Hidden Line Problem Formulated as a Set Union Problem

T. Hrúz[1], I. Považan[2] and R. Gosiorovský[3]

Abstract *This paper describes an alternative approach to the hidden line problem in computer graphics. It is assumed that a 3D visible scene consisting of convex planar polygons with known visibility order is given. An abstract data structure together with a set of operations for efficient solving of hidden line problem in the image space is defined. The main operation UNION is then implemented on the segment tree data structure. The solution of the visibility problem relies mainly on a two-way scan conversion process and the hidden line problem is formulated as a set union problem. The worst case complexity of the presented algorithm is* $O(sn \log s)$ *where* n *is the number of polygons and* s *is a resolution. The algorithm is output sensitive in the image space sense. In certain situations priority order is given or easily computable and resolution of raster space is very high. In these situations the algorithm presented can be faster than Z-buffer.*

1 Introduction

Hidden line and hidden surface problem is an important algorithmic problem in computer graphics. The problem is to efficiently render a scene consisting from objects in 3D space so that a realistic simulation of a view of some observer is generated. The effort is mainly concentrated on the identification of invisible parts of objects obscured by other objects. The result of an algorithmic solution of the problem is illustrated in Figures 6 and 5. These are the instances of scenes on which the basic features of the presented algorithm are later experimentally illustrated.

A realistic rendering of objects in 3 dimensional space is an extremely complex and computationally intensive task, therefore certain simplified models have been developed to get at least reasonable rendering speed. One of them is so called wire-frame model, which models surfaces of rendered objects with boundary polygons of faces approximating a given object. Such model of a set of tori is in Figure 6. Because the rendered parts of faces are only their edges (boundary) this model gives rise to the hidden line problem, where invisible parts of lines obscured by surfaces are sought. The result is that the visible parts of surfaces are represented by visible parts of lines on the boundary of approximating faces. In this setting hidden line problem is different from hidden surface problem which attempts to solve the visibility in each point of the object surface. The latter setting, in the case when an object is approximated by polygonal faces, means that the whole area of polygon is rendered and its visibility is decided.

The algorithm proposed in this article attempts to give a fast solution specialized to the above mentioned hidden line or wire-frame model problem. Then it is natural to ask whether it has a sense to deal with such simplified model any more, facing todays very fast computers on which for example Z-buffer (realized usually in hardware) which is a typical hidden surface technique can be reasonable fast. The response is definitively positive because there are at least two important situations where the wire-frame model is fully justified. It is common practise, that users of graphics systems when they want to navigate through complex scene they switch to wire-frame model because this gives them

[1]Tomáš Hrúz, Slovak Technical University, Faculty of Mechanical Engineering, Department of Automatic Control and Measurement Námestie Slobody 17, 812 31 Bratislava, Slovak Republic, Phone: +42-7-3594-571, 497193 e-mail: hruz@vm.stuba.sk Fax: +42-7-495315

[2]Ivo Považan, Institute of Control Theory and Robotics, Slovak Academy of Sciences, Dubravská cesta 9, 842 37 Bratislava, Slovak Republic, Phone: +42-7-3782985, e-mail: utrrpova@savba.savba.cs

[3]Richard Gosiorovský, Slovak Technical University, Faculty of Mechanical Engineering, Námestie Slobody 17, 812 31 Bratislava, Slovak Republic, Phone: +42-7-497193, 493041/ext.280, Fax: +42-7-495315, e-mail: gosiorov@vm.stuba.sk

much higher speed than the full rendering. The latter beeing still far from realistic speed on todays fastest graphics workstations. The authors have observed that users in such situations often use only wire-frame model without hidden line removal because neither hidden line removal algorithms deliver the sufficient speed. This problem is going to be really critical when the resolution of raster device approaches few thousands times few thousand which is the case of printing industry.

In the following sections, the article focuses mainly on the hidden line problem. There are two fundamental approaches to the hidden line removal. The first approach is called object space approach and relies in various ways to represent objects in the scene by discrete combinatorial representation. The basic objects used are geometrically defined subsets of Euclid space R^3 (i.e. points, edges, polygons, polyhedra e.t.c.). The brute force method of this kind compares every object with other objects and finds the visible resp. invisible parts of lines. The computational complexity is $O(n^2)$. The object space class of methods are improved from early sixties [1, 4]. Recently, by methods of computational geometry, new results have been obtained mainly with respect of the notion of output sensitivity. In [2] a simple output sensitivity method for triangles is presented which runs in time $O(n\sqrt{k}\log n)$ where k is combinatorial complexity of the output visibility map and n is the number of edges. In [3] the method is further improved to solve also the cyclic overlap and intersection of polygons. The computational complexity is $O((n+k)\log n)$. The hidden line problem for isothetic parallelepipeds is solved in [6] by object space method with complexity $O(n\log^2 n + d\log n)$ where d is a number of edges of the display.

The second fundamental approach (the image space approach) is to solve the visibility in each pixel of image space. A straightforward way relies in examining all objects and determining which of them is the closest to the viewer on the projector passing through a given pixel. Image space methods are often realized in hardware. The Z-buffer method is a widely used representative of this class but it has a drawback in the case of hidden line removal that it must process also the interior of the face, therefore when the number of objects and the pixel resolution is large the computational cost can be very high.

