Integration of preprocessed visibility

Jiří Bittner, Oliver Mattausch, Michael Wimmer
Institute of Computer Graphics and Algorithms, Vienna University of Technology

January 30, 2007

Contents

1 Introduction **2**
1.1 Structure of the report 2
1.2 Domain of visibility problems 2
1.3 Dimension of the problem-relevant line set 3
1.4 Classification of visibility algorithms 5
1.5 Summary 8

Chapter 1

Requirements

Integration of the preprocessed visibility into your engine Requirements

Following libraries compile the Preprocessor application and must be added to application

1. *zdll.lib*
2. *zziplib.lib*
3. *xercesc_2.lib*
4. *devil.lib*
5. *glut32.lib*
6. *OpenGL32.Lib*
7. *glu32.lib*
8. *glew32.lib*
9. *glew32s.lib*

These libariys are all part of the NonGtp directory of the repository.

Chapter 2

Build Preprocessor.lib

Open the GtpVisibility.sln with Visual Studio 2003 (2005 might work, but we did not test it). Use Release; as build target. Build the project Testpreprocessor.

Chapter 3

Generate Visibility Solution

3.1 Structure of the report

The report consists of two introductory chapters, which provide a theoretical background for description of the algorithms, and three chapters dealing with the actual visibility algorithms.

This chapter provides an introduction to visibility by using a taxonomy of visibility problems and algorithms. The taxonomy is used to classify the later described visibility algorithms. Chapter **??** provides an analysis of visibility in 2D and 3D polygonal scenes. This analysis also includes formal description of visibility using Plücker coordinates of lines. Plücker coordinates are exploited later in algorithms for mutual visibility verification (Chapter **??**).

Chapter **??** describes a visibility culling algorithm used to implement the online visibility culling module. This algorithm can be used accelerate rendering of fully dynamic scenes using recent graphics hardware. Chapter **??** describes global visibility sampling algorithm which forms a core of the PVS computation module. This chapter also describes view space partitioning algorithms used in close relation with the PVS computation. Finally, Chapter **??** describes mutual visibility verification algorithms, which are used by the PVS computation module to generate the final solution for precomputed visibility.

3.2 Domain of visibility problems

Computer graphics deals with visibility problems in the context of 2D, $2\frac{1}{2}$D, or 3D scenes. The actual problem domain is given by restricting the set of rays for which visibility should be determined.

Below we list common problem domains used and the corresponding domain restrictions:

1. *visibility along a line*
   (a) line
   (b) ray (origin + direction)

2. *visibility from a point* (*from-point visibility*)
   (a) point
   (b) point + restricted set of rays
      i. point + raster image (discrete form)
      ii. point + beam (continuous form)
3. *visibility from a line segment* (*from-segment visibility*)
   (a) line segment
   (b) line segment + restricted set of rays
4. *visibility from a polygon* (*from-polygon visibility*)
   (a) polygon
   (b) polygon + restricted set of rays
5. *visibility from a region* (*from-region visibility*)
   (a) region
   (b) region + restricted set of rays
6. *global visibility*
   (a) no further input (all rays in the scene)
   (b) restricted set of rays

The domain restrictions can be given independently of the dimension of the scene, but the impact of the restrictions differs depending on the scene dimension. For example, visibility from a polygon is equivalent to visibility from a (polygonal) region in 2D, but not in 3D.

3.3 Dimension of the problem-relevant line set

The six domains of visibility problems stated in Section 1.2 can be characterized by the *problem-relevant line set* denoted $\mathcal{L}_R$. We give a classification of visibility problems according to the dimension of the problem-relevant line set. We discuss why this classification is important for understanding the nature of the given visibility problem and for identifying its relation to other problems.

For the following discussion we assume that a line in *primal space* can be mapped to a point in *line space*. For purposes of the classification we define the line space as a vector space where a point corresponds to a line in the primal space[1].

