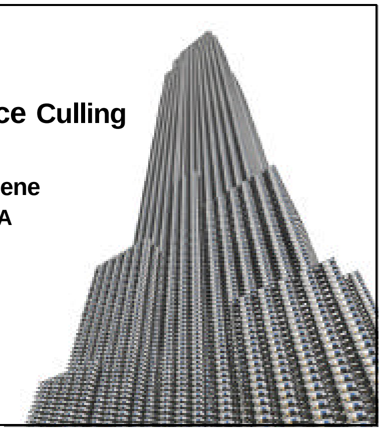


Conclusion

- Overview of hierarchical data structures for visibility
  - Bounding volumes
  - Space partitioning
  - BSP trees
    - tree merging
    - view volume and occlusion culling

Image Space Culling

Ned Greene
NVIDIA



General Approach

- Maintain coverage and depth info. as image is rendered
- Cull occluded geometry on-the-fly

Advantages

- Efficiency: Occlusion by actual scene geometry.
- Easy to fully exploit occluder fusion.
- Well suited to hierarchical methods.
- Well suited to hardware implementation.

Methods to be Discussed

- Hierarchical Z-Buffering
- Hierarchical Polygon Tiling
- *Umbra* Culling System (Aila & Miettinen)
- Sudarsky & Gotsman ‘99 (dynamic scenes)
- From-region vis. with 4D Hier. ZB.

Related Methods Covered Elsewhere

- HOMs (Zhang, Manoca, et al. ‘97)
- BSP-Tree Methods
- Klosowski & Silva ‘01 (PLP)
- Hardware-Assisted Box Culling

The “Hierarchical Visibility” Algorithm

- Hierarchical Z-Buffering
  (Greene, Kass, Miller, Siggraph ‘93)
- Hierarchical Polygon Tiling
  (Greene, Siggraph ‘96)

Hierarchical Visibility

- employs object-space and image-space hierarchies
- enables hierarchical culling of occluded geometry
- result: finds visible geometry by logarithmic search

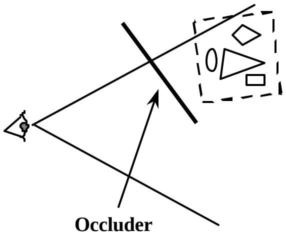
Object Space

- organize scene model in an octree
- traverse octree cubes front-to-back, culling occluded cubes

Image Space

- image vis. info. in a pyramid
- perform image-space culling hierarchically

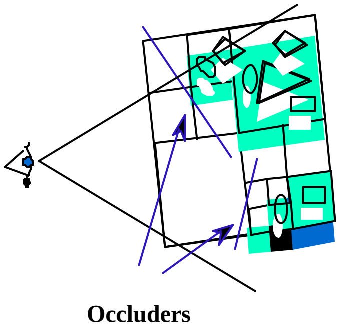
- If an octree cube is occluded by a z-buffer, all geometry inside the cube is also occluded.
- Apply recursively through the octree.



Recursive Subdivision Algorithm

Start with root cube.

- If octree cube is outside the viewing frustum, done.
- Test faces of the octree cube to see if it's visible; if occluded, done.
- Render geometry associated with octree cube.
- Subdivide octree node in front-to-back order, applying same algorithm to children.



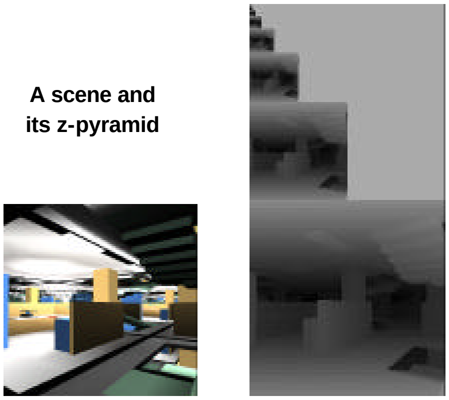
Properties of the Algorithm

- Only visits visible octree nodes and their children.
- Only renders geometry in visible octree nodes.

z-pyramid

- Finest level is a standard z-buffer.
- Each pyramid sample is the farthest sample in the corresponding 2x2 window at the next level.
- Each sample represents the farthest z for a square window of the screen.

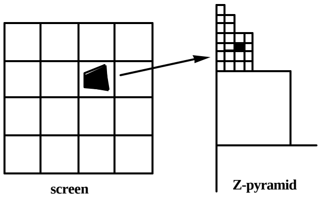
A scene and its z-pyramid



Does a Z-pyramid completely occlude a primitive?

Step 1:

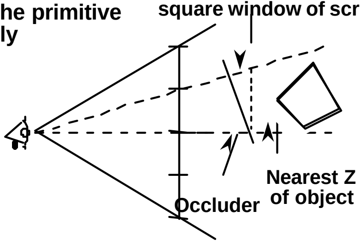
Find the finest-level pyramid value whose corresponding image region encloses the primitive.



Does a Z-pyramid completely occlude a primitive?

Step 2:

If nearest z of primitive is farther away than this value, the primitive is completely occluded.



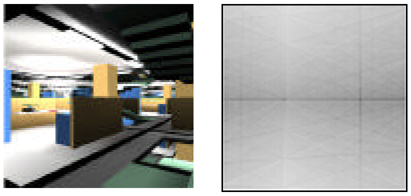
Z-pyramid value for square window of screen

Occluder

Nearest Z of object

Depth Complexity of the Visibility Computation

Naive Z-Buffering



avg. depth 83.7

Depth Complexity of Visibility Computation

Hierarchical Visibility (log scale)

z-pyramid tests	tiling polygons	total
.45	2.51	2.96

Hierarchical Visibility - Conclusion

- Works on arbitrary models.
- Effectively exploits object-space, image-space, and temporal coherence.
- Suitable for hardware acceleration.

Hierarchical Polygon Tiling with Coverage Masks

(Siggraph ‘96)

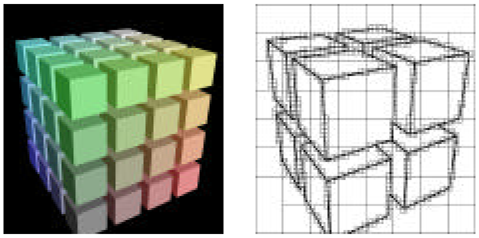
Modifications to Hierarchical Z-Buffering

- substitute <u>hierarchical tiling</u> for <u>z-buffering</u>
- substitute <u>coverage pyramid</u> for <u>z-pyramid</u>
- substitute <u>octree of BSP trees</u> for <u>octree</u>

Hierarchical Polygon Tiling

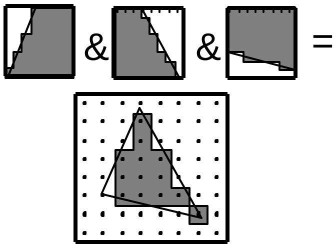
- tiling by recursive subdivision
- subdivision driven by 3-state *triage coverage masks*
- tiling and visibility done with bitmask operations (fast!)

Hierarchical tiling by recursive subdivision (Warnock ‘69) using “triage” (3-state) coverage masks

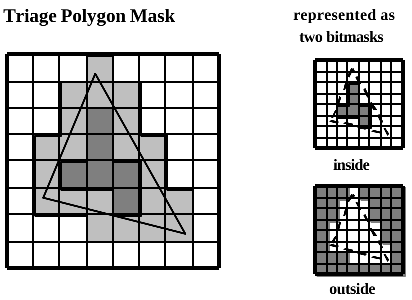


Making a Polygon Mask

- look up edges masks
- AND them together

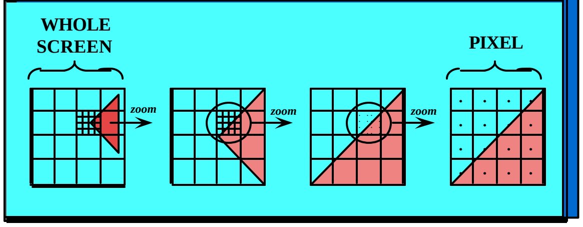


Triage Polygon Mask



Recursive Tiling Procedure

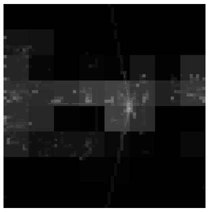
- ignore outside cells
- write image in covered cells
- subdivide intersected cells



Advantages of Hierarchical Tiling over Hierarchical Z-Buffering

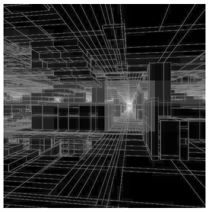
- faster
- uses much less memory when oversampling (e.g. 4%)
- no overwrite of raster samples

Disadvantage: Requires strict front-to-back traversal.



work tiling cubes

.09 cells visited per pixel



work tiling polygons

1.01 cells visited per pixel

3.1 seconds to tile and filter on 4096 x 4096 grid (75 mhz)

The Umbra Culling System
Aila & Miettinen '00

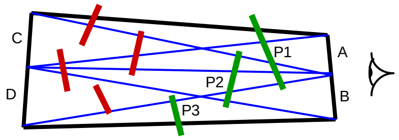
- Commercial scene-management software for conservative culling.
- Use silhouettes of foreground objects as occluders.
- Optimized variations of hierarchical polygon tiling and HOMs.
- Realtime performance for a limited class of complex scenes.

Dynamic Scenes
Sudarsky & Gotsman '99

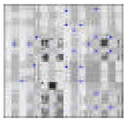
- Adapt hierarchical z-buffering and BSP-tree projections for dynamic scenes.
- Use "temporal bounding volumes" for dynamic objects.

From-region visibility with 4D Hierarchical Z-Buffering
(Greene '01)

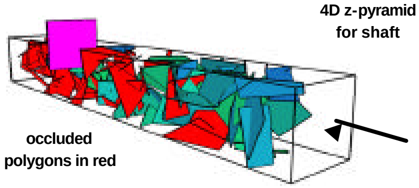
Determine visibility within a shaft by recursive subdivision.



A practical way to compute nearly exact from-region visibility for relatively simple polygonal scenes.



4D z-pyramid for shaft



Hierarchical Z-Buffer Visibility

Ned Greene* Michael Kass Gavin Miller

Apple Computer

(* and U. C. Santa Cruz)

Abstract

An ideal visibility algorithm should a) quickly reject most of the hidden geometry in a model and b) exploit the spatial and perhaps temporal coherence of the images being generated. Ray casting with spatial subdivision does well on criterion (a), but poorly on criterion (b). Traditional Z-buffer scan conversion does well on criterion (b), but poorly on criterion (a). Here we present a hierarchical Z-buffer scan-conversion algorithm that does well on both criteria. The method uses two hierarchical data structures, an object-space octree and an image-space Z pyramid, to accelerate scan conversion. The two hierarchical data structures make it possible to reject hidden geometry very rapidly while rendering visible geometry with the speed of scan conversion. For animation, the algorithm is also able to exploit temporal coherence. The method is well suited to models with high depth complexity, achieving orders of magnitude acceleration in some cases compared to ordinary Z-buffer scan conversion.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Hidden line/surface removal; J.6 [Computer-Aided Engineering]: Computer-Aided I.3.1 [Computer Graphics]: Hardware Architecture - Graphics Processors

Additional Key Words and Phrases: Octree, Pyramid, Temporal Coherence, Spatial Coherence, Z Buffer.

1 Introduction

Extremely complex geometric databases offer interesting challenges for visibility algorithms. Consider, for example, an interactive walk-through of a detailed geometric database describing an entire city, complete with vegetation, buildings, furniture inside the buildings and the contents of the furniture. Traditional visibility algorithms running on currently available hardware cannot come close to rendering scenes of this complexity at interactive rates and it will be a long time before faster hardware alone will suffice. In order to get the most out of available hardware, we need faster algorithms that exploit properties of the visibility computation itself.

There are at least three types of coherence inherent in the visibility computation which can be exploited to accelerate a visibility algorithm. The first is object-space coherence: in many cases a single computation can resolve the visibility of a collection of objects which are near each other in space. The second is image-space coherence: in many cases a single computation can resolve the visibility of an object covering a collection of pixels. The third is temporal coherence: visibility information from one frame can often be used to accelerate visibility computation for the next frame. Here we present a visibility algorithm which exploits all three of these types of coherence and sometimes achieves orders of magnitude acceleration compared with traditional techniques.

The dominant algorithms in use today for visibility computations are Z-buffer scan conversion and ray-tracing. Since Z buffers do not handle partially transparent surfaces well, we will restrict the discussion to models consisting entirely of opaque surfaces. For these models, only rays from the eye to the first surface are relevant for visibility, so the choice is between Z buffering and ray-casting (ray-tracing with no secondary rays).

Traditional Z buffering makes reasonably good use of image-space coherence in the course of scan conversion. Implementations usually do a set-up computation for each polygon and then an incremental update for each pixel in the polygon. Since the incremental update is typically much less computation than the set-up, the savings from image-space coherence can be substantial. The problem with the traditional Z-buffer approach is that it makes no use at all of object-space or temporal coherence. Each polygon is rendered independently, and no information is saved from prior frames. For extremely complex environments like a model of a city, this is very inefficient. A traditional Z-buffer algorithm, for example, will have to take the time to render every polygon of every object in every drawer of every desk in a building even if the whole building cannot be seen, because the traditional algorithm can resolve visibility only at the pixel level.

Traditional ray-tracing or ray-casting methods, on the other hand, make use of object-space coherence by organizing the objects in some type of spatial subdivision. Rays from the eye are propagated through the spatial subdivision until they hit the first visible surface. Once a ray hits a visible surface, there is no need to consider any of the surfaces in the spatial subdivisions further down along the ray, so large portions of the geometry may never have to be considered during rendering. This is an important improvement on Z buffering, but it makes no use of temporal or image-space coherence. While ray-casting algorithms that exploit temporal coherence have been explored, it seems extremely difficult to exploit image-space coherence in traditional ray casting algorithms.

Here we present a visibility algorithm which combines the strengths of both ray-casting and Z buffering. To exploit object-space coherence, we use an octree spatial subdivision of the type commonly used to accelerate ray tracing. To exploit image-space coherence, we augment traditional Z-buffer scan conversion with an image-space Z pyramid that allows us to reject hidden geometry very quickly. Finally, to exploit temporal coherence, we use the geometry that was visible in the previous frame to construct a starting point for the algorithm. The result is an algorithm which is orders of magnitude faster than traditional ray-casting or Z buffering for some models we have tried. The algorithm is not difficult to implement and works for arbitrary polygonal databases.

In section II, we survey the most relevant prior work on accelerating ray casting and scan conversion. In section III, we develop the data structures used to exploit object-space, image-space and temporal coherence. In section IV, we describe the implementation and show results for some complex models containing hundreds of millions of polygons.

2 Prior Work

There have been many attempts to accelerate traditional ray-tracing and Z buffering techniques. Each of these attempts exploits some aspect of the coherence inherent in the visibility computation itself. None of them, however, simultaneously exploits object-space, image-space and temporal coherence.

The ray-tracing literature abounds with references to object-space coherence. A variety of spatial subdivisions have been used to exploit this coherence and they seem to work quite well (e.g. [1,2,3,4,5]). Temporal coherence is much less commonly exploited in practice, but various techniques exist for special cases. If all the objects are convex and remain stationary while the camera moves, then there are constraints on the way visibility can change [6] which a ray tracer might exploit. On the other hand, if the camera is stationary, then rays which are unaffected by the motion of objects can be detected and used from the previous frame [7]. When interactivity is not an issue and sufficient memory is available, it can be feasible to render an entire animation sequence at once using spacetime bounding boxes [8,9]. While these techniques make good use of object-space coherence and sometimes exploit temporal coherence effectively, they unfortunately make little or no use of image-space coherence since each pixel is traced independently from its neighbors. There are heuristic methods which construct estimates of the results of ray-tracing a pixel from the results at nearby pixels (e.g. [10]), but there seems to be no guaranteed algorithm which makes good use of image-space coherence in ray tracing.

With Z-buffer methods (and scan conversion methods in general) the problems are very different. Ordinary Z-buffer rendering is usually implemented with an initial set-up computation for each primitive followed by a scan-conversion phase in which the affected pixels are incrementally updated. This already makes very good use of image-space coherence, so the remaining challenge with Z-buffer methods is to exploit object-space and temporal coherence effectively.

A simple method of using object-space coherence in Z-buffer rendering is to use a spatial subdivision to cull the model to the viewing frustum [11]. While this can provide substantial acceleration, it exploits only a small portion of the object-space coherence in models with high depth complexity. In architectural models, for example, a great deal of geometry hidden behind walls may lie within the viewing frustum.

In order to make use of more of the object-space coherence in architectural models, Airey et. al. [12,13] and subsequently Teller and Sequin [15] proposed dividing models up into a set of disjoint cells and precomputing the potentially visible set (PVS) of polygons from each cell. In order to render an image from any viewpoint within a cell, only the polygons in the PVS need be considered. These PVS schemes are the closest in spirit to the visibility algorithm presented here since they attempt to make good use of both object-space and image-space coherence. Nonetheless, they suffer from some important limitations. Before they can be used at all, they require an expensive precomputation step to determine the PVS and a great deal of memory to store it. Teller and Sequin, for example, report over 6 hours of precomputation time on a 50 MIP machine to calculate 58Mb of PVS data needed for a model of 250,000 polygons [15]. Perhaps more importantly, the way these methods make use of cells may limit their appropriateness to architectural models. In order to achieve maximum acceleration, the cells must be 3D regions of space which are almost entirely enclosed by occluding surfaces, so that most cells are hidden from most other cells. For architectural models, this often works well since the cells can be rooms, but for outdoor scenes and more general settings, it is unclear whether or not PVS methods are effective. In addition, the currently implemented algorithms make very special use of axially-aligned polygons such as flat walls in rectilinear architectural models. While the methods can in principle be extended to use general 3D polygons for occlusion, the necessary algorithms have much worse computational complexity [15]. Finally, although the implementations prefetch PVS data for nearby cells to avoid long latencies due to paging, they cannot be said to exploit temporal coherence in the visibility computation very effectively.

The algorithm presented here shares a great deal with the work of Meagher [16] who used object-space octrees with image-space quadtrees for rendering purposes. Meagher tried to display the octree itself rather than using it to cull a polygonal database, so his method is directly applicable to volume, rather than surface models. Nonetheless his algorithm is one of the few to make use of both object-space and image-space coherence. The algorithm does not exploit temporal coherence.

3 Hierarchical Visibility

The hierarchical Z-buffer visibility algorithm uses an octree spatial subdivision to exploit object-space coherence, a Z pyramid to exploit image-space coherence, and a list of previously visible octree nodes to exploit temporal coherence. While the full value of the algorithm is achieved by using all three of these together, the object-space octree and the image-space Z pyramid can also be used separately. Whether used separately or together, these data structures make it possible to compute the same result as ordinary Z buffering at less computational expense.