The above classification is only a basic pattern. Many algorithms combine above approaches (for survey on hidden line problem see [1, 4]). The algorithm proposed here is in essence an image space algorithm but the structures used are more rich compared to the Z-buffer.

In the following parts, we analyze the proposed algorithm under the assumption that the priority order of faces is given. However, when reasonably simple preprocessing (sorting) is possible, it is sufficient to add the term $n\log n$ to get the overall complexity $O(n\log n + sn\log s)$. On the other hand, for Z-buffer we obtain $O(s^2n)$ when the resolution is taken into the account.

The comparision shows that in the situations when preprocessing can be neglected and the resolution is very high the proposed algorithm can be better than Z-buffer because the latter must process the whole area of a given face whereas the proposed algorithm processes only the boundary. This conclusion is also supported by the results of experiments resumed in Figure 7 resp. 8. They show that when the mean area of face is larger than 64 pixels (the number is specific for the architecture used - in our case MIPS DEC5000) the proposed algorithm is faster. 64 pixels represent about 0.01% of the screen with resolution 1024x1024.

The article is organized in the following way. In section 2 one of the generally accepted models of the scene without cyclic overlaps and intersections is recalled. Section 3 describes the abstract data structures that are used together with their implementation. The main technical lemma about the worst case of a UNION operation on segment trees is derived. Section 4 presents the algorithm and the main theorem about the worst case of the algorithm is proved. The computational experiments are described in Section 5. Section 6 comments our development and suggests some further directions of research.

2 The model of scene

Let us denote by $\mathcal{P}^3$ the input set of convex polygons in 3D space without cyclic overlap and intersections. It is supposed that priority order is known so the input set can be enumerated as:

$$\mathcal{P}^3 = (P_1^3, P_2^3, \ldots, P_n^3)$$

where polygon P_1^3 is the front polygon. According to our approach polygons are processed subsequently from P_1^3 to P_n^3 (a front-to-back order).

Because priority order is known the following representation of our problem can be established: the polygons are projected to the viewing plane where we obtain a sequence of overlapping two dimensional convex polygons in plane

$$\mathcal{P}^2 = (P_1^2, P_2^2, \ldots, P_i^2, \ldots, P_n^2).$$

The visibility priority assigned to polygon P_i^2 equals i. The above representation is also called $2\frac{1}{2}D$ model [10].

The projection of vertices of polygons P_i^3 to the viewing plane determines the projection of a whole polygon. The (x,y)-coordinates of projected vertices are scan converted to the raster space integer coordinates. By this simple preprocessing a sequence of polygons characterized by integer coordinates of their vertices is obtained. The sequence represents a front-to-back visibility order. The above described sequence is an input to the algorithm CU which is introduced together with its data structures in the following sections.

3 Data structures for hidden line problem

3.1 Abstract data structures

Raster space is a set of ordered pairs (x, y) of integers defined as $RS(X_{res}, Y_{res}) = \{(x, y) \in Z^2 \mid 0 \leq x \leq X_{res}, 0 \leq y \leq Y_{res}\}$, where Z is the set of all integers.

The basic data structures called I-structures and C-structures are schematically illustrated in Figure 1. *I-structure* is an abstract data structure representing a union of a finite system of disjointed closed intervals on real line with integer end-points. I-structures are denoted by the capital letters S, R, Q. End-points and intervals belonging to some I-structure are denoted in the following way:

$$S = \bigcup_{i=1}^{n} \langle l_i^S, r_i^S \rangle_i^S.$$

The sense of the above notation is that l_i^S means the left end-point of the $i-th$ interval of the I-structure S. Sometimes an I-structure is treated as a numbered sequence of integer pairs.

UNION operation is defined on the set of all I-structures. Let S, R are I-structures, then the UNION operation is defined as I-structure T representing the minimal system of closed intervals with integer boundaries equal to the set union of interval systems represented by S and R.

In addition to the UNION operation also an operation BUILD is needed. Operation BUILD works on special I-structures S, R where $S = \langle l^S, r^S \rangle$ contains only one interval or S is empty and R contains exactly one degenerated interval $\langle l^R, l^R \rangle$. When S is empty the result of BUILD is $\langle l^R, l^R \rangle$, otherwise the result is I-structure which represents a set union of the intervals $\langle l^S, r^S \rangle$ and $\langle r^S, l^R \rangle$ when $r^S \leq l^R$ resp. $\langle l^S, r^S \rangle$ and $\langle l^R, r^S \rangle$ when $r^S > l^R$.

The purpose of BUILD operation is to create an interval from two points. This situation takes place in algorithm CU when an atomary contour for a convex polygon is built (the definition of an atomary contour follows).

C-structure is an ordered pair of sequences of I-structures $C = (S^x, S^y)$. The first sequence is related to the x-axis of raster space, the second one to the y-axis of raster space. C-structures are denoted by capital letters. Subscript i and superscript x like in S_i^x means an I-structure at i-th position of the x-axis sequence. S^x resp. S^y sequence has exactly X_{res} resp. Y_{res} I-structures. In the case of the capital letter without indices, it is clear from the context whether an I-structure or C-structure is meant. On the set of all C-structures a UNION operation is defined as performing the UNION operations on all corresponding I-structures.

