A COMPUTATIONAL GEOMETRIC APPROACH TO VISUAL HULLS

SYLVAIN PETITJEAN
CHIM-CONS & INRIA Lorraine, Bâtiment LORIA, BP 239,
54506 Vandœuvre-lès-Nancy cedex, France (petitjean@loria.fr)

Received 7 August 1996
Revised 28 January 1997
Communicated by Ming C. Lin and Dinesh Manocha

ABSTRACT

Recognizing 3D objects from their 2D silhouettes is a popular topic in computer vision. Object reconstruction can be performed using the volume intersection approach. The visual hull of an object is the best approximation of an object that can be obtained by volume intersection. From the point of view of recognition from silhouettes, the visual hull can not be distinguished from the original object. In this paper, we present efficient algorithms for computing visual hulls. We start with the case of planar figures (polygons and curved objects) and base our approach on an efficient algorithm for computing the visibility graph of planar figures. We present and tackle many topics related to the query of visual hulls and to the recognition of objects equal to their visual hulls. We then move on to the 3-dimensional case and give a flavor of how it may be approached.

Keywords: Object reconstruction, volume intersection, visual hulls, visibility graphs, visibility complexes

1. Introduction

The silhouettes of a 3D object have been indicated as effective psychophysical clues to shape understanding. One possible approach to object reconstruction from silhouettes is called volume intersection. It recovers a volumetric description of the object from different silhouettes by intersecting the solid regions (cones) of space to which the object is constrained to lie by each silhouette. As the number of viewpoints (and thus the number of cones) increases, the object is reconstructed with higher precision. But even for an infinite number of viewpoints constrained to lie in a given viewing region, it may well be that the largest object that we are able to reconstruct is different from the original object. This leads to the notion of visual hull of an object with respect to the given region, which is both the best approximation of the object one can get from this region and the largest solid silhouette-equivalent to the original object.

The theory of visual hulls has been given recently a firm footing. It raises many new and interesting questions. Given a model of an object and a viewing
region, how does one construct its visual hull? What type of objects are equal to their visual hull? What objects can be exactly reconstructed from silhouettes? What objects can be exactly reconstructed in a finite number of steps? How many viewpoints does it take to be reconstructed? Which parts of an object influence its silhouettes?

Some of these questions have started to receive answers recently, notably through the works of Laurentini.\(^{11,12,13}\) Among other things, the author presents properties of visual hulls with respect to different viewing regions. The first interesting case happens when viewpoints are constrained to lie outside the convex hull of the scene, giving birth to the external visual hull. If viewpoints are free to lie in the entire space minus the scene itself, the largest reconstructible object is called the internal visual hull. Laurentini presents algorithms for constructing (internal and external) visual hulls of 2D polygonal objects and moves on to generalize the procedure to 3D polyhedral solids. He also develops a procedure for exactly reconstructing a 3D object using a finite number of viewpoints (assuming the object can of course be reconstructed so).

It is our belief that tools developed within computational geometry can help tackling some of the issues introduced by visual hulls. In this paper, we aim at proposing answers to some of the above questions using such tools. We place ourselves first in flatland, which represents a good testing ground for possible generalizations to the 3D case. We present algorithms for computing the visual hulls of general (curved, singular) planar figures using visibility graphs, with better worst case complexities than those of Laurentini. We look at what happens when reconstructing a curved object and relate the reconstruction error to the number of viewpoints used. We finally indicate a number of possible extensions and/or connections to other fields of computational geometry.

The rest of this paper is as follows. In Section 2, we review some definitions, properties and past work on visual hulls. Section 3 establishes the link with visibility graphs, reviews some past work on tangent visibility graphs and extends an existing algorithm to handle more general curved singular objects. The actual construction of (external and internal) visual hulls is presented in Section 4 and time complexity bounds are given. Section 5 indicates different connections with other fields within or outside computational geometry. Some indications on how the 3D case can be tackled are given in Section 6, before concluding.