3.1 Object-space octree

Octrees have been used previously to accelerate ray tracing [5] and rendering of volume data sets [16] with great effectiveness. With some important modification, many of the principles of these previous efforts can be applied to Z-buffer scan conversion. The result is an algorithm which can accelerate Z buffering by orders of magnitude for models with sufficient depth complexity.

In order to be precise about the octree algorithm, let us begin with some simple definitions. We will say that a polygon is hidden with respect to a Z buffer if no pixel of the polygon is closer to the observer than the Z value already in the Z buffer. Similarly, we will say that a cube is hidden with respect to a Z buffer if all of its faces are hidden polygons. Finally, we will call a node of the octree hidden if its associated cube is hidden. Note that these definitions depend on the sampling of the Z buffer. A polygon which is hidden at one Z-buffer resolution may not be hidden at another.

With these definitions, we can state the basic observation that makes it possible to combine Z buffering with an octree spatial subdivision: If a cube is hidden with respect to a Z buffer, then all polygons fully contained in the cube are also hidden. What this means is the following: if we scan convert the faces of an octree cube and find that each pixel of the cube is behind the current surface in the Z buffer, we can safely ignore all the geometry contained in that cube.

From this observation, the basic algorithm is easy to construct. We begin by placing the geometry into an octree, associating each primitive with the smallest enclosing octree cube. Then we start at the root node of the octree and render it using the following recursive steps: First, we check to see if the octree cube intersects the viewing frustum. If not, we are done. If the cube does intersect the viewing frustum, we scan convert the faces of the cube to determine whether or not the whole cube is hidden. If the cube is hidden, we are done. Otherwise, we scan convert any geometry associated with the cube and then recursively render its children in front-to-back order.

We can construct the octree with a simple recursive procedure. Beginning with a root cube large enough to enclose the entire model and the complete list of geometric primitives, we recursively perform the following steps: If the number of primitives is sufficiently small, we associate all of the primitives with the cube and exit. Otherwise, we associate with the cube any primitive which intersects at least one of three axis-aligned planes that bisect the cube. We then subdivide the octree cube and call the procedure recursively with each of the eight child cubes and the portion of the geometry that fits entirely in that cube.

The basic rendering algorithm has some very interesting properties. First of all, it only renders geometry contained in octree nodes which are not hidden. Some of the rendered polygons may be hidden, but all of them are “nearly visible” in the following sense: there is some place we could move the polygon where it would be visible which is no further away than the length of the diagonal of its containing octree cube. This is a big improvement over merely culling to the viewing frustum. In addition, the algorithm does not waste time on irrelevant portions of the octree since it only visits octree nodes whose parents are not hidden. Finally, the algorithm never visits an octree node more than once during rendering. This stands in marked contrast to ray-tracing through an octree where the root node is visited by every pixel and other nodes may be visited tens of thousands of times. As a result of these properties, the basic algorithm culls hidden geometry very efficiently.

A weakness of the basic algorithm is that it associates some small geometric primitives with very large cubes if the primitives happen to intersect the planes which separate the cube's children. A small triangle which crosses the center of the root cube, for example, will have to be rendered anytime the entire model is not hidden. To avoid this behavior, there are two basic choices. One alternative is to clip the problematic small polygons so they fit in much smaller octree cells. This has the disadvantage of increasing the number of primitives in the database. The other alternative is to place some primitives in multiple octree cells. This is the one we have chosen to implement. To do this, we modify the recursive construction of the octree as follows. If we find that a primitive intersects a cube's dividing planes, but is small compared to the cube, then we no longer associate the primitive with the whole cube. Instead we associate it with all of the cube's children that the primitive intersects. Since some primitives are associated with more than one octree node, we can encounter them more than once during rendering. The first time we render them, we mark them as rendered, so we can avoid rendering them more than once in a given frame.

3.2 Image-space Z pyramid

The object-space octree allows us to cull large portions of the model at the cost of scan-converting the faces of the octree cubes. Since the cubes may occupy a large number of pixels in the image, this scan conversion can be very expensive. To reduce the cost of determining cube visibility, we use an image-space Z pyramid. In many cases, the Z pyramid makes it possible to conclude very quickly a large polygon is hidden, making it unnecessary to examine the polygon pixel by pixel.

The basic idea of the Z pyramid is to use the original Z buffer as the finest level in the pyramid and then combine four Z values at each level into one Z value at the next coarser level by choosing the farthest Z from the observer. Every entry in the pyramid therefore represents the farthest Z for a square area of the Z buffer. At the coarsest level of the pyramid there is a single Z value which is the farthest Z from the observer in the whole image.

Maintaining the Z pyramid is an easy matter. Every time we modify the Z buffer, we propagate the new Z value through to coarser levels of the pyramid. As soon as we reach a level where the entry in the pyramid is already as far away as the new Z value, we can stop.

In order to use the Z pyramid to test the visibility of a polygon, we find the finest-level sample of the pyramid whose corresponding image region covers the screen-space bounding box of the polygon. If the nearest Z value of the polygon is farther away than this sample in the Z pyramid, we know immediately that the polygon is hidden. We use this basic test to determine the visibility of octree cubes by testing their polygonal faces, and also to test the visibility of model polygons.

While the basic Z-pyramid test can reject a substantial number of polygons, it suffers from a similar difficulty to the basic octree method. Because of the structure of the pyramid regions, a small polygon covering the center of the image will be compared to the Z value at the coarsest level of the pyramid. While the test is still accurate in this case, it is not particularly powerful.

A definitive visibility test can be constructed by applying the basic test recursively through the pyramid. When the basic test fails to show that a polygon is hidden, we go to the next finer level in the pyramid where the previous pyramid region is divided into four quadrants. Here we attempt to prove that the polygon is hidden in each of the quadrants it intersects. For each of these quadrants, we compare the closest Z value of the polygon in the quadrant to the value in the Z pyramid. If the Z-pyramid value is closer, we know the polygon is hidden in the quadrant. If we fail to prove that the primitive is hidden in one of the quadrants, we go to the next finer level of the pyramid for that quadrant and try again. Ultimately, we either prove that the entire polygon is hidden, or we recurse down to the finest level of the pyramid and find a visible pixel. If we find all visible pixels this way, we are performing scan conversion hierarchically.

A potential difficulty with the definitive visibility test is that it can be expensive to compute the closest Z value of the polygon in a quadrant. An alternative is to compare the value in the pyramid to the closest Z value of the entire polygon at each step of the recursion. With this modification, the test is faster and easier to implement, but no longer completely definitive. Ultimately, it will either prove that the entire polygon is hidden, or recurse down to the finest level of the pyramid and find a pixel it cannot prove is hidden. Our current implementation uses this technique. When the test fails to prove that a polygon is hidden, our implementation reverts to ordinary scan conversion to establish the visibility definitively.

3.3 Temporal coherence list

Frequently, when we render an image of a complex model using the object-space octree, only a small fraction of the octree cubes are visible. If we render the next frame in an animation, most of the cubes visible in the previous frame will probably still be visible. Some of the cubes visible in the last frame will become hidden and some cubes hidden in the last frame will become visible, but frame-to-frame coherence in most animations ensures that there will be relatively few changes in cube visibility for most frames (except scene changes and camera cuts). We exploit this fact in a very simple way with the hierarchical visibility algorithm. We maintain a list of the visible cubes from the previous frame, the *temporal coherence list*, and simply render all of the geometry on the list, marking the listed cubes as rendered, before commencing the usual algorithm. We then take the resulting Z buffer and use it to form the initial Z pyramid. If there is sufficient frame-to-frame coherence, most of the visible geometry will already be rendered, so the Z-pyramid test will be much more effective than when we start from scratch. The Z-pyramid test will be able to prove with less recursion that octree cubes and model polygons are hidden. As we will see in section IV, this can accelerate the rendering process substantially. After rendering the new frame, we update the temporal coherence list by checking each of the cubes on the list for visibility using the Z-pyramid test. This prevents the temporal coherence list from growing too large over time.

One way of thinking about the temporal coherence strategy is that we begin by guessing the final solution. If our guess is very close to the actual solution, the hierarchical visibility algorithm can use the Z pyramid to verify the portions of the guess which are correct much faster than it can construct them from scratch. Only the portions of the image that it cannot verify as being correct require further processing.

4 Implementation and Results

Our initial implementation of the hierarchical visibility algorithm is based on general purpose, portable C code and software scan conversion. This implementation uses the object-space octree, the image-space Z pyramid and the temporal coherence list. Even for relatively simple models the pure software algorithm is faster than traditional software Z buffering, and for complex models the acceleration can be very large.

In order to test the algorithm, we constructed an office module consisting of 15K polygons and then replicated the module in a three dimensional grid. Each module includes a stairway with a large open stairwell making it possible to see parts of the neighboring floors. None of the office walls extends to the ceiling, so from a high enough point in any of the cubicles, it is possible to see parts of most of the other cubicles on the same floor.

For simple models with low depth complexity, the hierarchical visibility method can be expected to take somewhat longer than traditional scan conversion due to the overhead of performing visibility tests on octree cubes and the cost of maintaining a Z pyramid. To measure the algorithm's overhead on simple models, we rendered a single office module consisting of 15K polygons at a viewpoint from which a high proportion of the model was visible. Rendering time for a 512 by 512 image was 1.52 seconds with the hierarchical visibility method and 1.30 seconds with traditional scan conversion, indicating a performance penalty of 17%. When we rendered three instances of the model (45K polygons), the running time was 3.05 seconds for both methods indicating that this level of complexity was the breakeven point for this particular model. Hierarchical visibility rendered nine instances of the same model (105K polygons) in 5.17 seconds, while traditional scan conversion took 7.16 seconds.

The chief value of the hierarchical visibility algorithm is, of course, for scenes of much higher complexity. To illustrate the point, we constructed a 33 by 33 by 33 replication of the office module which consists of 538 million polygons. The model is shown rendered in figure 1. 59.7 million polygons lie in the viewing frustum from this viewpoint, about one tenth of the entire model. Using the hierarchical visibility method, the Z-pyramid test was invoked on 1746 octree cubes and culled about 27% of the polygons in the viewing frustum.. The bounding boxes of 687 cubes were scan converted which culled nearly 73% of the model polygons in the viewing frustum, leaving only 83.0K polygons of which 41.2K were front facing (.000076 of the total model) to be scan converted in software. On an SGI Crimson Elan, the entire process took 6.45 seconds. Rendering this model using traditional Z buffering on the Crimson Elan hardware took approximately one hour and fifteen minutes. Rendering it in software on the Crimson would probably take days.

The center left panel of figure 1 shows the depth complexity processed by the algorithm for the image in the upper left. The depth complexity displayed in this image is the number of times each pixel was accessed in a box visibility test or in Z-buffer polygon scan conversion. Note the bright regions corresponding to portions of the image where it is possible to see far into the model; these are regions where the algorithm has to do the most work. In this image, the average depth complexity due to box scans is 7.23, and due to polygon scan-conversion is 2.48 for a total of 9.71. The maximum depth complexity is 124. Dividing the number of times the Z pyramid is accessed by the number of pixels on the screen lets us assign a value of .43 for the "depth complexity" of the Z-pyramid tests. Thus, the total average depth complexity of Z-pyramid tests, box scans and polygon scans is 10.14. Note that this is not the depth complexity of the model itself, but only the depth complexity of the hierarchical visibility computation. Computing the true depth complexity of the scene would require scan converting the entire model of 538 million polygons in software, which we have not done. In the lower left of figure 1, we show the viewing frustum and the octree subdivision. The two long strings of finely divided boxes correspond to the two brightest regions in the depth complexity image. Note that the algorithm is able to prove that large octree nodes in the distance are hidden. In the lower right, we show the Z pyramid for the scene. Even at fairly coarse resolutions, the Z pyramid contains a recognizable representation of the major occluders in the scene.

The office environment of figure 1 was chosen in part because it is a particularly difficult model for PVS methods. From every office cubicle in this environment, there are points from which almost every other cubicle on the same floor is visible. As a result, if the cubicles were used as cells in a PVS method, the potentially visible set for each cell would have to include nearly all the cells on its floor and many on other floors. Since each floor contains about 4 million polygons, the PVS methods would probably have to render many more polygons than the hierarchical method. In addition, the precomputation time for published PVS methods would be prohibitive for a model of this complexity. This model has 2000 times as many polygons as the model described by Teller and Sequin [15] which required 6 hours of pre-processing.

Admittedly, the replication of a single cell in the model means that it may not be a representative example, but it will be some time before people use models of this complexity without a great deal of instancing. The hierarchical visibility program we used for this example makes use of the replication in only two ways. First, the algorithm does not need to store half a billion polygons in main memory. Second, the algorithm only needs to consider a single cell in constructing the octree. These same simplifications would apply to any complex model using a great deal of instancing.

Figure 2 shows the hierarchical visibility method applied to an outdoor scene consisting of a terrain mesh with vegetation replicated on a two-dimensional grid. The model used for the lower left image consists of 53 million polygons, but only about 25K polygons are visible from this point of view. Most of the model is hidden by the hill or is outside the viewing frustum. The corresponding depth complexity image for hierarchical visibility computations is shown at the top left. The algorithm works hardest near the horizon where cube visibility is most difficult to establish. This frame took 7 seconds to render with software scan conversion on an SGI Crimson. In the lower right, we show a model consisting of 5 million polygons. Even though the model is simpler than the model in the lower left, the image is more complicated and took longer to render because a much larger fraction of the model is visible from this point of view. This image took 40 seconds to render with software scan conversion on an SGI Crimson. The average depth complexity for the scene is 7.27, but it reaches a peak of 85 in the bright areas of the depth complexity image in the upper right. These outdoor scenes have very different characteristics from the building interiors shown in figure 1 and are poorly suited to PVS methods because (a) very few of the polygons are axis-aligned and (b) the cell-to-cell visibility is not nearly as limited as in an architectural interior. Nonetheless, the hierarchical visibility algorithm continues to work effectively.

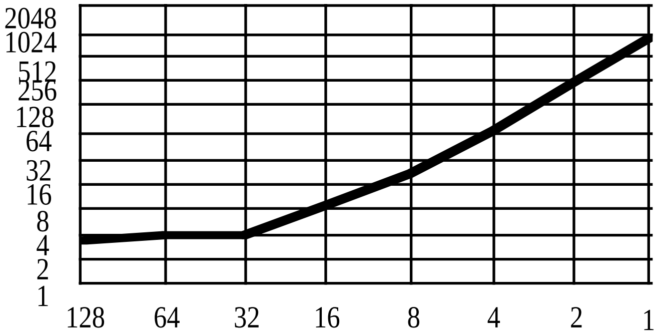


Fig. 3: Total time in seconds to render all windows as a unction of the number of pixels on the side of each window.

4.1 Parallelizability and Image-space coherence

We have made our hierarchical visibility implementation capable of dividing the image into a grid of smaller windows, rendering them individually and compositing them into a final image. The performance of the algorithm as the window size is varied tells us about the parallel performance of the algorithm and the extent to which it makes use of image-space coherence. If, like most ray tracers, the algorithm made no use of image-space coherence, we could render each pixel separately at no extra cost. Then it would be fully parallelizable. At the other extreme, if the algorithm made the best possible use of image-space coherence, it would render a sizeable region of pixels with only a small amount more computation than required to render a single pixel. Then it would be difficult to parallelize. Note that if we shrink the window size down to a single pixel, the hierarchical visibility algorithm becomes a ray caster using an octree subdivision.

Figure 3 graphs the rendering time for a frame from a walk-through of the model shown in figure 1 as a function of the window size. For window sizes from 32 by 32 on up, the curve is relatively flat, indicating that the algorithm should parallelize fairly well. For window sizes below 32 by 32, however, the slope of the curve indicates that the time to render a window is almost independent of the window size. The algorithm can, for example, render a 32 by 32 region for only slightly more than four times the computational expense of ray-casting a single pixel with this algorithm. Comparing the single pixel window time to the time for the whole image, we find that image-space coherence is responsible for a factor of almost 300 in running time for this example.

4.2 Use of graphics hardware

In addition to the pure software implementation, we have attempted to modify the algorithm to make the best possible use of available commercial hardware graphics accelerators. This raises some difficult challenges because the hierarchical visibility algorithm makes slightly different demands of scan-conversion hardware than traditional Z buffering. In particular, the use of octree object-space coherence depends on being able to determine quickly whether any pixel of a polygon would be visible if it were scan converted. Unfortunately, the commercial hardware graphics pipelines we have examined are either unable to answer this query at all, or take milliseconds to answer it. One would certainly expect some delay in getting information back from a graphics pipeline, but hardware designed with this type of query in mind should be able to return a result in microseconds rather than milliseconds.

We have implemented the object-space octree on a Kubota Pacific Titan 3000 workstation with Denali GB graphics hardware. The Denali supports an unusual graphics library call which determines whether or not any pixels in a set of polygons are visible given the current Z buffer. We use this “Z query” feature to determine the visibility of octree cubes. The cost of a Z query depends on the screen size of the cube, and it can take up to several milliseconds to determine whether or not a cube is visible. Our implementation makes no use of the Z pyramid because the cost of getting the required data to and from the Z buffer would exceed any possible savings. On a walk-through of a version of the office model with 1.9 million polygons, the Titan took an average of .54 seconds per frame to render 512 by 512 images. Because of the cost of doing the Z query, we only tested visibility of octree cubes containing at least eight hundred polygons. Even so, 36.5% of the running time was taken up by Z queries. If Z query were faster, we could use it effectively on octree cubes containing many fewer polygons and achieve substantial further acceleration. The Titan implementation has not been fully optimized for the Denali hardware and makes no use of temporal coherence, so these performance figures should be considered only suggestive of the machine's capabilities.

The other implementation we have that makes use of graphics hardware runs on SGI workstations. On these workstations, there is no way to inquire whether or not a polygon is visible without rendering it, so we use a hybrid hardware/software strategy. We do the first frame of a sequence entirely with software. On the second frame, we render everything on the temporal coherence list with the hardware pipeline. Then we read the image and the Z buffer from the hardware, form a Z pyramid and continue on in software. With this implementation, on the models we have tried, temporal coherence typically reduces the running time by a factor of between 1.5 and 2.

In the course of a walk-through of our office model, we rendered the frame in the upper left of figure 1 without temporal coherence, and then the next frame shown in the upper right of figure 1 using temporal coherence. The new polygons rendered in software are shown in magenta for illustration. For the most part, these are polygons that came into view as a result of panning the camera. The center right shows the depth complexity of the hierarchical computation for this frame. The image is much darker in most regions because the algorithm has much less work to do given the previous frame as a starting point. This temporal coherence frame took 3.96 seconds to render on a Crimson Elan, as compared with 6.45 seconds to render the same frame without temporal coherence.

Current graphics accelerators are not designed to support the rapid feedback from the pipeline needed to realize the full potential of octree culling in the hierarchical visibility algorithm. Hardware designed to take full advantage of the algorithm, however, could make it possible to interact very effectively with extremely complex environments as long as only a manageable number of the polygons are visible from any point of view. The octree subdivision, the Z pyramid and the temporal coherence strategy are all suitable for hardware implementation.