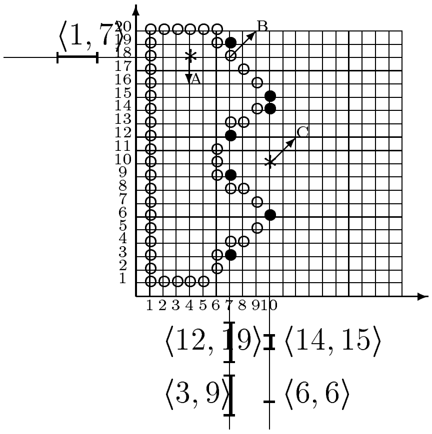

Figure 1: The figure illustrates I-structures and C-structures. Information about point B is not contained in the x-axis sequence of a C-structure, this is the reason for use of the y-axis sequence. By means of a C-structure point A can be identified as inside and point C as outside the region. For example, $S_7 =< 3, 9 > \cup < 12, 19 >$, $S_{10} =< 6, 6 > \cup < 14, 15 >$ are I-structures.

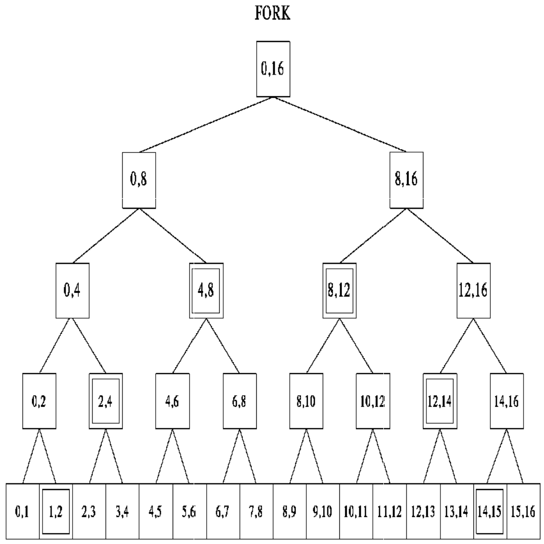

Figure 2: The segment tree for universum $< 0, 16 >$. The interval embedded is $< 1, 15 >$. Allocation of an interval is emphasized with double box.

Let a convex polygon in the plane R^2 is given. *Atomary contour* is the C-structure $Q = (S^x, S^y)$ approximating a boundary of such region where the following condition holds: if point (i, j) of raster space is in this region then j must be a member of some interval $\langle l, r\rangle_k^{S_i^x}$ and at the same time i must be a member of some interval $\langle l, r\rangle_m^{S_j^y}$. The atomary contour of a convex polygon can be described also constructively. Firstly, the edges of the polygon are scan converted to raster space. Because of a convexity, the polygon can be represented by simple interval (span) on every x and y line of raster space. In fact, this is the method which is used in algorithm CU. To generate the corresponding I-structures operation BUILD is used.

Atomary contours form a subset of C-structures. *Contour* is atomary contour or a union of some contour with atomary contour.

3.2 Implementation

Now one can look for algorithms and data structures which perform the UNION operations on C-structures in the most effective way. This leads to the elementary operation of range searching of the point $l_k^{S_j^x}$ in I-structure $R_j^x = \bigcup_{i=1}^n \langle l_i^{R_j^x}, r_i^{R_j^x}\rangle$. A variety of methods can be used. Algorithm CU performs UNION operation between general I-structure with n intervals and a simple I-structure consisting of one interval. A brute-force method is to use a simple list of intervals where the worst case complexity of the UNION operation is $O(n)$ for an I-structure with n intervals.

Another method, which is more efficient, is to use a variation of data structure called *segment tree* [6]. An example of segment tree is in Figure 2. In the following section, an instance of the segment tree structure together with a basic operation UNION is described.

Segment tree is a highly flexible and efficient data structure for performing various operations on interval systems. Usually it is used to maintain interval systems on real line to support a solution for various tasks from computational geometry. In the following brief description the basic features of segmented trees are revised [6] to the point where a new operation UNION on a segment tree is defined.

The *segment tree* is a binary tree. Every node carries an information about some interval with integral boundaries. These so called *standard intervals* become smaller and smaller in the divide and conquer fashion going down in the tree. Then any other given interval can be decomposed effectively to these intervals (when it does not exceed the basic interval covered by the segment tree). The segment tree is a static decomposition of some basic interval called *universum*. The dynamics is in various operations on systems of subintervals of the given universum. Accurately, let $T(l,r)$ denotes a subtree with an interval $< l, r >$; $l < r$ assigned to the root. Let functions $B(v) = l, E(v) = r$ return the left and the right boundary of an interval assigned to the node v. The tree is defined recursively, in a given node v the left subtree is given as $T(l, \lfloor (B(v)+E(v))/2 \rfloor)$ and the right subtree is given as $T(\lfloor (B(v)+E(v))/2 \rfloor, r)$ if $E(v) - B(v) > 1$ otherwise a leaf has been reached. The roots of these subtrees are denoted $S^l(v), S^r(v)$. Every level of segment tree defines a partition of the universum in a straightforward way: for example excluding the left boundary from every standard interval in that level.