2. Visual Hulls: Definitions, Properties and Past Work

We start in this section by giving precise definitions of different kinds of visual hulls and a few properties derived from the definitions. Note that though reconstruction by volume intersection was proposed more than twenty years ago,\(^3\) a precise formulation of what can be reconstructed (and hence of the concept of visual hull) is due to the recent works of A. Laurentini.\(^{11,12,13}\) This section thus uses much of the notation and terms introduced by this author.

For the rest of this paper, unless otherwise noted, we assume that the scene considered is in general position.
2.1. Visual Hull with Respect to a Region

In what follows, let $S$ be an object of the Euclidean space $E^k$ (where $k = 2, 3$ for our purpose, but there is no need to make restrictions here). Let also $R$ be a viewing region of $E^k$ (i.e., points from which the object is observed).

**Definition 1** The visual hull $\text{Vh}(S, R)$ of an object $S$ relative to the region $R$ is a region of $E^k$ such that for each point $P \in \text{Vh}(S, R)$ and each viewpoint $V \in R$, the half-line starting at $V$ and passing through $P$ contains at least a point of $S$.

Two examples of visual hulls are displayed in Figure 1. We now have the following theorem:

**Theorem 1** The following results characterize visual hulls from the point of view of object recognition:

- $\text{Vh}(S, R)$ is the maximal object that gives the same silhouette as $S$ when observed from any $V \in R$.
- $\text{Vh}(S, R)$ is the closest approximation of $S$ that can be obtained using volume intersection techniques with viewpoints in $R$.

Let now $\text{Ch}(S)$ be the convex hull of $S$. Of particular interest are the following viewing regions: $E^k \setminus \text{Ch}(S)$ and $E^k \setminus S$. In the first case we will talk of the external visual hull of $S$ (or simply the visual hull of $S$) and denote it by $\text{Vh}(S)$. In the second, it will be called the internal visual hull of $S$ and denoted $\text{IVh}(S)$.

The following results are fairly straightforward consequences of the definitions but nonetheless very useful in the actual construction of visual hulls.

**Theorem 2** \cite{11} We have:

1. A point of $E^k$ belongs to $\text{Vh}(S)$ iff any line passing through it contains at least a point of $S$.
2. A point of $E^k$ belongs to $\text{IVh}(S)$ iff any half-line starting at this point contains at least a point of $S$.
3. If $S \in E^2$ is connected, then $\text{Vh}(S) = \text{Ch}(S)$.
4. $S \subseteq \text{IVh}(S) \subseteq \text{Vh}(S) \subseteq \text{Ch}(S)$.
Figure 2: A chain of inclusions. From left to right: the original object, its internal visual hull, its external visual hull and its convex hull.

The fourth item is illustrated on Figure 2.

2.2 Computing Visual Hulls of Polygons in 2D

The use of visual hulls is in 3D object recognition so it may seem strange to consider visual hulls in the plane. This is actually not true for two reasons. First because 2D algorithms can deal with 2.5D sweep solids, as Laurentini pointed out. Second because the methodology we shall use may serve as a model for tackling the 3D case.

2.2.1. External visual hulls

Let us call a line of $E^2$ a visual line if this line does not intersect the object $S$. It is fairly clear that visual lines can be grouped in families, two lines being in the same family if they can be moved continuously one onto the other. Given a point $P$ of the plane its visual number will be the number of families of visual lines through $P$.

Theorem 2, first item suggests that there is a number of lines of the plane (or at least segments of these lines) that act as boundaries of the visual hull. These lines will be called active lines. Active segments (segments of active lines) separate regions of different visual numbers. In addition, the genericity of the scene implies that crossing a boundary of the partitioning means adding one or subtracting one to the visual number. Also, the connected components of the scene $S$ may be seen as regions of visual number 0. So starting with one such region (and knowing whether crossing a boundary means adding or subtracting one), we can iteratively compute all visual numbers.

The idea of the algorithm of Laurentini\cite{Laurentini11} is to partition the plane into regions of constant visual number and to merge those regions with visual number 0 (i.e. the visual hull). An example of such a partition is given in Figure 3. The algorithm described by Laurentini has complexity $O(n^3 + m \log m)$, where $n$ is the number of vertices of $S$ and $m$ is the number of vertices of the partition of the plane by active segments.