5 Conclusion

As more and more complex models become commonplace in computer graphics, it becomes increasingly important to exploit the available coherence in the visibility computation. Here we present an algorithm which combines the ability to profit from image-space coherence of Z-buffer scan conversion with the ability of ray tracing to avoid considering hidden geometry. It appears to be the first practical algorithm which materially profits from object-space, image-space and temporal coherence simultaneously. The algorithm has been tested and shown to work effectively on indoor and outdoor scenes with up to half a billion polygons.

The hierarchical visibility algorithm can make use of existing graphics accelerators without modification. Small changes in the design of graphics accelerators, however, would make a large difference in the performance of the algorithm. We hope that the appeal of this algorithm will induce hardware designers to alter future graphics hardware to facilitate hierarchical visibility computations.

Acknowledgements

We thank Frank Crow and the Advanced Technology Group at Apple Computer for supporting this research. We also thank Mike Toelle, Avi Bleiweiss, Helga Thorvaldsdottir and Mike Keller of Kubota Pacific Corporation for helping us test our algorithm on a Titan workstation.

References

[1] S. M. Rubin and T. Whitted, A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics,* 14(3): 110-116, July 1980.
[2] A Glassner, Space subdivision for fast ray tracing. *IEEE CG&A*, 4(10): 15-22, Oct. 1984.
[3] D. Jevans and B. Wyvill, Adaptive voxel subdivision for ray tracing. *Proc. Graphics Interface '89,* 164-172, June 1989.
[4] T. Kay and J. Kajiya. Ray tracing complex surfaces. *Computer Graphics,* 20:4: 269-278, Aug. 1986.
[5] M. Kaplan. The use of spatial coherence in ray tracing. In *Techniques for Computer Graphics, etc.*, D. Rogers and R. A. Earnshaw, Springer-Verlag, New York, 1987.
[6] H. Hubschman and S. W. Zucker, Frame to frame coherence and the hidden surface computation: constraints for a convex world, *ACM TOG*, 1(2): 129-162, April 1982.
[7] D. Jevans. Object space temporal coherence for ray tracing. *Proc. Graphics Interface '92*, Vancouver, B.C., 176-183, May 11-15, 1992.
[8] A. Glassner. Spacetime ray tracing for animation. *IEEE CG&A*, 8(3): 60-70, March 1988.
[9] J. Chapman, T. W. Calvert, and J. Dill, Spatio-temporal coherence in ray tracing. *Proceedings of Graphics Interface '90*, 190-204, 1990.
[10] S. Badt, Jr. Two algorithms for taking advantage of temporal coherence in ray tracing, *The Visual Computer,* 4: 123-132, 1988.
[11] B. Garlick, D. Baum, and J. Winget. Interactive viewing of large geometric databases using multiprocessor graphics workstation, *SIGGRAPH '90 Course Notes: Parallel Algorithms and Architectures for 3D Image Generation*, 1990.
[12] J. Airey, Increasing update rates in the building walkthrough system with automatic model-space subdivision, Technical Report TR90-027, The University of North Carolina at Chapel Hill, Department of Computer Science, 1990.
[13] J. Airey, J. Rohlf, and F. Brooks, Towards image realism with interactive update rates in complex virtual building environments, *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2): 41-50, 1990.
[14] S. Teller and C. Sequin, Visibility preprocessing for interactive walkthroughs, *Computer Graphics*, 25(4): 61-69, 1991.
[15] S. Teller and C. Sequin. Visibility computations in polyhedral three-dimensional environments, U.C. Berkeley Report No. UCB/CSD 92/680, April 1992.
[16] D. Meagher, Efficient synthetic image generation of arbitrary 3-D objects, *Proc.. IEEE Conf. on Pattern Recognition and Image Processing,* 473-478, June 1982.

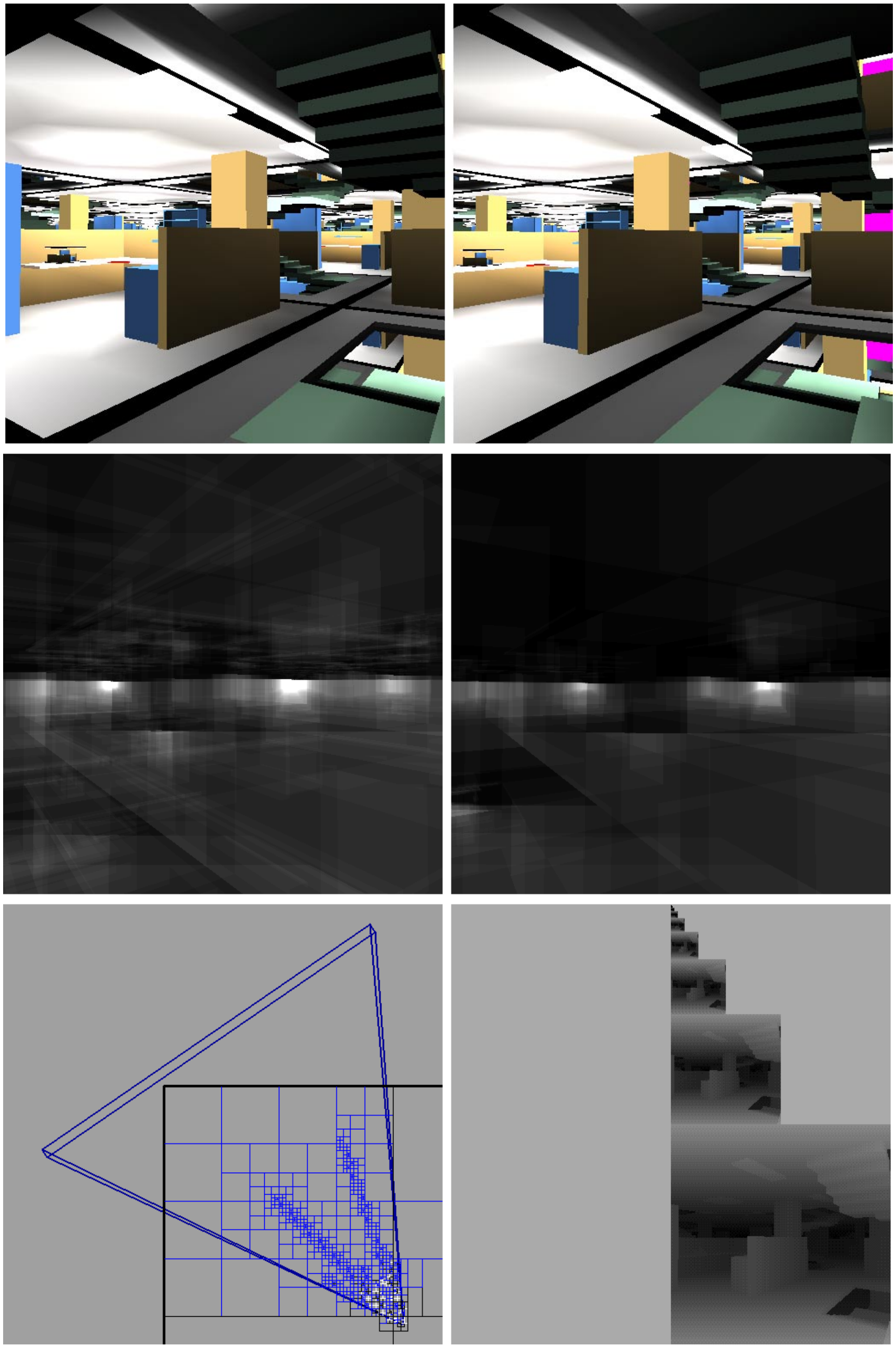


Figure 1: A 538 million polygon office environment rendered with hierarchical visibility. Upper left: Rendered image. Center left: Depth complexity of the hierarchical visibility computation. Lower Left: Viewing frustum and octree cubes examined while rendering the image in the upper left. Lower right: Z pyramid used to cull hidden geometry. Upper right: Image rendered with temporal coherence. Polygons not rendered in the previous frame are shown in magenta. Center right: Depth complexity of the hierarchical visibility computation for the frame rendered using temporal coherence.

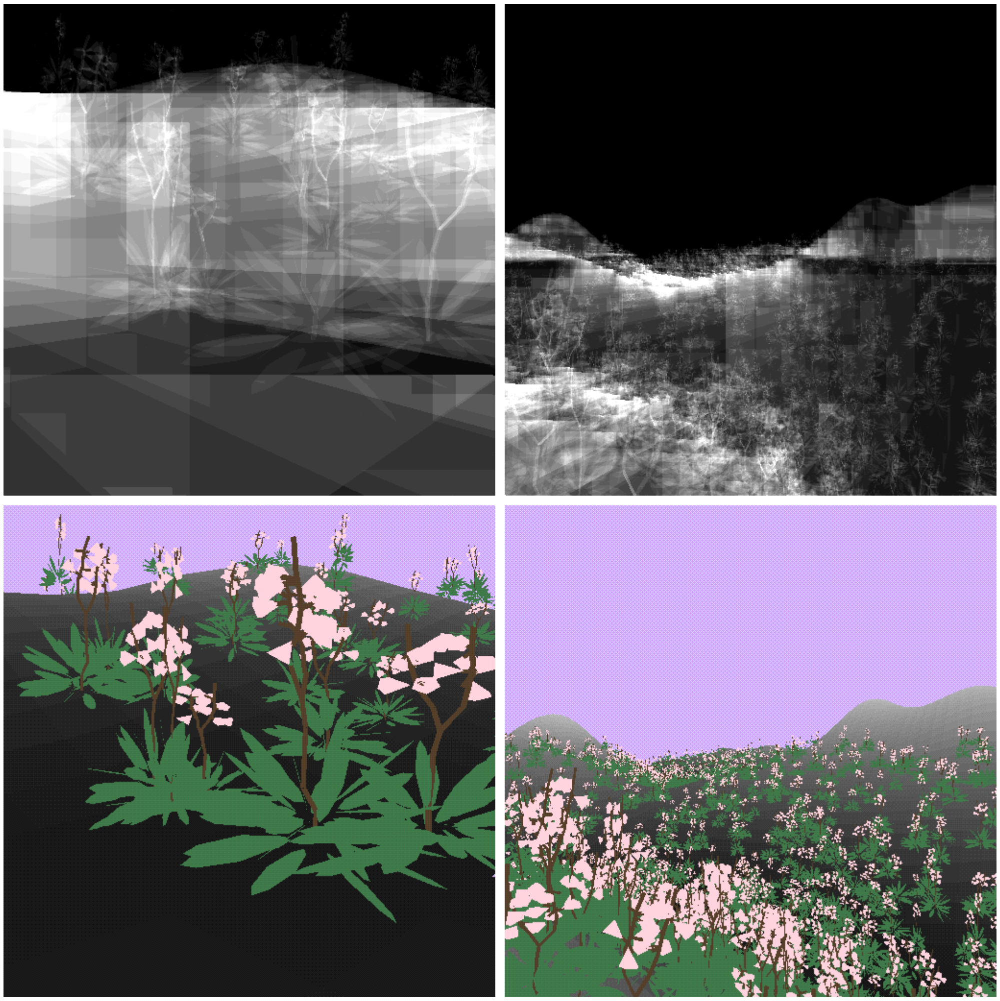


Figure 2: Lower left: Image of a 53 million polygon model (mostly hidden) rendered using hierarchical visibility. Upper left: Corresponding depth complexity for the hierarchical visibility computation. Lower right: Image of a 5 million polygon model. Upper right: Corresponding depth complexity for the hierarchical visibility computation.

Technical Note

A Quality Knob for Non-Conservative Culling with Hierarchical Z-Buffering

Ned Greene
NVIDIA

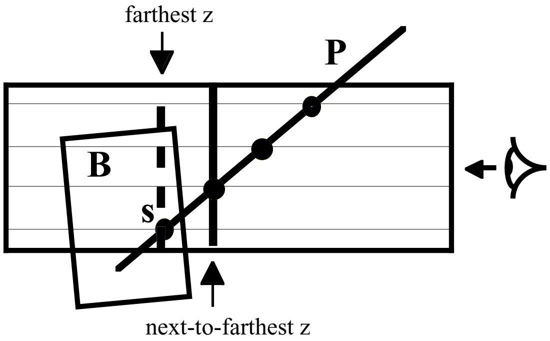
When rendering a deeply occluded scene with hierarchical z-buffering [GKM93] and the scene is too complex to render at the desired frame rate, there is a simple way to accelerate rendering by trading off image quality for rendering speed. This non-conservative culling method provides an error-bounded "quality knob" for hierarchical z-buffering.

With hierarchical z-buffering, z-buffer samples are maintained in an image pyramid called a z-pyramid, which is organized in NxN tiles (for example, 4x4 tiles). At all levels of a conventional z-pyramid, each z value is the farthest z in the corresponding NxN window of the adjacent finer level. Therefore, each z value represents the farthest z for a square region of the screen. This data structure enables efficient hierarchical culling of primitives and bounding boxes during hierarchical tiling, as described in [GKM93].

To maintain a conventional z-pyramid, whenever a z value is overwritten in an NxN tile in the z-buffer (i.e., the finest level of the z-pyramid), the farthest z in that tile is determined, and if this z value is nearer than the value stored for that tile in the next-to-the-finest pyramid level, that value is overwritten with the nearer value. This propagation procedure continues through the coarser levels of the pyramid until the existing entry in the pyramid is already as far away as the new z value.

Our method of non-conservative culling requires only a simple change to this propagation procedure. When a z-buffer sample is overwritten in an NxN tile in the z-buffer, instead of propagating that tile s farthest z value through the pyramid, we propagate the *Eth-to-the-farthest* z value, where E is an *error limit* which can be set to any value from zero to N^2-1. Propagation through coarser pyramid levels is done as with the original algorithm. With this simple change to propagation, the standard tiling procedure for hierarchical z-buffering (see GKM[93]) will cull primitives within regions of the screen where they are visible at E or fewer samples within any finest-level tile, and the output image will have E or fewer errors within any NxN screen tile. By an error at an image sample we mean that its color differs from a standard z-buffer image.

For example, the figure on the right shows a side view of a finest-level 4x4 tile which has been covered by a polygon P. Assuming that the error limit is one, the next-to-the-farthest z value will be propagated when P is processed, which will later result in culling bounding box B within the tile, even though B is visible at one image sample, labeled **s**.



This non-conservative culling method provides a simple quality knob that enables sacrificing image quality in exchange for faster rendering. If E is 0, a standard error-free z-buffer image is created, if E is 1 there will be at most one error in any NxN tile of samples in the output image, and so forth.

The method is particularly effective when the depth image of a scene contains numerous pinholes, because E or fewer pinholes within a finest-level tile will be plugged automatically. The method is also advantageous when applied to culling bounding boxes that are only "slightly visible," since this saves the work required to process all of the primitives they contain. Optionally, a separate z-pyramid (for levels coarser than the z-buffer) can be maintained for non-conservative culling of bounding boxes, so that tiling of primitives is unaffected.

In conclusion, the quality knob is a useful mechanism for a certain class of scenes, producing approximately correct images considerably faster than standard hierarchical z-buffering.

Reference: [GKM93] N. Greene, M. Kass, and G. Miller, “Hierarchical Z-Buffer Visibility," *Proc. of SIGGRAPH '93*, July 1993, 231-238.

OCCLUSION CULLING WITH OPTIMIZED HIERARCHICAL Z-BUFFERING

Ned Greene*

NVIDIA

Abstract

We introduce a method for occlusion culling tailored to z-buffer systems having rendering hardware that effectively reduces memory traffic in z and color values. The central innovation is to insert a *culling stage* into the pipeline that culls occluded geometry using a low-bandwidth variation of hierarchical z-buffering that performs conservative culling using only a small fraction of the bandwidth and storage of the original algorithm. Culling is performed with simple polygon-tiling operations that are well suited to implementation in hardware. On some scenes that we tested, culling reduced image memory traffic to less than standard z-buffering would generate when rendering just the polygons that are actually visible in the output image.

For deeply occluded scenes organized in bounding boxes, we enable efficient on-the-fly box culling on the host by providing the "tip" of the culling stage's z-pyramid, which requires only occasional low-bandwidth communication. Even for very complex scenes, the combination of front-to-back traversal of bounding boxes, box culling on the host, and an optimized culling stage can reduce image memory traffic more effectively than knowing in advance which polygons are visible, and rendering only them with standard z-buffering.

1 Introduction

The growing complexity of computer-animated scenes raises the relative importance of *occlusion culling*, which is the culling of occluded geometry prior to rasterization. In principle, accelerating z-buffer hardware with occlusion culling offers the prospect of rendering very deeply occluded scenes in real time, since only the visible parts of a scene need to be sent through the pipeline and rendered. In practice, however, existing culling algorithms have limitations, and none is capable of enabling real-time rendering of *arbitrary scenes*, a term that we will apply to polygonal scenes that have no particular occlusion relationships, may be deeply occluded, and may have high visible complexity. Some methods work effectively only on a limited class of models. For example, some methods fail to exploit *occluder fusion*, which is the collective occlusion of objects that overlap or abut on the screen. Other methods are completely general, but too slow to perform culling in real time. Moreover, culling algorithms generally make scene management much more complicated, for example, by requiring that the scene model be maintained in complex data structures or by incurring communication delays that hamper processing. Ideally, a culling method should work effectively on arbitrary polygonal scenes, efficiently cull occluded geometry using all available occluder fusion, and enable efficient on-the-fly scene management that is unimpeded by communication delays.

Although occlusion culling is usually thought of in the context of deeply occluded scenes, there is also ample opportunity for visibility operations to accelerate rendering of low-depth scenes. For example, standard z-buffering clears and reads the z-buffer at image samples before they are written for the first time. We show how these z reads can be avoided.

Our approach to devising a culling method that works automatically and effectively on virtually any polygonal model is to modify and optimize the *hierarchical visibility algorithm* [GKM93,Gre95]. This algorithm renders a scene that is organized in nested bounding boxes by traversing the boxes front to back, culling occluded boxes as they are encountered, and rendering the polygons in visible boxes with *hierarchical z-buffering*. This algorithm is very efficient because it needs to render only the polygons in visible boxes and hierarchical z-buffering efficiently processes individual polygons.

This article optimizations to the hierarchical visibility algorithm that facilitate integration with graphics hardware and reduce the algorithm's bandwidth and computation requirements. The key innovation is a novel variation of hierarchical z-buffering that requires only a small fraction of the bandwidth of the original algorithm to perform conservative culling. We add a separate *culling stage* to the pipeline that employs this algorithm to cull occluded geometry and passes visible and nearly visible image samples to rendering hardware. Tile records within a z-pyramid maintained by the culling stage include "znear" values, which usually enables image samples passed to the rendering stage to be identified as visible, in which case the z read that would normally be performed by standard z-buffering can be avoided.