3.3.1 Parametrization of lines in 2D

There are two independent parameters that specify a 2D line and thus the corresponding set of lines is two-dimensional. There is a natural duality between lines and points in 2D. For example a line expressed as: $l : y = ax + c$ is dual to a point $p = (-c, a)$. This particular duality cannot handle vertical lines. example of other dual mappings in the plane. To avoid the singularity in the mapping, a line $l : ax + by + c = 0$ can be represented as a point $p_l = (a, b, c)$ in 2D projective space $\mathcal{P}^2$ [**?**]. Multiplying p_l by a non-zero scalar we obtain a vector that represents the same line l. More details about this singularity-free mapping will be discussed in Chapter **??**.

To sum up: In 2D there are two degrees of freedom in description of a line and the corresponding line space is two-dimensional. The problem-relevant line set $\mathcal{L}_R$ then forms a k-dimensional subset of $\mathcal{P}^2$, where $0 \leq k \leq 2$. An illustration of the concept of the problem-relevant line set is depicted in

[1]A classical mathematical definition says: Line space is a direct product of two Hilbert spaces [**?**]. However, this definition differs from the common understanding of line space in computer graphics [**?**]

3.3.2 Parametrization of lines in 3D

Lines in 3D form a four-parametric space [**?**]. A line intersecting a given scene can be described by two points on a sphere enclosing the scene. Since the surface of the sphere is a two parametric space, we need four parameters to describe the line.

The *two plane parametrization* of 3D lines describes a line by points of intersection with the given two planes [**?**]. This parametrization exhibits a singularity since it cannot describe lines parallel to these planes. See two plane parameterizations.

Another common parametrization of 3D lines are the *Plücker coordinates*. Plücker coordinates of an oriented 3D line are a six tuple that can be understood as a point in 5D oriented projective space [**?**]. There are six coordinates in Plücker representation of a line although we know that the $\mathcal{L}_R$ is four-dimensional. This can be explained as follows:

- Firstly, Plücker coordinates are *homogeneous coordinates* of a 5D point. By multiplication of the coordinates by any positive scalar we get a mapping of the same line.

- Secondly, only 4D subset of the 5D oriented projective space corresponds to real lines. This subset is a 4D ruled quadric called the *Plücker quadric* or the *Grassman manifold* [**?**, **?**].

Although the Plücker coordinates need more coefficients they have no singularity and preserve some linearities: lines intersecting a set of lines in 3D correspond to an intersection of 5D hyperplanes. More details on Plücker coordinates will be discussed in Chapter **??** and Chapter **??** where they are used to solve the from-region visibility problem.

To sum up: In 3D there are four degrees of freedom in the description of a line and thus the corresponding line space is four-dimensional. Fixing certain line parameters (e.g. direction) the problem-relevant line set, denoted $\mathcal{L}_R$, forms a k-dimensional subset of $\mathcal{P}^4$, where $0 \leq k \leq 4$.

3.3.3 Visibility along a line

The simplest visibility problems deal with visibility along a single line. The problem-relevant line set is zero-dimensional, i.e. it is fully specified by the given line. A typical example of a visibility along a line problem is *ray shooting*.

A similar problem to ray shooting is the *point-to-point* visibility. The point-to-point visibility determines whether the line segment between two points is occluded, i.e. it has an intersection with an opaque object in the scene. Point-to-point visibility provides a visibility classification (answer 1a), whereas ray shooting determines a visible object (answer 2a) and/or a point of intersection (answer 3a). Note that the *point-to-point* visibility can be solved easily by means of ray shooting. Another constructive visibility along a line problem is determining the *for an illustration of typical visibility along a line problems.*

3.3.4 Visibility from a point

Lines intersecting a point in 3D can be described by two parameters. For example the lines can be expressed by an intersection with a unit sphere centered at the given point. The most common parametrization describes a line by a point of intersection with a given viewport. Note that this parametrization accounts only for a