The segment tree is balanced where all leaves belong at most to two contiguous levels. The depth of the segment tree is $\lceil \log_2(r-l) \rceil$ because the leftmost (the shortest in the tree) path represents in fact recursion $r_{n+1} = \lfloor r_n/2 \rfloor, r_0 = r - l$ which reaches value 1 exactly in $\lfloor \log_2(r-l) \rfloor$ steps but the tree can be one level deeper when $\log_2(r-l) > \lfloor \log_2(r-l) \rfloor$.

The basic option how to use a segment tree is to store intervals (belonging to some universum) in a dynamic way i.e. supporting insertion and deletion operations. The operation $INSERT(b, e, root(T))$ does a partitioning of a given interval $< b, e >$ in the fashion described by the algorithmic primitive in Figure 3.

The insert operation $INSERT(b, e, root(T))$ does a walk in T with the following structure: there is an initial path P_I (can be empty) from the root to a node v_f called *fork*,a left path P_L from the fork and a right path P_R from the fork.

```
procedure INSERT(b,e,v)
begin if (b ≤ B(v)) and (E(v) ≤ e) then
            allocate <b,e> to v
      else
            begin
            if (b < ⌊(B(v) + E(v))/2⌋) then
                  INSERT(b,e,S^l(v));
            if (⌊(B(v) + E(v))/2⌋ < e) then
                  INSERT(b,e,S^r(v));
            end
end.
```

Figure 3: Algorithmic primitive INSERT

Either the interval $< b, e >$ is allocated entirely to fork (than P_L, P_R are empty and embedded interval must be equal to some standard interval) or all right sons of nodes of P_L and all left sons of nodes of P_R define the fragmentation of the interval except the following two degenerated cases: P_L consists of exactly one node and P_R is not degenerated or P_R consists of exactly one node and P_L is not degenerated.

To see the above structure of operation $INSERT(b, e, root(T))$ the notion of *cut* can be used. The partition of universum corresponding to level l *cuts* the interval $< b, e >$ k times when k of the boundary points of standard intervals of this partition are in the interior of $< b, e >$. A definitoric level l_∞ cuts from the definition every subinterval of the universum exactly once. The set of levels which cut the interval $< b, e >$ is not empty. Let l_m is the minimal level from this set. l_m cuts the processed interval exactly once (when it cuts twice exactly one point vanishes in lower level and this is a contradiction with minimality of l_m). The fork is one level lower (see Figure 2). In every level lower than l_m the interval $< b, e >$ is contained in some standard interval therefore the algorithm primitive $INSERT(b, e, root(T))$ (Figure 3) subsequently enters the recursion until the fork is found. So the existence and uniqueness of fork has been obtained.

If the processed interval equals not to any standard interval then the both paths P_L, P_R are not empty.

Once a fork is obtained the left path P_L can be investigated. The analysis of the right path P_R is analogical. Left path P_L can be built as a result of the application of

```
function UNION(b,e,v)
10  begin
20  if (U(v)) then return (U(v))
30  if (b ≤ B(v)) and (E(v) ≤ e) then
40        begin
50              return(U(v) = true)
60        end
70  else
80        begin if (b < ⌊(B(v) + E(v))/2⌋) then
90                    begin
100                   l = UNION(b, e, S^l(v))
110                   end
120             else
130                   begin
140                   l = U(S^l(v))
150                   end
160             if (e > ⌊(B(v) + E(v))/2⌋) then
170                   begin
180                   r = UNION(b, e, S^r(v))
190                   end
200             else
210                   begin
220                   r = U(S^r(v))
230                   end
240             return(U(v) = l and r)
250       end
260 end.
```

Figure 4: Algorithmic primitive UNION

$INSERT(b, E(S^l(fork(T))), S^l(fork(T)))$ to the subtree starting at the left son of fork. The interval processed now is $< b, E(S^l(fork(T))) >$. Here the situation is more special because the right end-point of processed interval equals to the right end-point of the universum. This leads to the fact that either immediately the root is allocated (of the subtree starting at the left son of fork) and this branch ends or the algorithm goes deeper in the tree and looks for fork. When the fork is found the right son will be immediately allocated and the process follows through the left branch.

The above analysis of operation $INSERT$ shows that the worst case complexity of operation $INSERT$ is proportional to the depth of segment tree $\lceil \log_2(r-l) \rceil$.

If it is wanted to store more intervals into the segment tree a nonnegative integer $C(v)$ can be assigned to every node v which is incremented whenever $< b, e >$ is allocated to v. The applications of segment trees usually differ in a way how an allocation of $< b, e >$ to a given node v is made and what information is maintained in a node. For example the deletion operation has the same structure as the insert operation differing only in the allocation operation (step 20).