For deeply occluded scenes organized in bounding boxes, we enable efficient on-the-fly box culling on the host by providing the "tip" of the culler's z-pyramid, which requires only occasional low-bandwidth communication. When we combine front-to-back traversal of bounding boxes, box culling on the host, and an optimized

* ned@ngreene.com

culling stage, culling works automatically and effectively on virtually any polygonal scene.

Our culling methods effectively reduce overall image traffic, which includes reads and writes of image and texture values in addition to z values. This can enhance performance substantially, since image bandwidth requirements are often the limiting bottleneck in today's z-buffer rendering systems [Mor00].

Perhaps the most fundamental goal of visibility research is to make rendering work proportional to a scene's *visible complexity* and independent of its overall complexity [Cla76]. To use as a benchmark in pursuing this goal, we introduce the term *oracle z-buffering*, the z-buffering performed by a perfect oracle that knows in advance which polygons are visible in the output image, and renders only them in front-to-back order. As an indication of the effectiveness of our optimized culling algorithm, simulations reported in section 6 show that it often generates fewer z reads and writes than oracle z-buffering, even for pathologically complex scenes such as the *Naked Empire* skyscraper model [Gre96].

2 Related Work

Our survey of work relating to occlusion culling will focus on methods that leverage z-buffer rendering hardware and consider their potential for real-time rendering of arbitrary polygonal scenes. For a general survey of occlusion-culling methods, see [CCS00].

One influential approach to occlusion culling, which was originally applied to architectural interiors [Air90,Tel92,Fun93]. is to precompute lists of visible geometry for volumetric cells. Recently developed projection methods that exploit occluder fusion make this approach much more general [Dur00], as does a method for fusing volumetric occluders [Sch00]. However, in addition to requiring extensive precomputation, this general approach requires maintaining large, complex data structures, it does not exploit occlusion of or by dynamic objects, and since much more is typically visible from a viewing volume than from a single viewpoint, it usually culls only a fraction of occluded geometry in any particular frame.

Another object-space method is to select foreground occluders and perform on-the-fly culling of geometry that they occlude [CT96,Hud97]. However, these methods do not effectively exploit occluder fusion, so good performance is limited to scenes having relatively simple visibility relationships.

The *hierarchical visibility algorithm* [GKM93, Gre95] maintains the scene model in an octree and the z-buffer in an image pyramid (a *z-pyramid*). A z-pyramid enables efficient hierarchical culling because it fully represents occluder fusion, but the polygon tiling required to maintain it cannot be done in real time in software. If a scene is traversed strictly front-to-back, software polygon tiling can be accelerated with coverage masks [Gre96], but the order requirement impairs practicality. Hierarchical visibility has precursors in the work of Warnock [War69], Clark [Cla76], and Meagher [Mea82], and has since been extended by Sudarsky and Gotsman [SG99], who focus on processing dynamic scenes, Xie and Shantz [XS99], who present optimizations for hardware implementation in tiled architectures, and by Zhang and Manocha et al. [Zha97,Zha98]. Naylor also uses image-space and object-space hierarchies to facilitate culling, projecting a scene represented with a 3D BSP tree into a 2D BSP tree representing the output image [Nay92], but this algorithm does not exploit occluder fusion, and as with [Gre96], it requires maintaining spatial hierarchies to enable traversal in depth order.

While descriptions have not been formally published, some flight-simulation hardware has employed two-level occlusion-image hierarchies to facilitate culling [Mue95], and ATI's *Radeon* graphics chip performs hierarchical z-buffering with a two-level z-pyramid [Mor00]. Although two-level hierarchies are an improvement over traditional rasterization, a full hierarchy culls large objects much more efficiently.

Aila and Miettinen have implemented a variation of hierarchical polygon tiling [Gre96] optimized for culling with silhouettes of foreground polyhedra that provides real-time software culling in some cases [Am00], but only for a limited class of scenes.

Zhang and Manocha et al. exploit z-buffer hardware by using it to render a “hierarchical occlusion map” of foreground occluders, which is then used for culling during a rendering pass [Zha97,Zha98]. Although this method improves performance in some cases, rendering in two passes adds complexity, effectiveness depends on being able to select efficient foreground occluders, and existing implementations do not produce standard z-buffer images.

To support culling of geometry contained in bounding boxes, some z-buffer accelerators report whether a portal or bounding box is visible [Tit93,Sco98]. A major problem with this approach is the delay between the time the host processor requests and receives box-visibility information, complicating scene management and impairing performance (for performance figures, see [KS01]).

Model simplification (e.g. [Hop96]) and geometric compression (e.g. [Dee94]) are complementary strategies that can be used in combination with occlusion culling to reduce computation and bandwidth requirements.

Some of the innovations discussed in this article were presented informally in [Gre99a] and [Gre99b].

3 The Proposed *Optimized Culling* Architecture

Memory traffic in depth and color values is typically the bottleneck limiting the performance of today's z-buffer accelerators [Mon00]. Our culling architecture reduces this image traffic by including a separate *culling stage* in the pipeline that performs *optimized hierarchical z-buffer-*

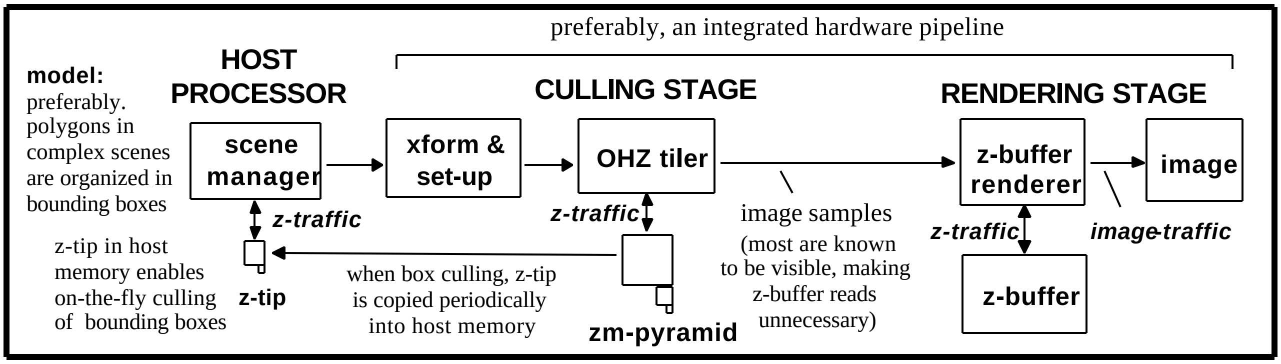

preferably, an integrated hardware pipeline
model:
preferably. polygons in complex scenes are organized in bounding boxes
HOST PROCESSOR
CULLING STAGE
RENDERING STAGE
scene manager
xform & set-up
OHZ tiler
z-buffer renderer
image
z-traffic
z-traffic
image samples
(most are known to be visible, making z-buffer reads unnecessary)
z-traffic
image-traffic
z-tip in host memory enables on-the-fly culling of bounding boxes
z-tip
when box culling, z-tip is copied periodically into host memory
zm-pyramid
z-buffer

Figure 1 The proposed architecture for *optimized culling*.

Culling is performed with a separate culling stage that tiles primitives with *optimized hierarchical z-buffering*(OHZ) and maintains its own compact occlusion image in a *zm-pyramid*. The zm-pyramid maintains *znear*values for image tiles, which enables most image samples passed to the rendering stage to be identified as visible, thereby eliminating most z-buffer reads. On-the-fly culling of bounding boxes on the host is supported by copying the ÒtipÓ of the zm-pyramid into host memory.

ing, a novel variation of hierarchical z-buffering [GKM93] that requires only a small fraction of the bandwidth and storage needed by the original algorithm to perform conservative culling. These savings are made possible by a data structure called a *zm-pyramid*, which represents occlusion information about NxN-sample screen tiles as a coverage mask and two pairs of *znear* and *zfar* values. Maintaining the occlusion image in a pyramid enables efficient hierarchical culling of occluded geometry within the culling stage, and also on the host, if the "tip" of the zm-pyramid is copied into host memory.

Figure 1 shows the proposed architecture for *optimized culling*, in which a host processor performs scene management and sends polygons to a transformation and set-up module, which in turn sends transformed polygons to a culling stage, which culls occluded geometry and passes visible and nearly visible image samples (or polygons) to a z-buffer rendering stage.

Preferably, transformation, culling, and rendering are all part of an integrated hardware pipeline, in which case the culling stage sends records for image samples to the rendering stage, each identified as *accepted*, meaning known to be visible (so the rendering stage can avoid reading z), or *ambiguous*, meaning the sample may or may not be visible.

Alternatively, if the culling stage is not integrated with the rendering pipeline (as would be the case, for example, if it were integrated with a memory controller that feeds a graphics card) it outputs polygon records, each tagged as *accepted* (meaning all samples on the polygon are visible) or *ambiguous*.

4 Optimized Hierarchical Z-Buffering

The culling stage employs *optimized hierarchical z-buffering* (OHZ) to minimize the bandwidth requirements of conservative culling. OHZ uses a *zm-pyramid*, which is similar to a conventional z-pyramid [GKM93] that is missing its finest level. For example, the skyscraper image of figure 8 has resolution 1024x1024, and the finest level in the corresponding zm-pyramid has resolution 256x256, as shown in figure 2.

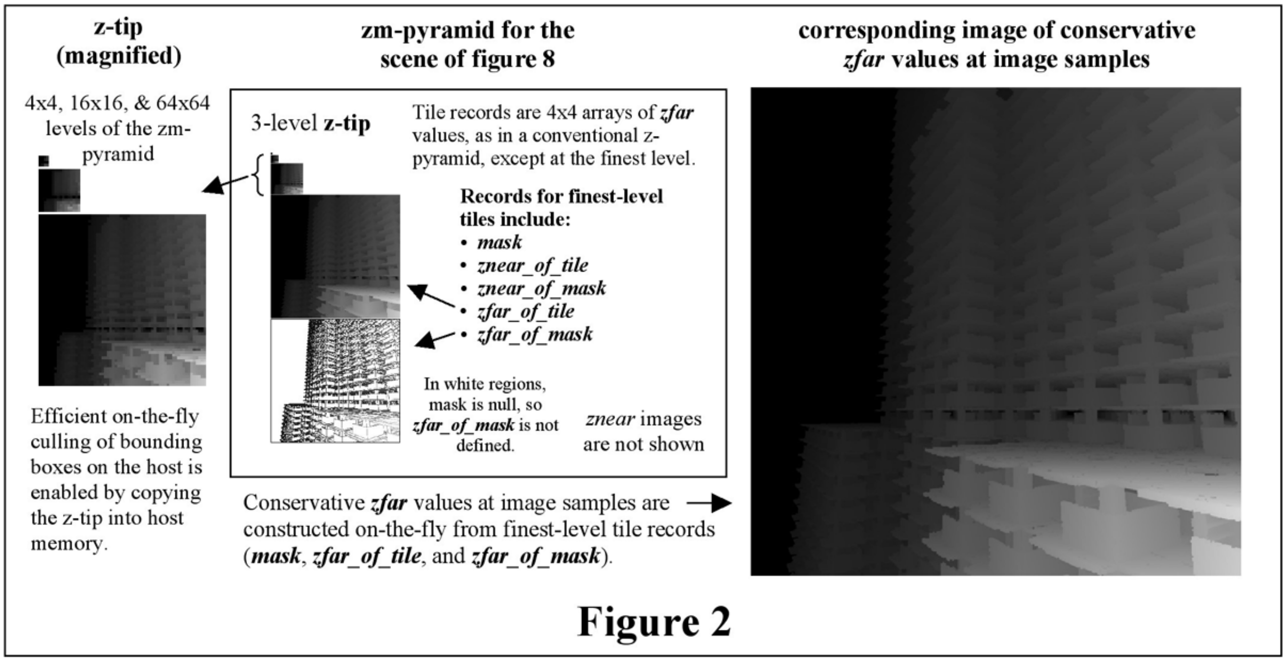
For each entry in the finest level of a zm-pyramid, we store a coverage mask, which defines two regions of a tile (unless the mask is null), and a *znear* and a *zfar* value for each of the regions. A region's *znear* and *zfar* values are the nearest and farthest depth of any potentially visible sample on polygons processed thus far which overlap the region. All other levels of the zm-pyramid are arrays of z values, as in a conventional z-pyramid.

To reduce storage and bandwidth, all z values in the zm-pyramid are stored at low-precision, for example, in 8 or 12 bits. A zm-pyramid with 4x4 decimation and 12-bit z values requires approximately 3.75 bits per image sample, so storage requirements are approximately 1/6 of a 24-bit z-buffer. The A-buffer [Car84] also uses coverage masks to expedite processing within image tiles, but the data structure that we use is much more compact and specifically adapted for conservative culling.

Tiling Procedure

The OHZ tiling procedure is entirely analogous to conventional hierarchical z-buffering as described in [GKM93], except for the way that finest-level tiles are processed. Refer to this article for a discussion of the original tiling algorithm.

Briefly, hierarchical tiling of a polygon begins at the coarsest enclosing NxN tile in the zm-pyramid. Within each tile, overlap and depth tests identify cells within which the polygon may be visible, which are recursively subdivided. If subdivision reaches a finest-level tile in the zm-pyramid, the tile's mask and z values establish the z range of previously rendered polygons at each image sample, enabling incoming samples to be classified as



Efficient on-the-fly culling of bounding boxes on the host is enabled by copying the z-tip into host memory.

Conservative ***zfar*** values at image samples are constructed on-the-fly from finest-level tile records (***mask***, ***zfar_of_tile***, and ***zfar_of_mask***).

Figure 2

culled, ambiguous, or *accepted*, as will be discussed in more detail later.

At the beginning of a frame we set a "virgin tile" flag in each finest-level tile of the zm-pyramid and clear the z values in all other levels of the pyramid to the far clipping plane. A tile record's virgin tile flag in combination with the mask and *zfar* values always indicate which, if any, image samples are at this cleared z value, so it is not necessary to clear the z-buffer at the beginning of a frame, saving considerable bandwidth.

Tiling Example

The OHZ procedure for tiling a polygon is analogous to conventional hierarchical z-buffering [GKM93], except for the processing of finest-level tiles, which is illustrated in figure 3, where *zfar_of_tile* indicates the farthest visible z for the whole tile, a coverage *mask* indicates samples that have been covered by one or more polygons since *zfar_of_tile* was established, and *zfar_of_mask* is the farthest z value of these samples. *Znear_of_mask* is the nearest z value of the samples covered by the mask and *znear_of_tile* is the nearest z value of the samples not covered by the mask.

Suppose that *zfar_of_tile* and *znear_of_tile* have been established before processing polygons P1 and P2 (e.g., by polygon P0). When P1 is encountered, the mask of visible samples that it covers (s1 and s2) is created (labeled *mask*), *zfar_of_mask* is set to the *zfar* value of these samples, and *znear_of_mask* is set to the nearest z of the samples (since this is nearer than *znear_of_tile*). Since sample s1 on P1 is in front of *znear_of_tile*, it is known to be visible with respect to the rendering stage's z-buffer, and is tagged *accepted*, which informs the rendering stage that there is no need to read the z-buffer. Sample s2, however, is farther than *znear_of_tile*, so it may or may not be visible, and it is tagged as *ambiguous*, indicating that it will be necessary to read the z-buffer and make a depth comparison at this sample.

Later, when polygon P2 is processed, since it covers the tile collectively with the stored *mask*, a new *zfar* value is established for the tile, which is written to *zfar_of_tile* (this is the old value for *zfar_of_mask*). The tile's *mask* is set to P2's *mask* and *zfar_of_mask* is set to P2's *zfar* value. *Znear* values are also updated, with *znear_of_tile* being set to the old value of *znear_of_mask*, and *znear_of_mask* being set to the depth of the nearest sample on P2.

In all, there are five cases that need to be considered when updating *zfar* values in tile records, which are diagrammed in figure 4, where dashed lines in the upper left diagram indicate whether the *zfar* value of the visible samples covered by the polygon is nearer or farther than *zfar_of_mask* and whether the polygon's visible samples cover the tile in combination with the stored *mask*. The example of figure 3 corresponds to case C4. Although not

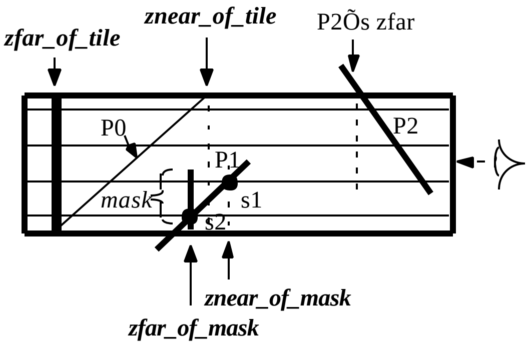


Figure 3

Schematic side view of a finest-level 4x4 tile in the zm-pyramid.

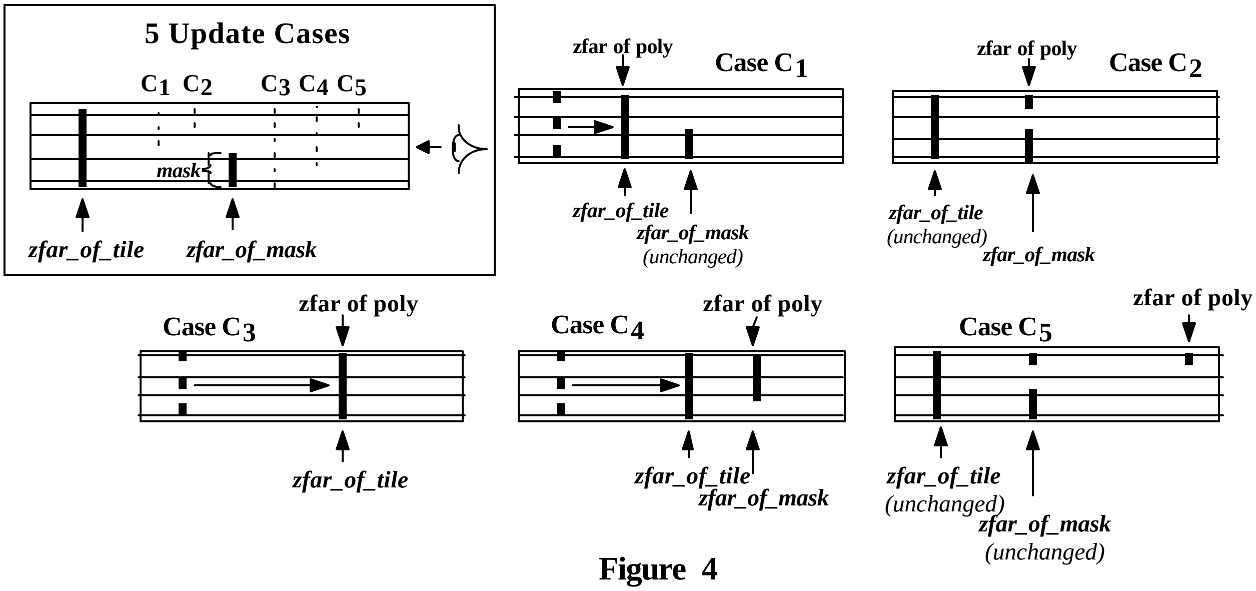


Figure 4

Schematic illustration of the five different cases for updating the mask and *zfar* values in a tile record.

illustrated in figure 4, *znear* values also need to be updated using that rule that they must indicate the nearest depth encountered so far for the region of the screen corresponding to the mask, in the case of *znear_of_mask*, and the non-mask region, in the case of *znear_of_tile*.