In 3D the problem-relevant line set $\mathcal{L}_R$ is a 2D subset of the 4D line space. In 2D the $\mathcal{L}_R$ is a 1D subset of the 2D line space. The typical visibility from a point problem is the visible surface determination. Due to its importance the visible surface determination is covered by the majority of existing visibility algorithms. Other visibility from a point problem is the construction of the visibility map *or the* point-to-region visibility *that classifies a region as visible, invisible, or partially visible with respect to the given point.*

3.3.5 Visibility from a line segment

Lines intersecting a line segment in 3D can be described by three parameters. One parameter fixes the intersection of the line with the segment the other two express the direction of the line. The problem-relevant line set $\mathcal{L}_R$ is three-dimensional and it can be understood as a 2D cross section of $\mathcal{L}_R$ swept according to the translation on the given line segment (see

In 2D lines intersecting a line segment form a two-dimensional problem-relevant line set. Thus for the 2D case the $\mathcal{L}_R$ is a two-dimensional subset of 2D line space.

3.3.6 Visibility from a region

Visibility from a region (or from-region visibility) involves the most general visibility problems. In 3D the $\mathcal{L}_R$ is a 4D subset of the 4D line space. In 2D the $\mathcal{L}_R$ is a 2D subset of the 2D line space. Consequently, in the presented classification visibility from a region in 2D is equivalent to visibility from a line segment in 2D.

A typical visibility from a region problem is the problem of region-to-region *visibility that aims to determine if the two given regions in the scene are visible, invisible, or partially visible (see computation of a* potentially visible set *(PVS) with respect to a given view cell. The PVS consists of a set of objects that are potentially visible from any point inside the view cell. Further visibility from a region problems include computing form factors between two polygons, soft shadow algorithms or discontinuity meshing.*

3.3.7 Global visibility

According to the classification the global visibility problems can be seen as an extension of the from-region visibility problems. The dimension of the problem-relevant line set is the same ($k = 2$ for 2D and $k = 4$ for 3D scenes). Nevertheless, the global visibility problems typically deal with much larger set of rays, i.e. all rays that penetrate the scene. Additionally, there is no given set of reference points from which visibility is studied and hence there is no given priority ordering of objects along each particular line from $\mathcal{L}_R$. Therefore an additional parameter must be used to describe visibility (visible object) along each ray.

3.3.8 Summary

*The classification of visibility problems according to the dimension of the problem-relevant line set is summarized in Table 1.1. This classification provides means for understanding how difficult it is to compute, describe, and maintain visibility for a particular class of problems. For example a data structure representing the visible or occluded parts of the scene for the visibility from a point problem needs to partition a 2D $\mathcal{L}_R$ into visible and occluded sets of lines. This observation conforms with the traditional visible surface algorithms – they partition a 2D viewport into empty/nonempty regions and associate each nonempty regions (pixels) with a visible object. In this case the viewport represents the $\mathcal{L}_R$ as each point of the viewport corresponds to a line through that point. To analytically describe visibility from a region a subdivision of 4D $\mathcal{L}_R$ should be performed. This is much more difficult than the 2D subdivision. Moreover the description of visibility from a region involves non-linear subdivisions of both primal space and line space even for polygonal scenes [**?**, **?**].*

3.4 Classification of visibility algorithms

The taxonomy of visibility problems groups similar visibility problems in the same class. A visibility problem can be solved by means of various visibility algorithms. A visibility algorithm poses further restrictions on the input and output data. These restrictions can be seen as a more precise definition of the visibility problem that is solved by the algorithm.

Above we classified visibility problems according to the problem domain and the desired answers. In this section we provide a classification of visibility algorithms according to other important criteria characterizing a particular visibility algorithm.

2D		
domain	$d(\mathcal{L}_R)$	problems
visibility along a line	0	ray shooting, point-to-point visibility
visibility from a point	1	view around a point, point-to-region visibility
visibility from a line segment visibility from region global visibility	2	region-to-region visibility, PVS
3D		
domain	$d(\mathcal{L}_R)$	problems
visibility along a line	0	ray shooting, point-to-point visibility
from point in a surface	1	see visibility from point in 2D
visibility from a point	2	visible (hidden) surfaces, point-to-region visibility, visibility map, hard shadows
visibility from a line segment	3	segment-to-region visibility (rare)
visibility from a region global visibility	4	region-region visibility, PVS, aspect graph, soft shadows, discontinuity meshing

Table 3.1: Classification of visibility problems in 2D and 3D according to the dimension of the problem-relevant line set.