Now we are in the position to tailor the segment tree data structure for the purpose of hidden line problem. The goal is to use segment tree to maintain I-structures and to do UNION operation on them as it was defined in the previous section. In fact the basic building block is the UNION operation between an I-structure already in the tree and one interval. A logical variable $U(v)$ is assigned to every node and the allocation of an interval to a node consists simply in a setting this variable to true. The operation $UNION(b, e, root(T))$ is defined by the algorithmic primitive in Figure 4.

The operation $UNION(b, e, root(T))$ realizes the following two ideas:

1. the algorithm proceeds down into the tree in the same way as during the INSERT but if a node already allocated is encountered the process stops and returns to higher levels.

2. during the returning process the algorithm inquires whether the both sons are allocated. If they are allocated, the current node is also allocated.

It must be remarked that the algorithmic primitive presented is not meant to be efficient in implementation. The main concern has been simplicity of presentation and of comparison with operation INSERT. Authors have designed several efficiency improvements for the concrete implementation of algorithm for UNION operation. There is also a question how to establish a visibility of certain end-point of embedded interval, but this can be solved by straightforward search down in the tree along the left-most respectively right-most path.

Lemma 1 *The worst case complexity of the operation $UNION(b, e, root(T(l, r)))$ is $O(\lceil \log_2(r-l) \rceil)$.*

Proof. From the definition of the operation $UNION(b, e, root(T(l, r)))$ it follows that when an interval disjoint to the I-structure already in the tree is processed the operation has the same course as the operation $INSERT(b, e, root(T))$. But when the processed interval is intersecting the I-structure already in the tree then the operation is shorter.

On the other hand all processed intervals can be the same and of unit length so that every UNION operation reaches maximal depth of the segment tree. $\boxed{\text{QED}}$

It is straightforward to see that when uniform distribution on the input intervals is assumed the asymptotical average case complexity is $O(1)$ because after enough number of input intervals the root should be allocated. Then all subsequent $UNION$s have the complexity $O(1)$. But the uniform random distribution on intervals is surely not describing average hidden line problem properly. This extremal example only supports the suggestion that in real applications the complexity of the operation $UNION(b, e, root(T(l, r)))$ is better than the worst case $O(\log_2(r - l))$. The feature that the algorithm never goes deeper in the tree when an allocated node has been reached is responsible for the output sensitivity of the whole algorithm.

4 The algorithm

The input to the algorithm is a visibility ordered sequence of overlapping two dimensional convex polygons in the plane:

$$\mathcal{P}^2 = (P_1^2, P_2^2, \ldots, P_i^2, \ldots, P_n^2).$$

Algorithm CU.

1. Initialize the contour $Q = (S^x, S^y)$, $i = 1$.

   Comment: After initialization every I-structure of the sequences S^x, S^y of the C-structure Q is empty.

2. Take a polygon P_i^2 from the sequence $\mathcal{P}^2$.

3. Initialize the atomary contour $B = (A^x, A^y)$. Interpolate to generate the points $p_i = (x_i, y_i)$ of the boundary of the polygon. For every point $p_i = (x_i, y_i)$ obtained, perform the following two steps:

   (a) Use the BUILD operation on I-structure $A^x_{x_i}$ and I-structure $\langle y_i, y_i \rangle$.

   (b) Use the BUILD operation on I-structure $A^y_{y_i}$ and I-structure $\langle x_i, x_i \rangle$.

   Comment: Bresenham interpolator is used for the generation of the raster space points $p_i = (x_i, y_i)$. When this step of the algorithm CU is finished a complete atomary contour is built. Usually Bresenham interpolator works so that raster space points coordinates are subsequently generated and frame buffer is updated in these points by means of some $set\_pixel(x, y)$ operation. In our case $set\_pixel(x, y)$ operation is replaced by the BUILD operation.

4. Perform the UNION operation between the contour Q and the atomary contour B.

   Comment: This requires use of the UNION operation on each I-structure. The UNION operation contains range searching of boundary points of intervals of the atomary contour B in interval systems of the contour Q. As a side affect of the former the information whether these points are in some interval of the contour Q or not is obtained. If the boundary points of intervals of B are not in any interval of Q, the value in frame buffer is updated. In fact this is the only place in the algorithm where graphics operations in frame buffer can be done.

5. $i = i + 1$

6. If $i \leq n$ go to step 2.

7. end of CU.

To analyse the worst case complexity of algorithm CU denote with $s = max(X_{res}, Y_{res})$ the maximum of the raster space resolutions in both axes and with $d(\mathcal{P}^3)$ the following quantity for a given scene $\mathcal{P}^3$:

$$d(\mathcal{P}^3) = \max_{P \in \mathcal{P}^3} 2((x_{max} - x_{min}) + (y_{max} - y_{min}))$$

where x_{max}, x_{min} are the minimal x-coordinate and the maximal x-coordinate of polygon P, analogically y_{max}, y_{min}.