Despite its compact size compared with a z-buffer, a zm-pyramid culls efficiently, since the coverage mask and two *zfar* values stored in finest-level records encode conservative sample-by-sample depth values that usually closely approximate the real z-buffer values. This enables effective culling, even within screen tiles where z values vary greatly, as may occur when a silhouette edge crosses a tile, or when a tile is covered by a finely tessellated object. The right panel of figure 2 shows sample-by-sample z values constructed from the zm-pyramid for figure 8.

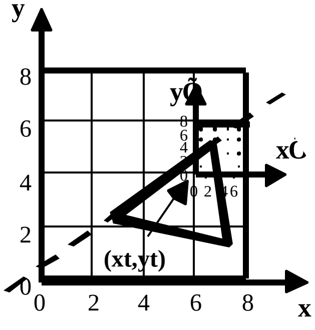
Typically, zm-pyramids cull most occluded samples (often, about 90%) and "accept" most visible samples (often, well over 90%). In combination with avoiding z-buffer clears, this saves a great deal of memory traffic in z and color values, while reading and writing the zm-pyramid consumes only a small fraction of the savings. Culling early in the pipeline also eliminates texture fetches for culled samples, providing additional bandwidth savings.

Preferably, the culling stage maintains the zm-pyramid using depth information it generates while tiling primitives, rather than by propagating z values from the z-buffer, as with conventional hierarchical z-buffering [GKM93]. With this approach, the zm-pyramid is always up to date,

Efficient Hardware Implementation of OHZ

OHZ is well suited to hardware implementation because recursive subdivision can be implemented with a stack of tile records and the linear equations describing a polygon can be evaluated without general-purpose multiplication using the following novel hierarchical method, which is conceptually similar to Fuch's method of evaluating equations on a pixel grid [Fuc85].

As diagrammed in figure 5, within zm-pyramid tiles it is necessary to evaluate the polygon's edge equations (form: $A_e x + B_e y + C_e$), and depth equation, $z = A_d x + B_d y + C_d$. In set-up computations, coefficients of the edge and depth equations are computed relative to the coordinate frame (scaled as shown) of the smallest enclosing z-pyramid tile, which is where tiling begins. When the tile is subdivided, the edge and depth equations are trans-



Transformation Formulas

Depth Equation

parent tile: $z = A_d x + B_d y + C_d$

child tile: $z = A_d' x' + B_d' y' + C_d'$

where $A_d' = A_d/N$

$B_d' = B_d/N$

$C_d' = A_d xt + B_d yt + C_d$

Edge Equation

parent tile: $A_e x + B_e y + C_e = 0$

child tile: $A_e x' + B_e y' + C_e' = 0$

where $C_e' = N(A_e xt + B_e yt + C_e)$

Figure 5

Formulas for hierarchical computation of depth and edge equations.

formed to the child tile's coordinate frame using the formulas stated in figure 5. Since term Axt + Byt + C (where (xt,yt) is the origin of the child tile) has already been computed at the parent tile and N is a power of 2, evaluating these formulas requires only shifting and addition. Note that it is necessary to evaluate each edge and depth equation at only one corner of each cell. Depth equations are evaluated at the corner where the polygon's plane is nearest, which corresponds to the quadrant of the screen projection of a backfacing normal to the polygon. Similarly, the "normal" to an edge indicates which corner of a cell should be substituted into the edge's equation. This hierarchical evaluation method can also be applied to Gouraud interpolation and interpolation of texture coordinates (and in fact, any polynomial equation in N dimensions), and it is compatible with jitter if jittering is restricted to samples on an "oversize" integer grid within a pixel, say a 32x32 grid.

A hardware culling stage can substantially reduce texture bandwidth requirements if, for texture-mapped polygons that are not culled, it computes a bit map indicating which rectangular tiles on a coarse grid registered with the texture map are "visible," thereby indicating which texture values need to be made available to the renderer. Visibility is established by simply tiling the corresponding rectangle with OHZ. The same approach can be used to ignore regions of alpha-mapped polygons that are transparent, allowing culling of polygons that are only "visible" in transparent regions.

5 Culling Bounding Boxes with the Z-Tip

In deeply occluded scenes, traffic in occluded polygons can be the bottleneck limiting overall performance. This problem can be overcome by organizing the scene in bounding boxes, and while traversing boxes front to back, culling occluded boxes and sending the polygons in visible boxes to be rendered [GKM93]. This procedure has ideal box-culling performance in the sense that only polygons contained in boxes that are actually visible in the output image are processed.

It is straightforward to achieve this with software rendering, but when rendering with graphics hardware, the host does not have fast access to an occlusion image. Although some z-buffer hardware supports box culling [Tit93,Sco98], there is a delay between when the processor issues a visibility query and when it receives a reply, and in the meantime, it is not known if the box's contents need to be processed, making efficient scene management problematic. For example, Klosowski and Silva [KS01] report that visibility queries take between .1 and 1 milliseconds on an HP Kayak fx6 [Sco98]. Significant delays are fundamental because pipeline queues often contain numerous polygons, and if boxes wait their turn in queues, delays are long, and if queues are skipped over, culling efficiency is impaired. Consequently, many hardware designers do not consider traditional hardware-assisted box culling to be a very useful feature [Tar99].

Our approach to enabling the host to cull boxes on-the-fly without incurring communication delays is for the culling stage to periodically copy the "tip" of the zm-pyramid (a *z-tip*) into host memory with DMA. This enables the host to cull bounding boxes as they are encountered. Simulations reported in the following section show that copying 3-level z-tips (i.e. the 4x4, 16x16, and 64x64 levels) a dozen or two times a frame enables the host to cull a high fraction of boxes that are actually occluded, despite the fact that the z-tips are slightly out of date. Since z-tips require only a few kilobytes of storage, copying z-tips consumes little bandwidth. In short, only coarse-grained, low-bandwidth, asynchronous communication is required to support this culling method.

Note that whenever the farthest z value in the coarsest level changes in a z-tip copied to the host, this advances the scene's far clipping plane, and software that culls boxes to the view frustum [Cla76,GBW90] should exploit this.

Frame-Coherent Box Culling

One shortcoming of the box-culling method described above is that the host must work in unison with the graphics hardware on the current frame, which prevents it from queuing up geometry in order to even out its workload and obtain smoother animation. In today's systems, the host commonly works a frame or more ahead of the hardware.

This problem can be addressed with a method we call *frame-coherent box culling*, which uses z-tips in combination with the frame-coherence variation of the hierarchical visibility algorithm [GKM93]. This two-pass method permits the host to work ahead of graphics hardware most of the time.

Suppose we'd like the host to work two or three frames ahead of the hardware. To render frame F, in a first pass the host traverses the scene's bounding boxes in front-to-back order, and as each box is encountered, transforms it to frame F-3's coordinate frame and determines whether it is visible with respect to frame F-3's z-tip, which was previously stored. If so, the polygons contained in the box are added to the "pass-one geometry queue" for frame F that will be rendered later. When all boxes have been traversed, the host is done with pass one, and it starts another task.

The graphics hardware starts rendering frame F's pass-one geometry when it is done with the preceding frame, and when it finishes this rendering it copies the zm-pyramid's z-tip into host memory and notifies the host that it is ready for pass two. Typically, box visibility is highly coherent from frame to frame [GKM93], so the frame is nearly complete after rendering pass-one geometry, and pass two requires relatively little time.

During pass two, the host works in unison with the graphics hardware to complete frame F. The host again traverses frame F's bounding boxes in front-to-back order, skipping boxes that have already been processed, testing the remaining boxes for visibility against the current frame's z-tip, and sending the primitives in visible boxes to the hardware, thereby completing the frame.

The advantage of frame-coherent box culling is that the only time that the host needs to work in unison with the graphics hardware is during pass two, which is typically a small fraction of the time. Most of the time the host can be working one or more frames ahead, queuing up pass-one geometry.

In unusual situations when frame coherence is low (for example, at a camera cut), a complementary approach can be used to enable the host to work ahead of the hardware. With this method we create a low-resolution (e.g.32x32) depth image for the frame that the host wants to start working on, using a highly simplified model of the scene's major occluders. Then, the host uses this occlusion image to determine boxes that are likely to be visible and adds the polygons they contain to the appropriate geometry queue. As with frame-coherent box culling, the scene is later completed in a second pass in which the host works in unison with the hardware, using copied z-tips to cull occluded bounding boxes.

6 Simulation Results

Our discussion of culling performance will focus on image-bandwidth consumption, because traffic in depth and color values is commonly the bottleneck limiting the performance of today's z-buffer accelerators [Mon00]. We will compare the image bandwidth requirements of our optimized culling architecture, which we'll abbreviate *optimized_culling*, to those of standard z-buffering and also *oracle z-buffering*. Comparison with oracle z-buffering offers a good indication of how close we come to optimal culling performance, although its bandwidth requirements are not a strict lower limit on those of actual systems, which may employ z compression, fast z clear, and other methods to reduce bandwidth [Mor00].

In order to characterize the image bandwidth requirements of z-buffering (both standard and hierarchical), we will use the term *z-traffic* to refer to the total number of z reads and writes that are generated in producing a frame, and the term *z+color-traffic* to refer to the sum of *z-traffic* and the total number of writes to the output image, including the initial write at each sample to clear the image. The term *average z-traffic* will refer to the average number of z accesses per image sample in the output image, and *average z+ color-traffic* is defined analogously.

We implemented a high-level software simulator programmed in C to measure the performance of *optimized_culling*. The culling stage performed optimized hierarchical z-buffering into a zm-pyramid with 4x4 decimation (as in figure 2), hierarchically culling occluded geometry and sending *ambiguous* and *accepted* samples to the rendering stage, where ordinary z-buffering was performed, except that z reads were skipped for *accepted* samples. The figures included here are for point-sampled 1024x1024 images where image samples are pixel centers. The relative performance of *optimized_culling* is better when oversampling (which requires using a zm-pyramid having an additional level), since occlusion within finest-level tiles is more coherent.

For the simulated scenes we measured *average z-traffic* and *average z+color-traffic*. As summarized in Table 1, we compared these traffic figures for *optimized_culling* to that of oracle z-buffering and standard z-buffering. For oracle z-buffering, figures for *z-traffic and z+color-traffic* are those generated by front-to-back z-buffering of just the polygons that are visible in the output image. For standard z-buffering these figures apply to z-buffering all polygons in scene order. For both oracle z-buffering and standard z-buffering we counted one z write and one color write per pixel for clearing.

For *optimized_culling, z-traffic* is the sum of z-buffer reads and writes in the rendering stage plus zm-pyramid accesses in the culling stage (and z-tip reads by the host, when box culling), which consists of reads and writes of *znear*, *zfar*, and *mask* values in tile records plus all reads and writes of z values at coarser levels of the pyramid when performing depth comparisons and propagating z values. We counted one write per 4x4 tile for clearing the virgin tile flag, as explained in section 4. Average *z+color_traffic* for *optimized_culling* is average *z-traffic* plus one write each time the image buffer is updated, including one write for the initial clear.

Winbench Computer Game

To evaluate performance on typical computer-game scenes, we simulated *optimized_culling* performance on several 3D Winbench 2000 scenes, including a frame from the *Canyon* sequence (figure 6). As listed in Table 1, *Canyon* has 11,025 front-facing polygons (excluding text billboards, which we omitted), their average depth is 2.55, and the average depth of polygons that are actually visible in the output image (panel b) is 1.46. Image resolution was 1024x768 and we used 8-bit z values in the zm-pyramid.

Figure 6c shows pixel-by-pixel z-buffer traffic (the sum of reads and writes), which averaged 1.11 per pixel. Z-buffer traffic is 1 at 91% of pixels (blue region), indicating that the first polygon processed at a pixel was *accepted* (generating a z write but no read), and all other polygons arriving at that pixel were culled, indicating that culling performed by the zm-pyramid was ideal at these pixels. Of pixels sent to the renderer, 97% were accepted and 3% where ambiguous, indicating that z-buffer reads needed to be performed only 3% of the time. In other words, the zm-pyramid definitively established visibility at 97% of samples, despite its compact size, approximately 1/8 of the z-buffer in this simulation. Within the culling stage, accesses to the zm-pyramid averaged .58 per pixel when amortized over the corresponding screen area. Total aver-

		versus standard z-buffering		versus oracle z-buffering						scene statistics		
TABLE 1 Simulation	avg. *z-traffic* *optimized_culling* total = z-buffer + zm-pyramid	% *z-traffic* of standard ZB	% *z+color-traffic* of standard ZB	% *z-traffic* of oracle ZB	%*z+color-traffic* of oracle ZB	% of samples ÒacceptedÓ*	% pixels where z-traffic is 1	% of polygons culled	bits Z precision in zm-pyramid	scene depth	depth vis. polys	no. of polygons
Canyon, trace order (fig 6c)	1.69 = 1.11 + .58	37%	56%	49%	69%	97%	91%	84%	8	2.55	1.46	11k
Canyon, reverse order (fig 6d)	3.58 = 2.53 +1.05	64%	77%	103%	121%	77%	25%	58%	Ò	Ò	Ò	Ò
Lightscape Interior (fig 7)	3.13 = 1.53 + 1.60	72%	82%	103%	110%	89%	73%	65%	12	1.98	1.03	173k
Fig 8: box culling with z-tips	3.58 = 2.53 + 1.05	?	?	109%	?	87%	31%	-	16	23.4	1.86	988k
Fig 8: frame-coh. box culling	3.12 = 1.74 + 1.38	?	?	81%	?	88%	25%	-	Ò	Ò	Ò	Ò

* percentage of samples sent to the renderer that are accepted

age *z-traffic* for *optimized_culling* was 1.69, which is 49% of the figure for oracle z-buffering and 37% of the figure for standard z-buffering. As this comparison shows, *z-traffic* was reduced to half of that which would have been generated if we knew in advance which polygons were visible and rendered only them in front-to-back order with standard z-buffering. *z+color-traffic* for *optimized_culling* was 69% of the figure for oracle z-buffering and 56% of the figure for standard z-buffering.

One reason that *optimized_culling's z-traffic* is so low is that the polygons in this test happen to be stored in roughly front-to-back order. To quantify the impact of traversal order, we reversed the order of the polygons and found that average *z-traffic* rose to 3.58, or 103% of oracle z-buffering and 64% of standard z-buffering. Figure 6d shows pixel-by-pixel z-buffer traffic for *optimized_culling*, which averaged 2.53. Even when traversal order is unfavorable, as it was in this simulation, *optimized_culling* generally performs much better than standard z-buffering because most pixels are accepted (77% in this example), thereby avoiding z reads required by standard z-buffering.

Overall, our tests on this and other computer-game traces showed that *optimized_culling* reduced *z-traffic* effectively, often well below that of oracle z-buffering. This often occurred even with traces that had already benefited from portals culling within a game application, because numerous occluded polygons remained. In the games that we tested, most translucent polygons wrote z values, so they did not impair culling efficiency, and clipping plane values permitted good culling efficiency with 8-bit z values in the zm-pyramid.

An Architectural Interior

To measure *optimized_culling's* performance on highly tessellated models, we rendered the *Lightscape Interior* of figure 7 (1024x768), which has 173,000 polygons and an average depth of 1.98. 89% of samples were accepted and z-buffer traffic was one at 73% of image samples, those shown in blue in figure 7d. Thus, the zm-pyramid efficiently culled and accepted samples, despite the extreme tessellation. If the culling stage were operating in "polygon culling" mode it would have culled 65% of polygons, the ones outlined in black in panel c. (The corresponding figures for the forward and reverse *Canyon* simulations are 84% and 58%.) Overall *z-traffic* for *optimized_culling* was 103% of oracle z-buffering and 72% of standard z-buffering. The corresponding figures for a reverse-order *Lightscape* simulation were 98% of oracle z-buffering and 64% of standard z-buffering.

Skyscraper Model

To test *optimized_culling's* performance on deeply occluded scenes organized in bounding boxes, we animated the skyscraper model of figure 8 (1024x1024) using the motion parameters of the *Naked Empire* animation [Gre96]. In this simplified variation of the publicly available model, each cubic bounding box contains two office modules having a total of 316 rectangles, organized for backface culling. These boxes were organized in an octree, but within each box there was no further organization into smaller boxes. Polygons were stored in no particular order within each box, and to prevent polygon order from distorting culling measurements, we reversed the order that polygons were traversed every other time a box was processed. The culling stage tiled polygons one-by-one into the zm-pyramid without exploiting the fact that some abutting polygons form larger polygons.

Average scene depth is 23.4, average depth of visible polygons is 1.86, and 988,000 polygons (including back-facing ones) in 3126 bounding boxes lie inside the view frustum. As is apparent in figure 9, it is possible to see completely through the model in places.

We tested the novel variations of the hierarchical visibility algorithm [GKM93] described in section 5, which traverse the octree front to back, use z-tips to cull occluded boxes, and render the polygons in visible boxes. First we simulated conditions when the host and graphics hardware work in unison. While the host traversed the scene's boxes front-to-back and sent polygons in visible boxes to be rendered, the culling stage periodically copied a 3-level z-tip (the 4x4, 16x16, and 64x64 levels of the zm-pyramid) into host memory, which was done 16 times in the course of rendering figure 8. Total average *z-traffic* was 3.58 (2.53 in the z-buffer, 1.05 in the zm-pyramid and z-tip), which was 109% of the figure for oracle z-buffering.

Next we simulated *frame-coherent box culling*, beginning figure 8 by rendering the polygons in bounding boxes that were visible three frames back (frame "F-3"). To do this, the host traversed the octree in front-to-back order, culled occluded boxes with frame F-3's z-tip (which had been previously stored in host memory), and sent the polygons in visible boxes to be rendered. Frame F-3's z-tip is shown magnified in the left-hand panel of figure 2 and at actual scale at upper left in figure 8. Then, in a second front-to-back traversal of the octree, the host used z-tips of the current frame (copied by the culler into host memory) to identify the visible boxes that remained to be rendered. In this particular frame, the first pass produced a nearly complete image, with only 404 pixels overwritten in the second pass. This indicates that most of work the host has to do was on the first pass, and this work can be queued up two frames ahead of the current frame, since frame's F-3's final z-tip is available then. Figure 11 is pixel-by-pixel z-buffer traffic for this simulation, wherein z-buffer traffic is one at 25% of covered pixels and two at 28% of covered pixels, indicating that no z reads were performed at 53% of pixels. Total average *z-traffic* was 3.12 (1.74 in the z-buffer,1.38 in the zm-pyramid and z-tip), which was 81% of the figure for oracle z-buffering. Average *z+color-traffic* for *optimized_culling* was nearly the same as the figure for oracle z-buffering. Thus, even for this very complex and deeply occluded scene, *optimized_culling* reduced image memory traffic as effectively as knowing in advance which polygons would be visible and rendering only them.