3.4.1 Scene restrictions

Visibility algorithms can be classified according to the restrictions they pose on the scene description. The type of the scene description influences the difficulty of solving the given problem: it is simpler to implement an algorithm computing a visibility map for scenes consisting of triangles than for scenes with NURBS surfaces. We list common restrictions on the scene primitives suitable for visibility computations:

- *triangles, convex polygons, concave polygons,*
- *volumetric data,*
- *points,*
- *general parametric, implicit, or procedural surfaces.*

Some attributes of scenes objects further increase the complexity of the visibility computation:

- *transparent objects,*
- *dynamic objects.*

The majority of analytic visibility algorithms deals with static polygonal scenes without transparency. The polygons are often subdivided into triangles for easier manipulation and representation.

3.4.2 Accuracy

Visibility algorithms can be classified according to the accuracy of the result as:

- *exact,*
- *conservative,*
- *aggressive,*
- *approximate.*

An exact algorithm provides an exact analytic result for the given problem (in practice however this result is typically influenced by the finite precision of the floating point arithmetics). A conservative algorithm overestimates visibility, i.e. it never misses any visible object, surface or point. An aggressive algorithm always underestimates visibility, i.e. it never reports an invisible object, surface or point as visible. An approximate algorithm provides only an approximation of the result, i.e. it can overestimate visibility for one input and underestimate visibility for another input.

The classification according to the accuracy is best illustrated on computing PVS: an exact algorithm computes an exact PVS. A conservative algorithm computes a superset of the exact PVS. An aggressive algorithm determines a subset of the exact PVS. An approximate algorithm computes an approximation to the exact PVS that is neither its subset or its superset for all possible inputs.

A more precise quality measure of algorithms computing PVSs can be expressed by the relative overestimation *and the* relative underestimation *of the PVS with respect to the exact PVS. We can define a quality measure of an algorithm A on input I as a tuple* $\boldsymbol{Q}^A(I)$*:*

$$\boldsymbol{Q}^A(I) = (Q_o^A(I), Q_u^A(I)), \qquad I \in \mathcal{D} \tag{3.1}$$

$$Q_o^A(I) = \frac{|S^A(I) \setminus S^{\mathcal{E}}(I)|}{|S^{\mathcal{E}}(I)|} \tag{3.2}$$

$$Q_u^A(I) = \frac{|S^{\mathcal{E}}(I) \setminus S^A(I)|}{|S^{\mathcal{E}}(I)|} \tag{3.3}$$

where I is an input from the input domain $\mathcal{D}$*,* $S^A(I)$ *is the PVS determined by the algorithm A for input I and* $S^{\mathcal{E}}(I)$ *is the exact PVS for the given input.* $Q_o^A(I)$ *expresses the* relative overestimation *of the PVS,* $Q_u^A(I)$ *is the* relative underestimation.

The expected quality of the algorithm over all possible inputs can be given as:

$$Q^A = E[\|\boldsymbol{Q}^A(I)\|] \tag{3.4}$$

$$= \sum_{\forall I \in \mathcal{D}} f(I).\sqrt{Q_o^A(I)^2 + Q_o^A(I)^2} \tag{3.5}$$

where f(I) is the probability density function expressing the probability of occurrence of input I. The quality measure $\boldsymbol{Q}^A(I)$ *can be used to classify a PVS algorithm into one of the four accuracy classes according to Section 1.4.2:*

1. *exact*
   $\forall I \in \mathcal{D} : Q_o^A(I) = 0 \wedge Q_u^A(I) = 0$
2. *conservative*
   $\forall I \in \mathcal{D} : Q_o^A(I) \geq 0 \wedge Q_u^A(I) = 0$
3. *aggressive*
   $\forall I \in \mathcal{D} : Q_o^A(I) = 0 \wedge Q_u^A(I) \geq 0$
4. *approximate*
   $\exists I_j, I_k \in \mathcal{D} : Q_o^A(I_j) > 0 \wedge Q_u^A(I_k) > 0$

3.4.3 Solution space

The solution space is the domain in which the algorithm determines the desired result. Note that the solution space does not need to match the domain of the result.