As was mentioned in the introduction the algorithm CU is output sensitive as can be seen from the following: When there is a polygon which hides all other polygons then all I-structures processed throughout the run of algorithm CU consist of simple intervals. Therefore the complexity of the UNION operation is lower in this case. On the other hand, to establish accurately a notion of output sensitivity in raster space which can be used in all circumstances is not simple. Here only some possibilities are mentioned which need more investigations to decide between them. Another aim of this paragraph is to motivate the expression of the complexity of operation UNION as a function $f(t,s)$ of some output sensitivity parameter t and a resolution s. If the output sensitivity parameter t is defined as a maximal number of intervals in any I-structure throughout the whole computation and UNION operation is implemented on I-structures as a simple list of intervals then the function $f(t,s)$ equals t. Another possibility is to implement UNION operation on a segment tree and to define the output sensitivity parameter t as a maximal depth in any tree achieved throughout the whole computation. The function $f(t,s)$ then takes the form $f(t,s) = t \leq \log_2 s$.

Theorem 2 *Let $\mathcal{P}^3 = (P_1^3, P_2^3, \ldots, P_n^3)$ is a system of n non-intersecting planar convex polygons without cyclic overlap and let the front-to-back priority order of the system $\mathcal{P}^3$ is known. Then hidden line problem can be solved in time proportional to $O(dnf(t,s))$ where $d = d(\mathcal{P}^3)$ and $f(t,s)$ is the worst case complexity of UNION operation throughout the whole computation of a scene as a function of resolution s and output sensitivity parameter t.*

Proof. It follows from the definition of algorithm CU that an upper bound on the time to produce atomary contour equals $O(d)$ because Bresenham interpolator is used to obtain an atomary contour of a polygon. The upper bound on UNION operation between atomary contour and contour is $O(\frac{d}{2}f(t,s))$ of time units because the atomary contour contains no more than $\frac{d}{2}$ of I-structures and the UNION operation on intervals has a worst case complexity $O(f(t,s)$.

Finally, for n polygons the upper bound on complexity of the algorithm CU is:

$$O(nd + n\frac{d}{2}f(t,s))$$

$\boxed{\text{QED}}$

Adding the worst case complexity for UNION operation achieved with the segment tree implementation the following corollary can be derived:

Corollary 3 *Let the UNION operation for an I-structure with a given interval is implemented on a segment tree as is described in section 3.2. Then the worst case complexity for algorithm CU is $O(nd + n\frac{d}{2}t)$ where $t \leq \log_2 s$ is a maximal depth of any allocation in all segment trees achieved throughout the whole computation of the scene.*

Proof. The universum for every segment tree is a subinterval of the interval $< 0, s >$ and therefore according Lemma 1 the worst case complexity of the UNION operation is $f(t,s) = O(\log_2(s))$. Denoting by t the maximal depth of any allocation achieved in all segment trees throughout the whole computation, it is obtained that $t \leq \log_2(s)$ and $f(t,s) = O(t)$. The parameter t expresses the output sensitivity in raster space sense. $\boxed{\text{QED}}$

It holds that $d \leq s$, $t \leq \log_2 s$. Then the complexity of algorithm CU can be expressed as $O(sn \log_2 s)$. If the resolution is considered to be constant and the number of polygons is very high ($n \gg s$), the running time is $O(n)$.

5 Experimental Results

In the previous section was shown that if priority order is known algorithm CU is asymptotically very fast. To see the practical importance of this algorithm an experimental study has been made which compares algorithm CU with Z-buffer. Z-buffer is very simple algorithm and reasonably fast, therefore it is often realized in hardware. So the comparison with Z-buffer gives a relevant information about practical value of the proposed algorithm.

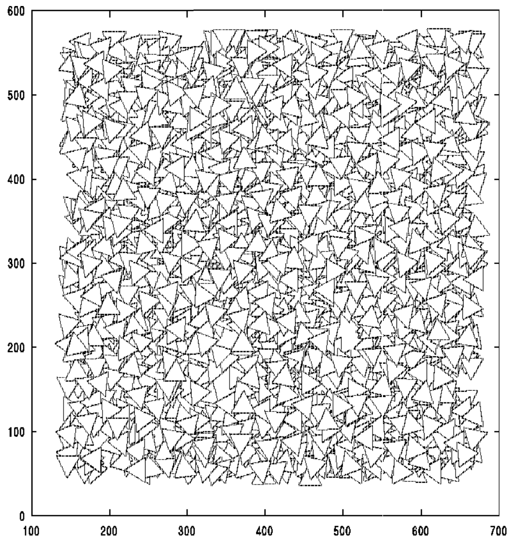

Figure 5: A scene consisting of a set of 2000 randomly positioned triangles with area 400 pixels.

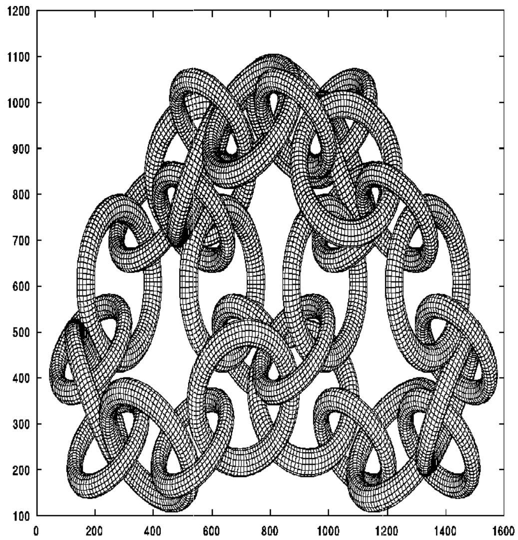

Figure 6: A scene consisting of 40000 planar polygons approximating a set of tori in 3D space.