Since this scene has high frame coherence, there is no need to use the complementary method of identifying boxes that are likely to be visible by rendering a low-resolution image of major occluders and performing box-visibility tests using this image's z-tip, as discussed in section 5. But to illustrate the method, we rendered a simple polyhedral model of the skyscraper at 64x64 (lower half of figure 12) and used its depth image (upper half of image 12) to test boxes for visibility, which identified the frontmost parts of the skyscraper as visible. In general, this method enables the host to identify geometry that is likely to be visible in any frame for which the camera transformation is known (or can be approximated), allowing it to work ahead of graphics hardware and queue up geometry to be rendered.

"Topiary Tower"

Given bounding boxes and favorable traversal order, *optimized_culling* can efficiently cull virtually any polygonal model, because a zm-pyramid represents nearly all of the occluder fusion of previously processed polygons. To illustrate that efficiency does not depend on construction from abutting polygons, as occurs in the skyscraper model, we constructed
Topiary Tower (figure 13) by stacking topiary balls within the skyscraper framework, one of which is shown sliced in half. Even though individual balls make poor occluders, collectively they occlude very effectively when culling with a zm-pyramid. For figure 13, *z-traffic* generated by *optimized_culling* was approximately 150% of the figure for oracle z-buffering.

Note that for this model is highly problematic for methods that precompute visibility for regions of space, even those that are able to fuse occluders [Dur00], because a great deal more is visible from reasonably-sized view volumes than from a single viewpoint, due to the multitude of pinhole lines of sight. Although this is a contrived example, real-world scenes such as jungle canopies have similar occlusion relationships.

Discussion

This article is intended to explore the theoretical performance of our optimized culling algorithm. Clearly, we do not provide a full discussion of practical implementation issues or predict relative performance compared with actual systems. For one thing, our *z-traffic* comparisons with standard z-buffering don't account for existing methods for reducing bandwidth, such as fast clear and z compression [Mor00]. Also, we assume that values in the zm-pyramid can be accessed individually, even though actual systems often perform tile-based memory accesses. Regarding box culling, our methods assume that scenes are organized in boxes which can be easily reordered without affecting the output image, even though support for this is largely absent in today's application programs. Further, our approach doesn't provide a way of anticipating geometry that will be coming into view, as precomputed visibility methods do [Tel92]. Despite these limitations, we hope that our analysis of theoretical performance offers incites into practical applications.

7 Acknowledgements

I am deeply indebted to Pat Hanrahan for inviting me to work on this problem at Stanford and for contributing good ideas. I also thank Kai Li and Adam Levinthal for critiquing an earlier draft of this paper, and Dave Kirk and Nvidia for permitting me to publish my latest results.

8 Conclusion

Within z-buffer systems, memory traffic in image values can be dramatically reduced by including an optimized conservative culling stage in the graphics pipeline that employs a zm-pyramid and simple polygon-tiling operations to cull occluded geometry and identify image samples that are actually visible. Commonly, optimized culling reduces image memory traffic more effectively than knowing in advance which polygons are visible in the output image, and rendering only them with standard z-buffering. In some cases, this even applies to rendering deeply occluded scenes of great visible complexity, provided that they are organized in bounding boxes and traversed in approximately front-to-back order. These characteristics indicate that integrating optimized culling procedures into hardware pipelines could go a long way towards enabling real-time animation of complex arbitrary scenes.

References

[Air90] J. Airey, “Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations,” PhD Thesis, Technical Report TR90-027, Computer Science Dept., UNC Chapel Hill, 1990.

[AM00] T. Aila and V. Miettinen, “Umbra Reference Manual," Hybrid Holding, Ltd., Helsinki, Finland, Oct. 2000.

[Car84] L. Carpenter, “The A-Buffer, an Antialiased Hidden Surface Method,” *Proc. of SIGGRAPH '84*, July 1984, 103-108.

[CCS00] D. Cohen-Or, Y. Chrysanthou, and C. Silva, "A Survey of Visibility for Walkthrough Applications," *Siggraph 2000 Course Notes: Visibility, Problems, Techniques, and Applications*, July, 2000.

[Cla76] J. H. Clark, “Hierarchical Geometric Models for Visible Surface Algorithms,” *Communications of the ACM* 19(10), Oct. 1976, 547-554.

[CT96] S. Coorg and S. Teller, “Temporally Coherent Conservative Visibility,” *Proc. Of 12th ACM Symposium on Computational Geometry*, 1996..

[Dee94] M. Deering, S. Schlapp, and M. Lavelle, “FBRAM: A new Form of Memory Optimized for 3D Graphics,” *Proc. of SIGGRAPH '94*, July 1994, 167-174.

[Dur00] F. Durand, G. Drettakis, J. Thollot, and C. Puech, “Conservative Visibility Preprocessing using Extended Projections,” *Proc. of SIGGRAPH '00*, July 2000, 239-248.

[Fuc85] H. Fuchs, J. Goldfeather, J. Hulquist, S. Spach, J. Austin, F. Brooks, Jr., J. Eyles, and J. Poulton, “Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes,” *Proc. of SIGGRAPH '85*, July 1985, 111-120.

[Fun93] T. Funkhouser and C. Sequin, “Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments,” *Proc. of SIGGRAPH '93*, Aug. 1993, 247-254.

[GBW90] B. Garlick, D. Baum, and J. Winget, “Interactive Viewing of Large Geometric Databases Using Multiprocessor Graphics Workstations,” *Siggraph ’90 Course Notes: Parallel Algorithms and Architectures for 3D Image Generation*, 1990.

[GKM93] N. Greene, M. Kass, and G. Miller, “Hierarchical Z-Buffer Visibility," *Proc. of SIGGRAPH '93*, July 1993, 231-238.

[Gre95] N. Greene, “Hierarchical Rendering of Complex Environments,” PhD Thesis, Univ. of California at Santa Cruz, Report UCSC CRL-95-27, June 1995.

[Gre96] N. Greene, “Hierarchical Polygon Tiling with Coverage Masks,” *Proc. of SIGGRAPH '96*, Aug 1996.

[Gre99a] N. Greene, “Occlusion Culling with Optimized Hierarchical Z-Buffering,*"* Siggraph Technical Sketch, Siggraph '99 Conference Abstracts and Applications, Aug. 1999.

[Gre99b] N. Greene, “Optimized Hierarchical Occlusion Culling for Z-Buffering Systems, Siggraph '99 Conference Abstracts and Applications (CD-ROM only), Aug. 1999.

[Hop96] H. Hoppe, “Progressive Meshes,” *Proc. of SIGGRAPH '96*, Aug 1996, 99-108.

[Hud97] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang, “Accelerated Occlusion Culling Using Shadow Frusta,” *Proc. Of ACM Symposium on Computational Geometry*, 1997.

[KS01] J. Klosowski and C. Silva, "Efficient Conservative Visibility Culling Using the Prioritized-Layered Projection Algorithm, IEEE Transactions on Visualization and Computer Graphics, to appear.

[LG95] D. Luebke and C. Georges, “Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets,” *ACM Interactive 3D Graphics Conference*, 1995.

[Mor00] S. Morein, "ATI Radeon HyperZ Technology" (slides outline only*), Graphics Hardware 2000, Hot3D Proceedings*, Aug. 2000.

[Mea82] D. Meagher, “The Octree Encoding Method for Efficient Solid Modeling,” PhD Thesis, Electrical Engineering Dept., Rensselaer Polytechnic Institute, Troy, New York, Aug. 1982.

[Mue95] C. Mueller, "Architectures of Image Generators for Flight Simulators," Tech. Report TR95-015, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, 1995.

[Nay92] B. Naylor, “Partitioning Tree Image Representation and Generation from 3D Geometric Models,” *Proc. of Graphics Interface*, 1992.

[Sch00] G. Schaufler, J. Dorsey, X. Decoret, and F. Sillion, “Conservative Volumetric Visibility with Occluder Fusion,” *Proc. of SIGGRAPH '00*, July 2000, 229-238.

[Sco98] N. Scott, D. Olsen, and E. Gannett,. “An Overview of the VISUALIZE fx Graphics Accelerator Hardware,” *The Hewlett-Packard Journal*, 49(2), May 1998, 28-34.

[SG99] O. Sudarsky and C. Gotsman, “Dynamic Scene Occlusion Culling,” *IEEE Transactions on Visualization and Computer Graphics*, 5(1), Jan. 1999.

[Tar99] Gary Tarolli, personal communication, 1999.

[Tel92] S. Teller, “Visibility Computations in Densely Occluded Polyhedral Environments,” PhD Thesis, Univ. of California at Berkeley, Report UCB/CSD 92/708, Oct. 1992.

[Tit93] “Denali Technical Overview,” Kubota Pacific Computer, Jan 1993.

[War69] J. Warnock, “A Hidden Surface Algorithm for Computer Generated Halftone Pictures,” PhD Thesis, TR 4-15, Computer Science Dept., Univ. of Utah, June 1969.

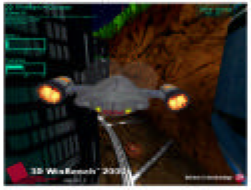
[XS99] F. Xie and M. Shantz, “Adaptive Hierarchical Visibility in a Tiled Architecture,” *Proc. Eurographics/Siggraph Workshop on Graphics Hardware*, Aug. 1999, 75-84.

[Zha97] H. Zhang, D. Manocha, T. Hudson, and K. Hoff, “Visibility Culling Using Hierarchical Occlusion Maps,” *Proc. of SIGGRAPH '97*, Aug. 1997, 77-88.

[Zha98] H. Zhang, “Effective Occlusion Culling for the Interactive Display of Arbitrary Models,” PhD Thesis, Computer Science Dept., UNC Chapel Hill, 1998.

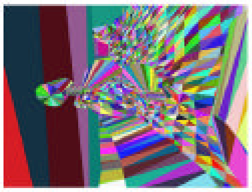
Winbench "Canyon"

Figure 6a



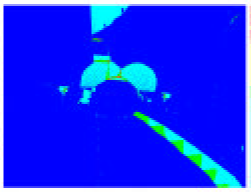
Image

Figure 6b



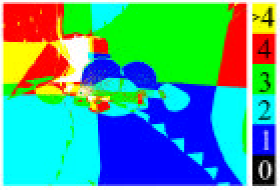
Object Tag

Figure 6c



number of z-buffer accesses
(trace order)
1.11 per sample avg.

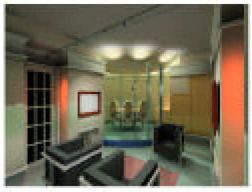
Figure 6d



number of z-buffer accesses
(reverse order)
2.53 per sample avg.

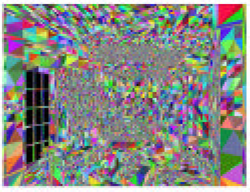
Lightscape Interior

Figure 7a



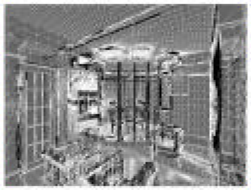
Image

Figure 7b



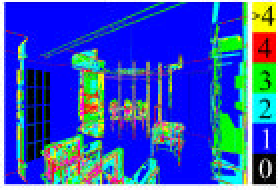
Object Tag

Figure 7c



visible polygons outlined
in white, occluded
polygons outlined in black

Figure 7d



number of z-buffer accesses
(reverse order)
1.53 per sample avg.

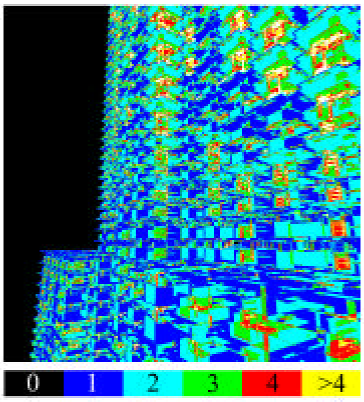


Figure 11. Z-buffer accesses generated by *optimized_culling* when rendering figure 8 with frame-coherent box culling (1.74 accesses per sample on average).

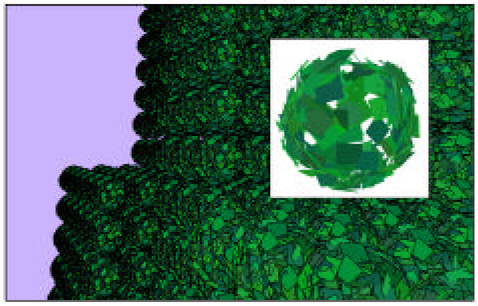


Figure 13. "Topiary Tower" model created by replacing office modules with topiary balls. Even with this pathological scene, *optimized_culling* culls effectively because the zm-pyramid exploits nearly all available occluder fusion.

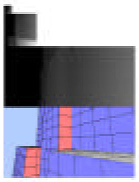


Figure 12
Simplified occluder model superimposed over corresponding z-tip.

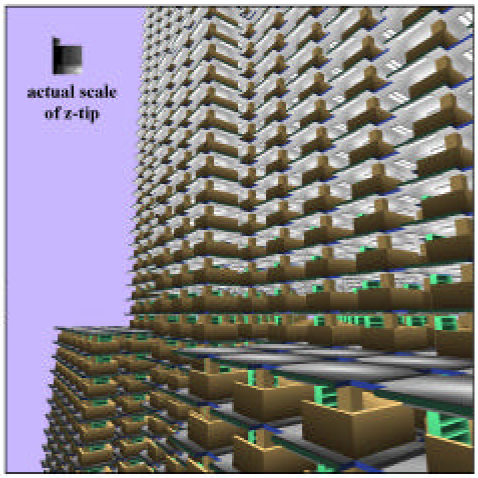


Figure 8. *optimized_culling* generates fewer z reads and writes in rendering this scene than "oracle z-buffering" (z-buffering just the polygons that are actually visible in front-to-back order).

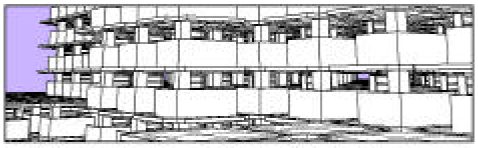
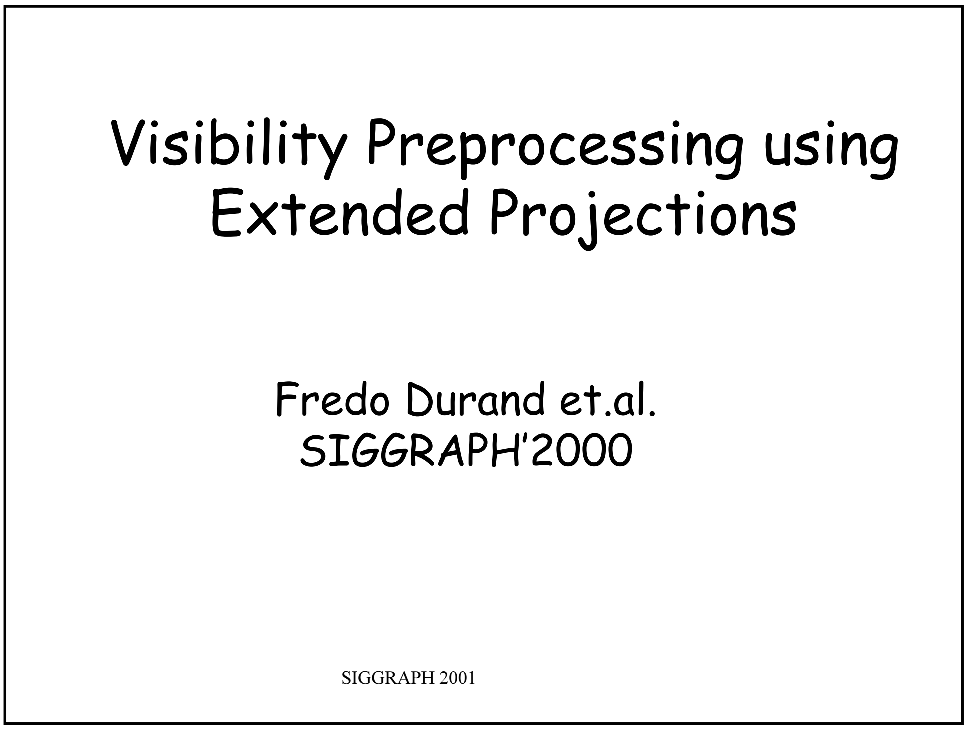


Figure 9. Wireframe closeup of skyscraper.