The algorithms can be classified as:

- *discrete,*
- *continuous,*

- *hybrid.*

A discrete algorithm solves the problem using a discrete solution space; the solution is typically an approximation of the result. A continuous algorithm works in a continuous domain and often computes an analytic solution to the given problem. A hybrid algorithm uses both the discrete and the continuous solution space.

*The classification according to the solution space is easily demonstrated on visible surface algorithms: The z-buffer [**?**] is a common example of a discrete algorithm. The Weiler-Atherton algorithm [**?**] is an example of a continuous one. A hybrid solution space is used by scan-line algorithms that solve the problem in discrete steps (scan-lines) and for each step they provide a continuous solution (spans).*

Further classification reflects the semantics of the solution space. According to this criteria we can classify the algorithms as:

- *primal space (object space),*
- *line space,*
  - *image space,*
  - *general,*
- *hybrid.*

A primal space algorithm solves the problem by studying the visibility between objects without a transformation to a different solution space. A line space algorithm studies visibility using a transformation of the problem to line space. Image space algorithms can be seen as an important subclass of line space algorithms for solving visibility from a point problems in 3D. These algorithms cover all visible surface algorithms and many visibility culling algorithms. They solve visibility in a given image plane that represents the problem-relevant line set $\mathcal{L}_R$ — each ray originating at the viewpoint corresponds to a point in the image plane.

The described classification differs from the sometimes mentioned understanding of image space and object space algorithms that incorrectly considers all image space algorithms discrete and all object space algorithms continuous.

3.5 Summary

The presented taxonomy classifies visibility problems independently of their target application. The classification should help to understand the nature of the given problem and it should assist in finding relationships between visibility problems and algorithms in different application areas. The algorithms address the following classes of visibility problems:

- *Visibility from a point in 3D* $d(\mathcal{L}_R) = 2$.
- *Global visibility in 3D* $d(\mathcal{L}_R) = 4$.
- *Visibility from a region in 3D,* $d(\mathcal{L}_R) = 4$.

This chapter discussed several important criteria for the classification of visibility algorithms. This classification can be seen as a finer structuring of the taxonomy of visibility problems. We discussed important steps in the design of a visibility algorithm that should also assist in understanding the quality of a visibility algorithm. According to the classification the visibility algorithms described later in the report address algorithms with the following properties:

- *Domain:*
  - *viewpoint (online visibility culling),*
  - *global visibility (global visibility sampling)*

  - *polygon or polyhedron (mutual visibility verification)*
- *Scene restrictions (occluders):*
  - *meshes consisting of convex polygons*
- *Scene restrictions (group objects):*
  - *bounding boxes*
- *Output:*
  - *PVS*
- *Accuracy:*
  - *conservative*
  - *exact*
  - *aggresive*
- *Solution space:*
  - *discrete (online visibility culling, global visibility sampling, conservative and approximate algorithm from the mutual visibility verification)*
  - *continuous (exact algorithm from mutual visibility verification)*
- *Solution space data structures: viewport (online visibility culling), ray stack (global visibility sampling, conservative and approximate algorithm from the mutual visibility verification), BSP tree (exact algorithm from the mutual visibility verification)*
- *Use of coherence of visibility:*
  - *spatial coherence (all algorithms)*
  - *temporal coherence (online visibility culling)*
- *Output sensitivity: expected in practice (all algorithms)*
- *Acceleration data structure: kD-tree (all algorithms)*
- *Use of graphics hardware: online visibility culling*