As the following experiments show, Z-buffer is slower than algorithm CU when the faces have larger area. The reason is that Z-buffer must process the whole area of the given face whereas algorithm CU processes only the boundary. The above argument holds only when the preprocessing complexity can be neglected and at the same time resolution is taken into the account. However, these circumstances can occur in certain practical situations as is shown by experiments.

The experiments has been made in Lab-3D interactive software system developed at authors site on DEC5000/240 RISC system. The experiment consists of two different scenes. The first scene consists of randomly positioned triangles in 3D space. The parameters of the scene are the number of polygons and the area of the polygons. This scene is used for analysis of overall features of algorithm CU and for comparison with Z-buffer. An instance of such scene is in Figure 5 consisting of 2000 triangles with constant area of 400 pixels. The results of the experiments are summarized in Figure 7. Firstly, a series of experiments was made to identify an area of triangle for which algorithm CU has approximately the same speed as Z-buffer. This area was about 64 pixels. Then the experiments was made with larger area of 400 pixels and smaller area of 20 pixels. The Figure contains two curves for each triangle area, one for algorithm CU and one for Z-buffer. The result is that if the area is larger than 64 pixels, algorithm CU is faster. In the case of 400 pixels (about 0.04resolution 1024x1024) CU is already much faster than Z-buffer.

The second scene which is on Figure 6 consists of a set of tori in 3D space. The parameter of this scene is a growing number of faces approximating the same set of tori, while the total area remains constant. This series of experiments compares various implementations of UNION operation on interval systems. Also, the output sensitivity of the proposed algorithm is tested on this scene. The results are summarized in Figure 8. There is one curve for brute-force implementation with linear lists of intervals denoted with "Contour Union 2" and one curve for implementation on augmented segmented trees denoted with "Contour Union 1". To test the output sensitivity features of the al-

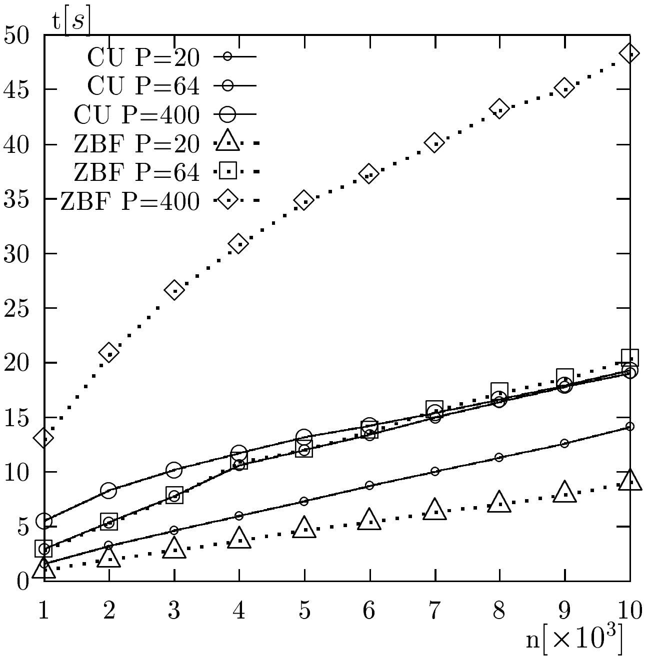

Figure 7: The Figure summarizes experimental results on the scene in Figure 5. It contains two curves for each triangle area, one for algorithm CU and one for Z-buffer. CU denotes algorithm CU, ZBF denotes Z-buffer and P denotes area of triangles in pixels.

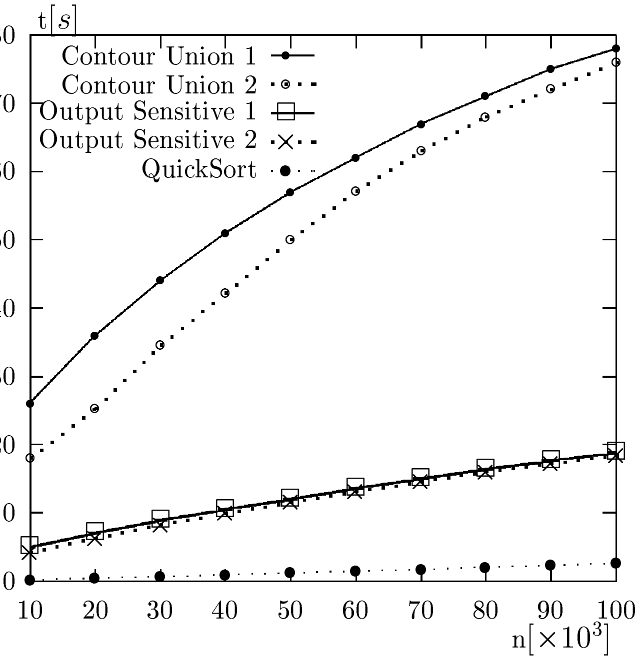

Figure 8: The Figure summarizes the results of the series of experiments with various implementations of UNION operation on the scene in Figure 6.

gorithm the same scene has been hidden with one large square face. The results of the experiment series for both implementations of UNION are in the same figure. The sorting, which is an inevitable part of front-to-back class of algorithms, is not influential even if the scene contains number of faces of order 100000 as can be seen from the last curve in Figure 8, which shows only the sorting time whereas other curves show the total time.