From-region Visibility (second part)

Vladlen Koltun
Tel-Aviv University

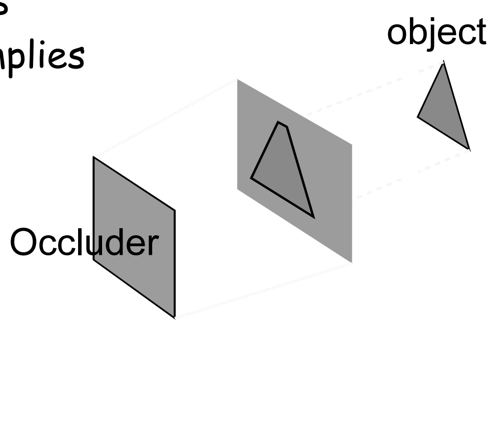


Visibility Preprocessing using Extended Projections

Fredo Durand et.al.
SIGGRAPH’2000

Extended projections

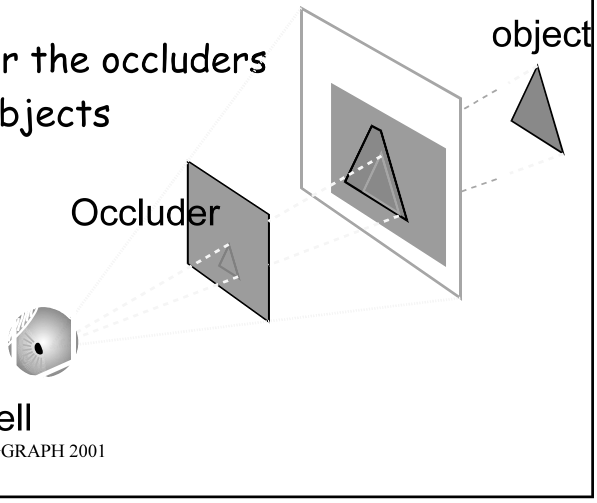
- Projection from a ~~point~~ volume
- Overlap and depth tests
- image-space inclusion implies object-space occlusion



Extended projections

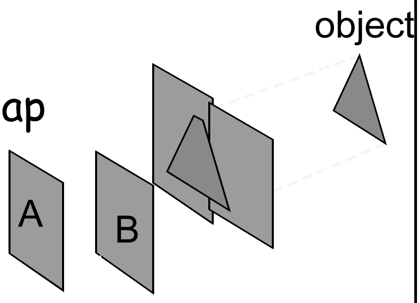
Conservative

- intersection for the occluders
- union for the objects



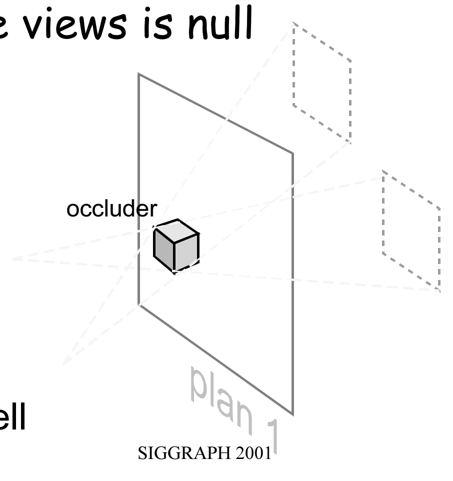
Occluder fusion

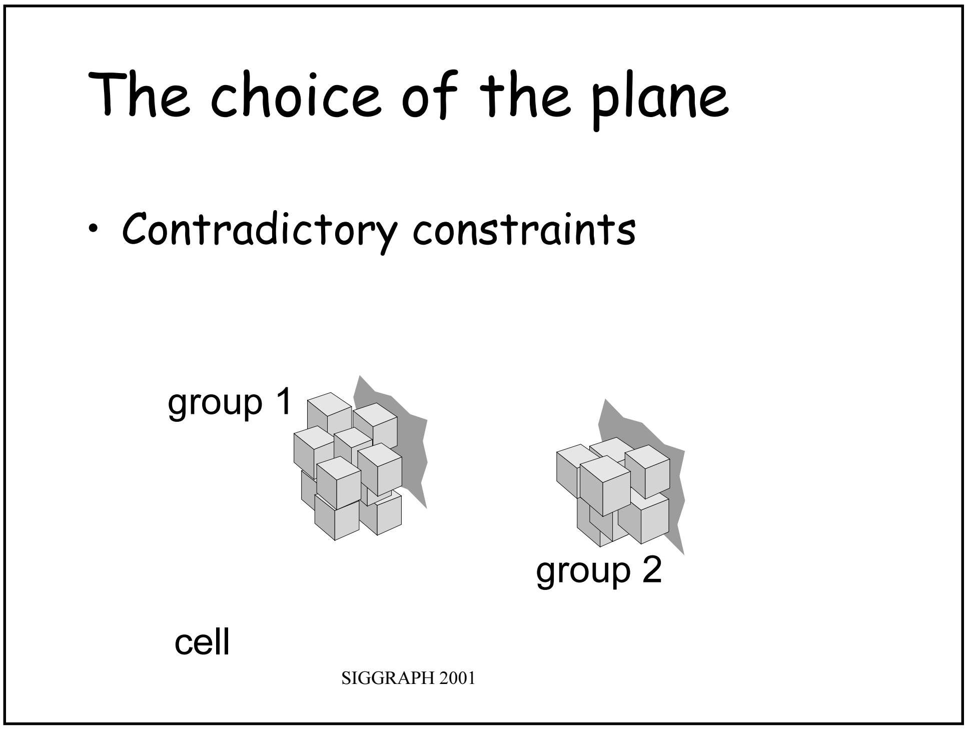
- Projection of the two occluders
- *Aggregation in a bitmap*



Problem of the choice of the plane

The intersections
of the views is null

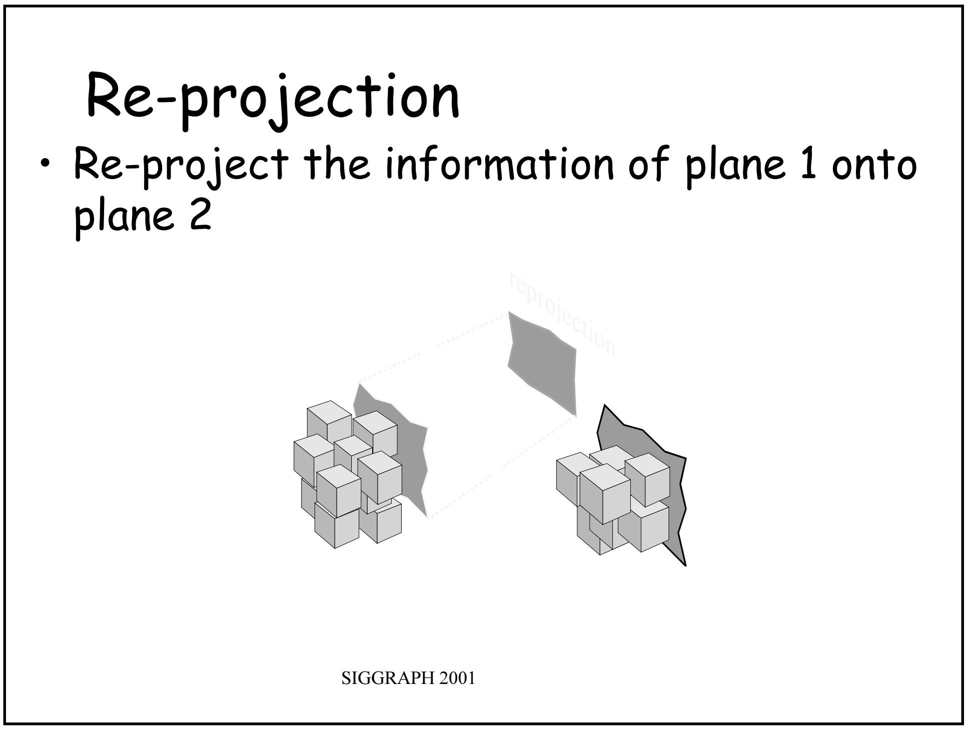




The choice of the plane

- Contradictory constraints

SIGGRAPH 2001



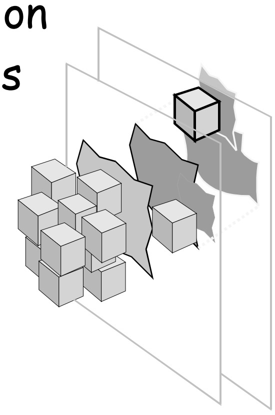
Re-projection

- Re-project the information of plane 1 onto plane 2

SIGGRAPH 2001

Occlusion sweep

Projection and Re-projection to aggregate the occlusions



The algorithm

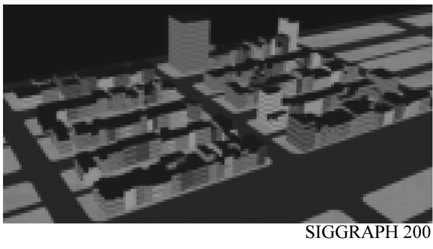
- Identify occluders
- Place projection planes
- Project all occluders onto the projection planes, and construct HOMs
- For each occludee, project it onto the closest projection plane, and test whether occluded

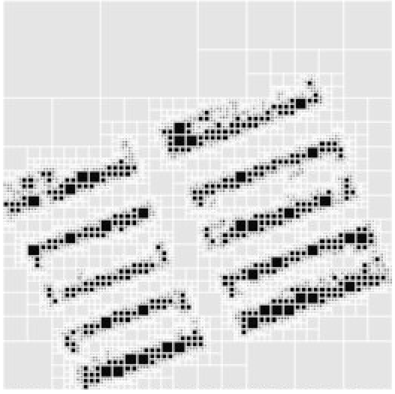
Volumetric Visibility with Occluder Fusion

Gernot Schaufler et.al.
SIGGRAPH'2000

Scene Discretisation

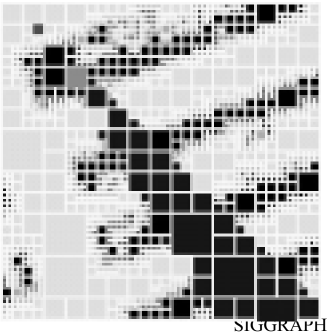
From polygonal representation to a volumetric representation

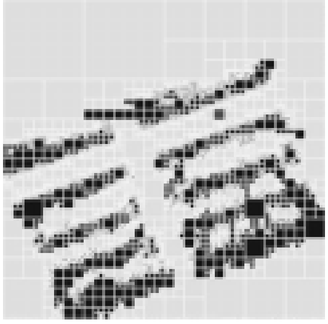




Voxel Classification

- Classify voxels that are occluded by opaque voxels as opaque.
- Conceptually equivalent to strong occlusion

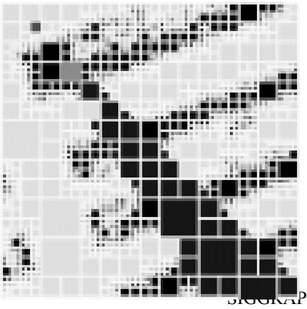




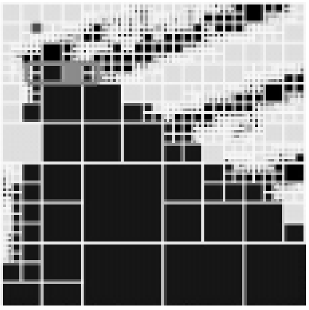
Blocker extension

- Adjacent opaque voxels are merged into larger blockers to yield larger occlusion

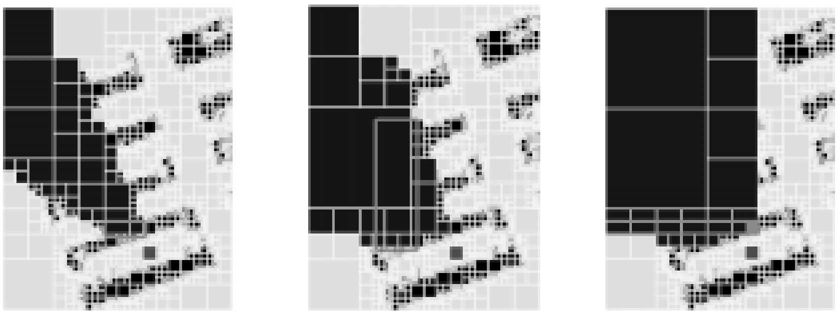
Occlusion of a single voxel



Occlusion of a combined occluders



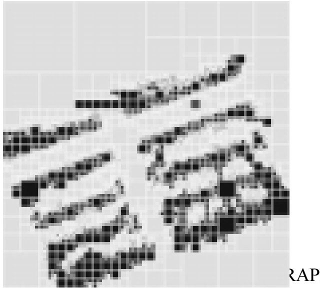
Extend blockers into hidden regions (iteratively)

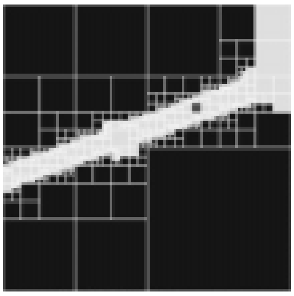


Results

Without blocker extension (strong occlusion only)

With blocker extension

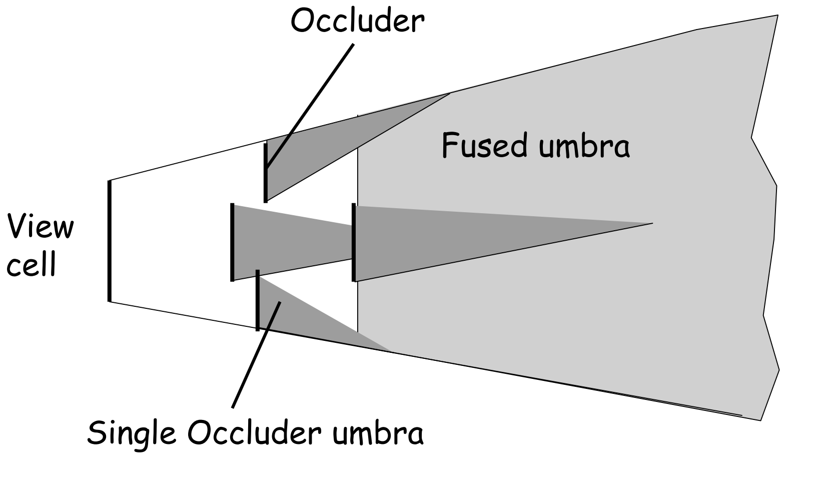


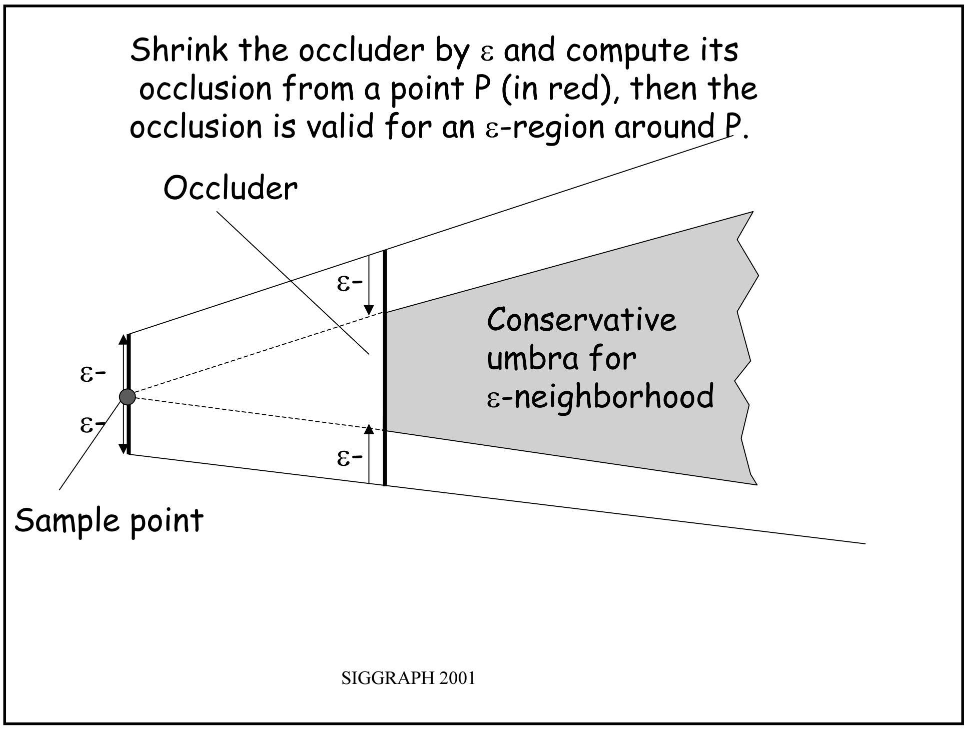


Occluder Fusion by Occluder Shrinking and Point Sampling

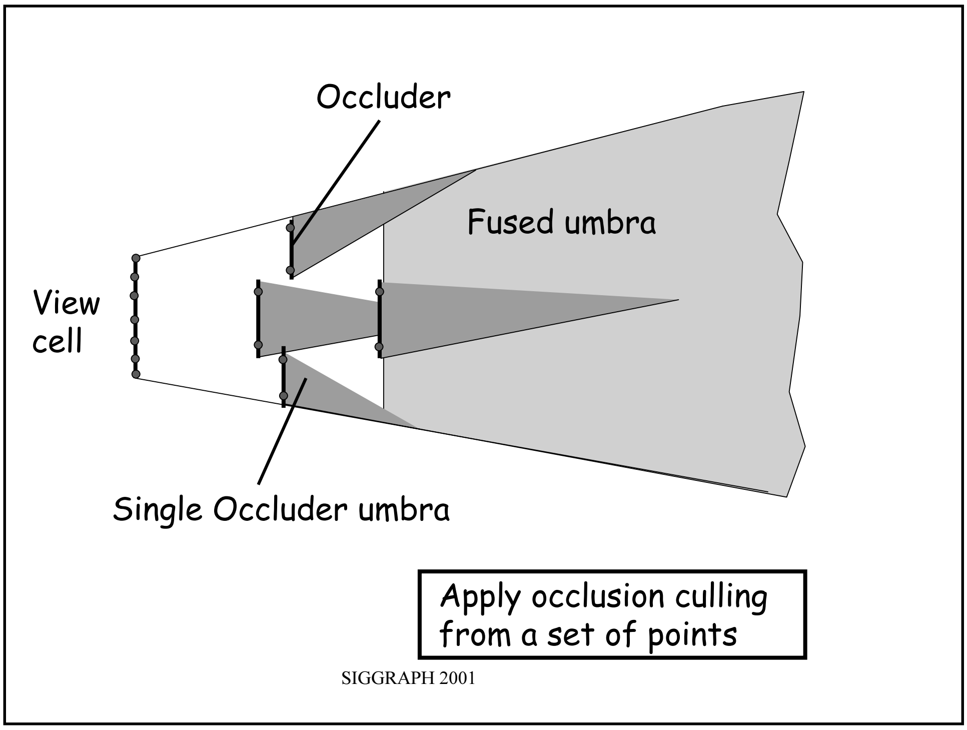
Peter Wonka et.al.

EGRW'2000

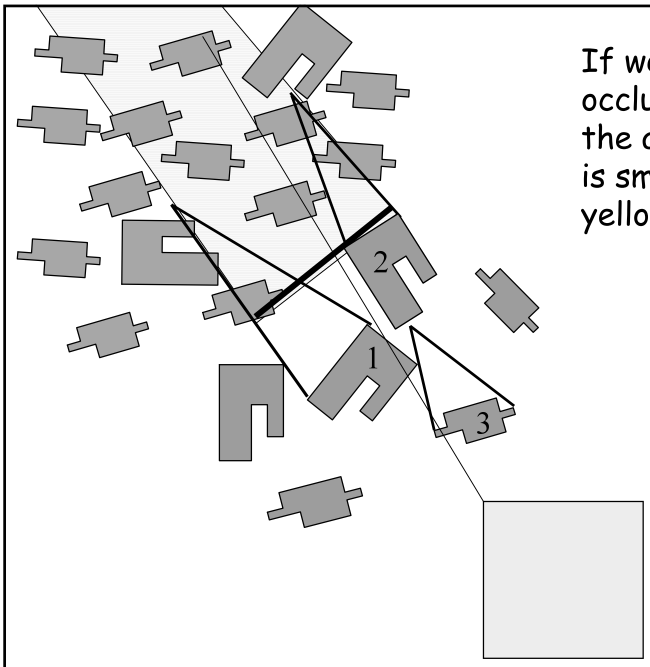




Shrink the occluder by ε and compute its occlusion from a point P (in red), then the occlusion is valid for an ε-region around P.



Apply occlusion culling from a set of points



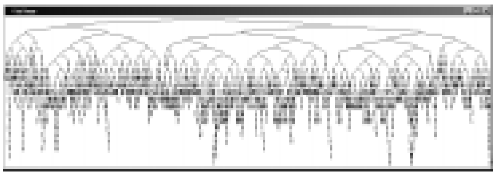
If we shrink the
occluders by ε then
the aggregate umbra
is small ! (see the
yellow sight ray)

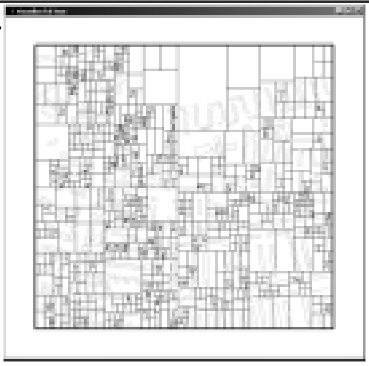
Hardware-Accelerated from-Region Visibility using a Dual Ray Space

Vladlen Koltun
Daniel Cohen-Or
Yiorgos Chrysanthou (UCL)

Overview of the new method

- Place the scene in an axis aligned partitioning tree (kd-tree)
- Select a region of interest (viewcell)
- Traverse the kd-tree hierarchically top-down
- At each step decide if node is visible

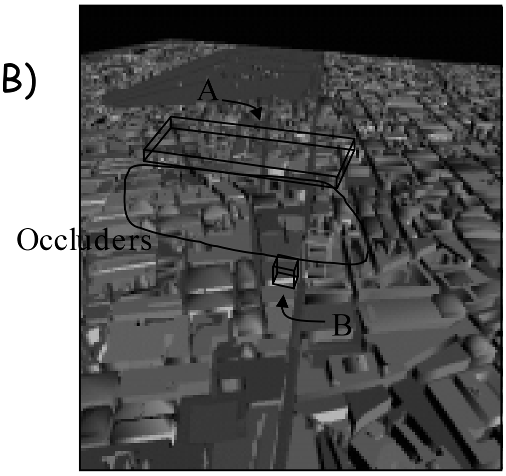




SIGGRAPH 2001

Determine Visibility Between Two Boxes

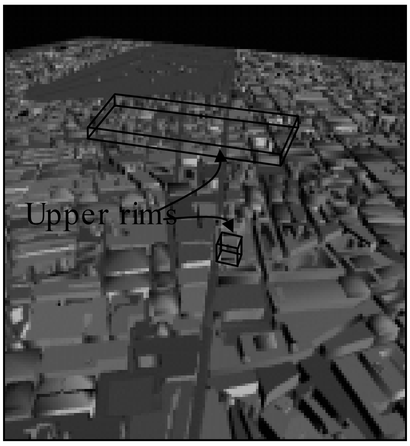
- Q: given two axis aligned boxes (A, B) and a set of occluding objects (S), are A and B mutually visible?



SIGGRAPH 2001

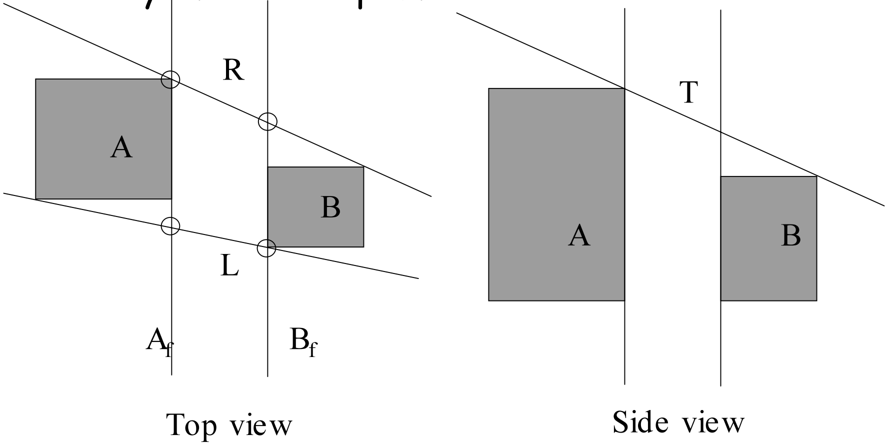
Reduction to a Planar Problem

- Assumption: All occluders are xy-monotonic
- Observation: A and B are mutually visible if and only if their upper rims are visible



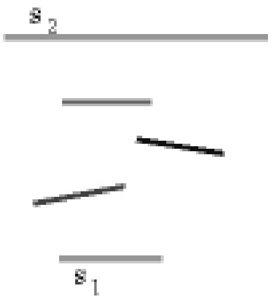
Reduction to a Planar Problem

Visibility between A and B is equivalent to visibility inside the plane T

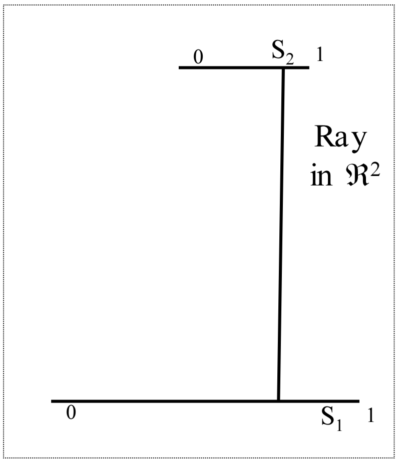


Planar Visibility Test

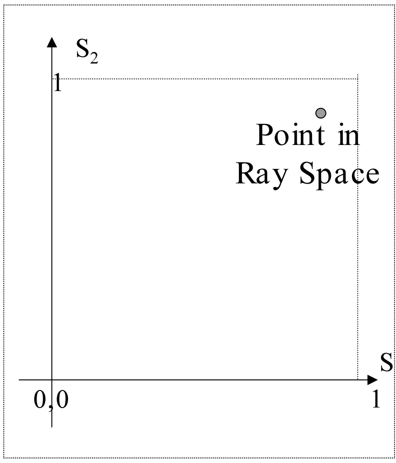
- Q:given two edges, *s_1* and *s_2*, in the plane and a set of occluder edges *O*, are the edges mutually visible?
- Linear separation, *$O(n^2)$* on the number of occluder edges



The Dual Ray Space

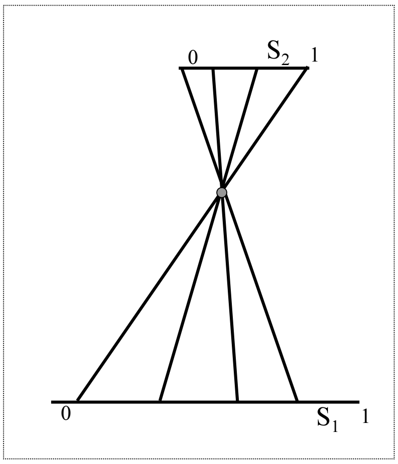


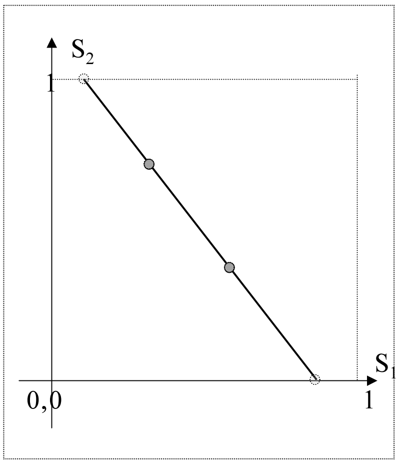
$\Re^2$



Ray Space

The Dual Ray Space

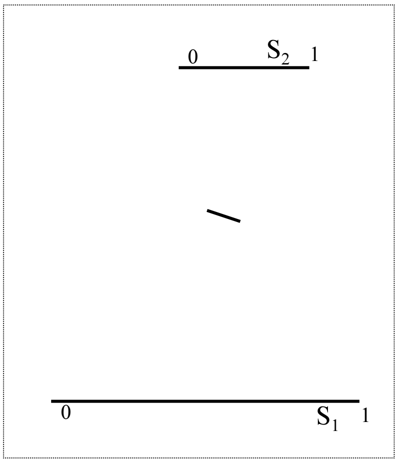


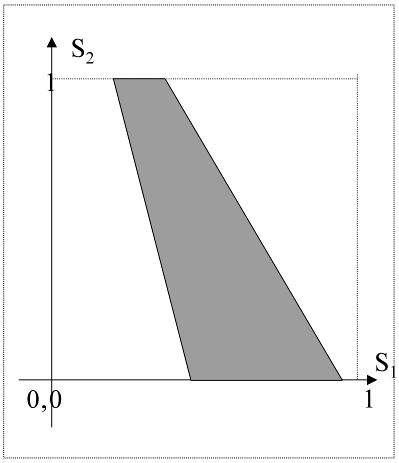


$\Re^2$

Ray Space

The Dual Ray Space

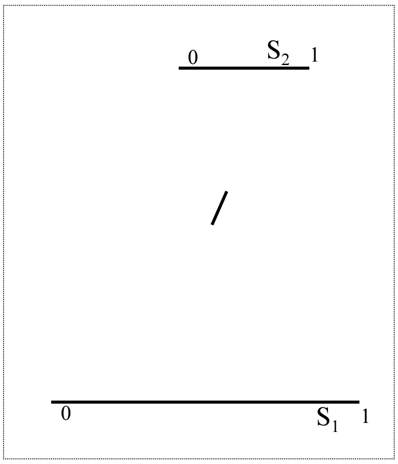




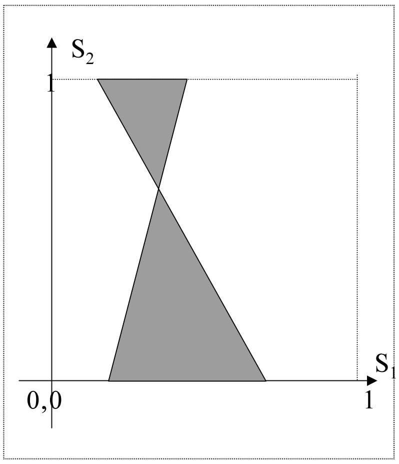
$\Re^2$

Ray Space

The Dual Ray Space



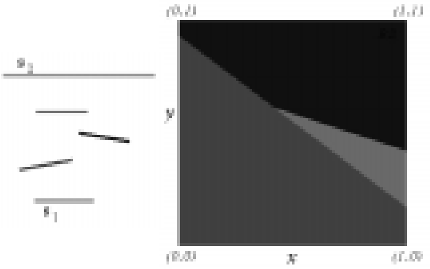
$\Re^2$



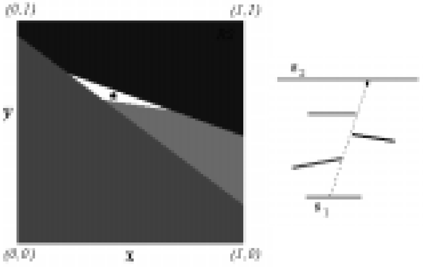
Ray Space

Hardware Accelerated

- Render all the trapezes and double-triangles (the segments in the dual space)
- Check whether the frame buffer is fully covered



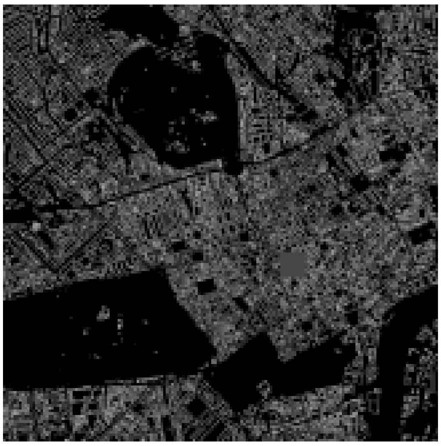
Occluded



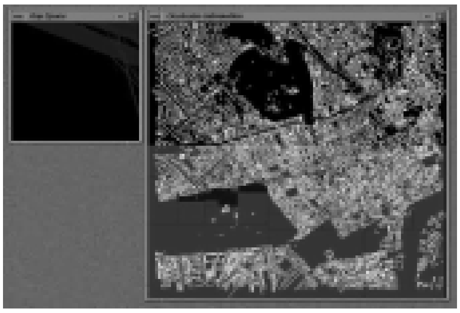
Visible

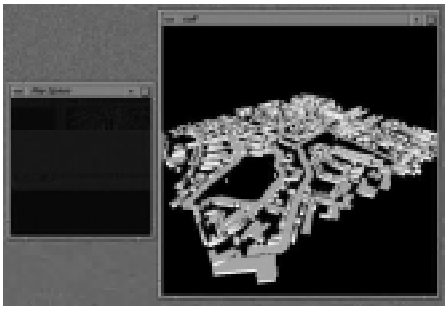
Results

- Visibility for a 100x100 viewcell is computed in 2.5 seconds on average
- It takes 60 seconds to walk 100 meters with a speed of 6 km/h
- It can be employed *on-line*, without preprocessing!



Demo





2D overview ▷

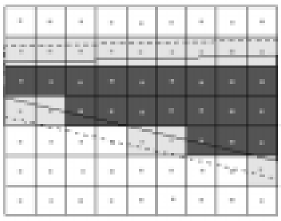
3D view ▷

xy-monotone occluders

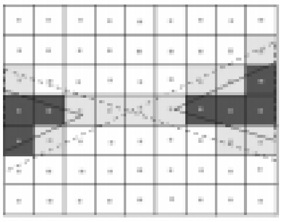
- Commercial Virtual Environments usually model realistic scenes, e.g. urban and architectural environments
- *xy*-monotone occluders can be synthesized (e.g. by techniques similar to [Andujar et.al., CAD 2000])
- Our approach is not as restrictive as assuming *xy*-monotonicity of the whole scene.

Ensuring conservativity

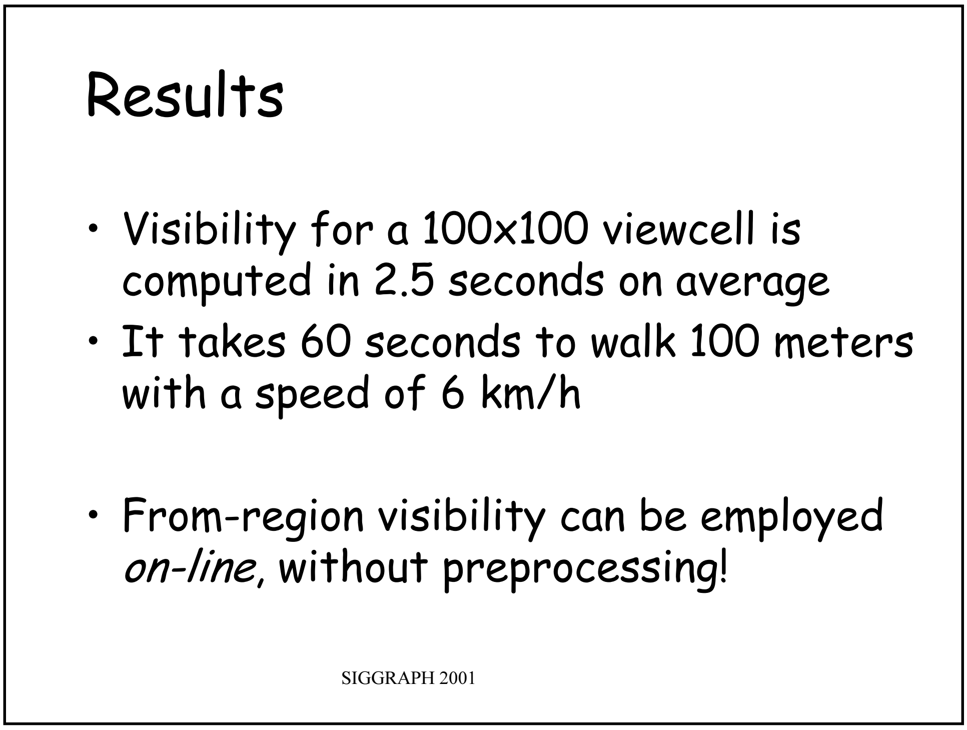
- Shrinking is employed to ensure that only pixels that are *fully* covered are colored
- The edges are moved inward by (√2/2)a, where a is the pixel size



$\mathcal{T}(v)$ is a trapeze



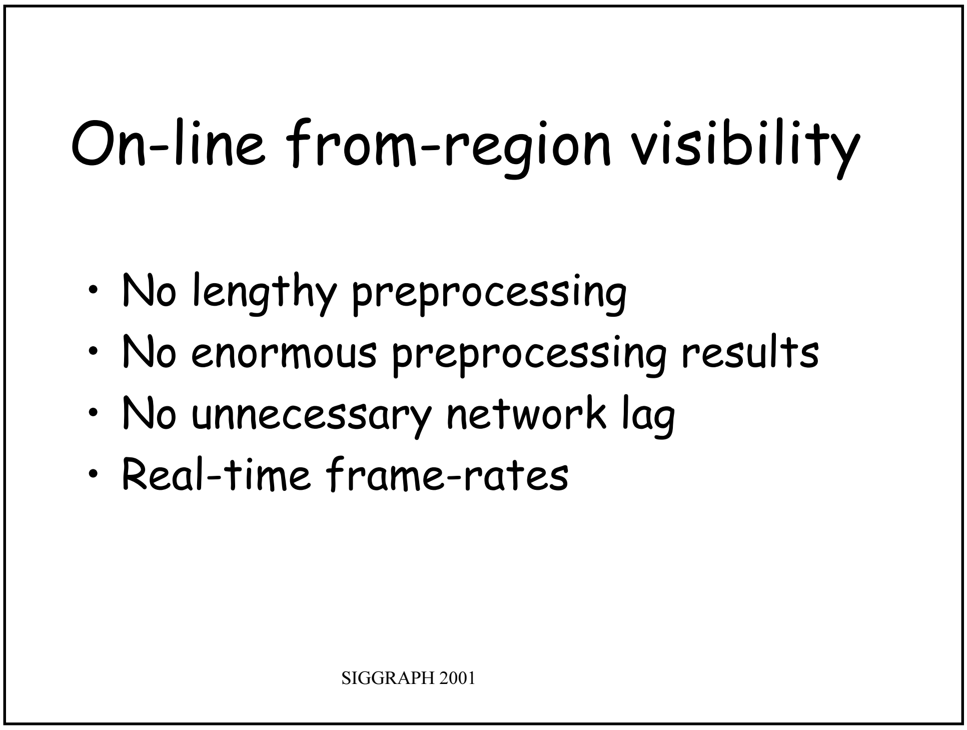
$\mathcal{T}(v)$ is a double-triangle



Results

- Visibility for a 100x100 viewcell is computed in 2.5 seconds on average
- It takes 60 seconds to walk 100 meters with a speed of 6 km/h

- From-region visibility can be employed *on-line*, without preprocessing!



On-line from-region visibility

- No lengthy preprocessing
- No enormous preprocessing results
- No unnecessary network lag
- Real-time frame-rates

Thank you

SIGGRAPH 2001