6 Conclusions

The experimental data and the worst case analysis of the proposed algorithm show that for the scenes with larger number of faces and higher resolution interesting speed-up can be achieved. On the other hand the justification of the proposed algorithm depends on the scene complexity and particular system architecture. It is reasonable in practical situations to have at the disposal few different hidden line algorithms. In this case, the proposed algorithm seems to be a good candidate.

The technique presented, contains a natural parallelism suited for massively parallel machines. This is because operations on I-structures can be done independently of each other. When for example a processing element is allocated to every I-structure in the C-structure (i.e. the number of allocated elements equals $X_{res} + Y_{res}$) then the atomary contour is generated in time $O(1)$.

It can be suggested (mainly relying on the [11]) that faster implementations of abstract UNION operation are possible. The worst case complexity $\log_2 n$ where n is the number of intervals instead of having $\log_2 s$ where s is the resolution can be expected. Also the behaviour of the segment tree implementation in an average problem would need a further research.

The algorithm presented in this paper could be improved to solve intersections of polygons resp. cyclic overlap. On the other hand when the scene is face coherent [4], the brute-force sort can be sufficient to obtain a good solution. Because of a high speed of algorithm CU it is possible in real application to approximately solve the intersection problem by firstly decomposing the polygons to much smaller parts and than relying on the face coherency.

As was mentioned earlier algorithm CU is output sensitive. But the deeper understanding of output sensitivity in raster space sense would need further investigation. What is clear is that the output sensitivity of algorithm CU is weaker than the output sensitivity of the algorithm from [2] because all edges are virtually interpolated. On the other hand scan conversion is incorporated into the heart of algorithm CU therefore the algorithm CU can be accurately compared with object space algorithms only if the scan conversion cost is added to object space algorithms.

Because algorithm CU is output sensitive the method of grouping introduced in [2] could be used to improve its performance. But the concrete way how to choose appropriately the groups would need a further research.

References

[1] James D. Foley, Andries Van Dam, Steven K. Feiner, John H. Hughes, Computer Graphics, Principles and Practise, second edition, Addison Wesley, Reading, Mass., November 1991.

[2] M. Sharir, M. H. Overmars: A Simple Output-Sensitivity Algorithm for Hidden Surface Removal, ACM Transactions on Graphics, Vol. 11, No. 1, January 1992, pp. 1-11.

[3] M. de Berg, M. H. Overmars: Hidden surface removal for c-oriented polyhedra, Computational Geometry: Theory and Applications, No. 1, 1992, pp. 247-268.

[4] I. E. Sutherland, R. F. Sproull, and R. A. Shumacker: A characterization of ten hidden-surface algorithms, Computer Surveys, Vol. 6, No. 1, March 1974,pp. 1-55.

[5] Donald E. Knuth, The Art of Computer Programming. Vol. III: Sorting and Searching, Addison Wesley, Reading, Mass., 1973.

[6] Franco P. Preparata, Michael I. Shamos, Computational Geometry, Springer Verlag, 1985.

[7] F. P. Preparata, J. S. Vitter and M. Yvinec: Computation of the Axial View of a Set of Isothetic Parallelepipeds, ACM Transactions on Graphics, Vol. 9, No. 3, July 1990, pp. 278-300.

[8] H. Fuchs, Z. M. Kedem and B. F. Naylor: On visible surface generation by apriori tree structures, Computer Graphics (proc. SIGGRAPH) July 1980, pp. 124-133.

[9] J. E. Steinhart: Scanline Coherent Shape Algebra, Graphics Gems II, edited by Jim Arvo, Academic Press 1991.

[10] J. P. Braquelaire and P. Guitton: $2\frac{1}{2}D$ Scene update by insertion of contour, Computers & Graphics, Vol. 15, No. 1, 1991, pp. 41-48.

[11] Zvi Galil, G. F. Italiano: Data Structures and Algorithms for Disjoint Set Union Problems, ACM Computing Surveys, Vol. 23, No. 3, September 1991, pp. 319-344.

[12] Dan Gordon and Shuhong Chen: Front-to-Back Display of BSP Trees, IEEE Computer Graphics & Applications, Vol. 11, No. 5, September 1991, pp. 79-85.

[13] T.Y. Kong and A. Rosenfeld: Digital Topology: Introduction and Survey, Computer Vision, Graphics and Image Processing, Vol. 48, 1989, pp. 357-393.

[14] V.A.Kovalevski: Finite Topology as Applied to Image Analysis, Computer Vision, Graphics and Image Processing, Vol. 46, 1989, pp. 141-161.

[15] C.A. Wuthrich and P. Stucki: An Algorithmic Comparison between Square- and Hexagonal-Based Grids, CVGIP: Graphical Models and Image Processing Vol. 53, No.4, July 1991, pp. 324-339.