It is clear that active lines must be searched among the lines “tangent” to $S$ that do not further intersect $S$ transversally. Actually, only three different cases may
Figure 3: Partition of the plane by active segments for a configuration of three triangles. The visual numbers are displayed.

occur, as shown in Figure 4. In this figure, the active segments are shown as thick lines, while the rest of the active lines are dotted. A + sign indicates the region of higher visual number.

Figure 4: Lines and segments visually active for the construction of the external visual hull.

2.2.2. Internal visual hulls

Laurentini\textsuperscript{11} also describes an algorithm for computing the internal visual hull of polygons. This algorithm is very similar to the above. It starts by defining the notion of \textit{visual half-line} of $E^2$, which is simply a half-line that does not intersect $S$. The \textit{internal visual number} of a point of the plane is then the number of families of visual half-lines going through that point.
The algorithm consists in partitioning the plane into regions of constant internal visual number. For this, the possible configurations of active lines are identified (same as in Figure 4, except that a. is replaced by Figure 5). After the determination of active lines, the partition is traversed as above. This algorithm has also complexity $O(n^3 + m \log m)$.

Figure 5: A new set of active segments for the computation of the internal visual hull.

Note that for the configuration displayed in Figure 3, the internal visual hull of $S$ is equal to $S$. Several comments are in order. First the “continuity” in visual number that was observed in the case of external visual hulls is still valid in free space, but not true as far as crossing the boundary of $S$ is concerned: immediately outside $S$, the internal visual number can assume any integer value as Fig. 6 shows. Second, only the active segments of Figure 5 can actually bound the internal visual hull but the others are needed for computing the visual numbers.

Figure 6: Crossing the boundary of $S$ does not mean adding one to the internal visual number.

Note that the continuity in visual number may not always be true in non-generic situations. If for instance three vertices are aligned, two of which are on opposite sides of their bitangent line, then there will be some partial overlap of active segments and crossing the boundary of such segments may possibly mean increasing the visual number by 2.

2.3. Visual Hulls of Curved Objects

The discussion above extends quite naturally when considering a collection of curved obstacles in the plane. Indeed, it suffices to realize that active lines generalize to one of three types: lines through two vertices, lines through a vertex and tangent
to some object, lines tangent to two objects. We will collectively refer to these lines as **bitangent lines**.

### 3. Visibility Graphs and Visual Hulls

Visual hulls bear remarkable resemblance with a structure that has received a lot of attention: the visibility graph. Indeed, it turns out that the active lines we have defined are particular cases of lines supporting “bitangent” segments that the visibility graph stores. So the rough idea of our algorithm for constructing visual hulls will first be to compute the visibility graph of the scene, prune those bitangents that are not meaningful with respect to visual numbers and then finish as above by partitioning free space and traversing this partition.

In this section, we first review some of the work done on visibility graphs. We then see how to tailor these algorithms to our needs.

#### 3.1. Visibility Graph

Let $\mathcal{S}$ be the union of planar polygonal objects. Let $n$ be the total number of vertices. One can construct a graph whose nodes are the vertices of $\mathcal{S}$ and such that two nodes are connected if the corresponding vertices see each other (*i.e.* if the segment joining the two vertices does not intersect $\mathcal{S}$ transversally). This graph is called the **visibility graph** of $\mathcal{S}$. Asano et al.$^2$ present an algorithm with complexity $O(n^2)$ in both time and space. A practically feasible algorithm is given by Overmars and Welzl$^{10}$ with complexity $O(k \log n)$, where $k$ is the number of arcs of the visibility graph (which may be $O(n^2)$ in the worst case). Finally, Ghosh and Mount$^9$ describe an output-sensitive optimal algorithm with complexity $O(n \log n + k)$.

#### 3.2. Tangent Visibility Graph

The above notions have been generalized for the computation of the **tangent visibility graph** of a collection $\mathcal{S} = \cup \mathcal{S}_i$ of $n$ pairwise disjoint strictly convex plane objects. A **bitangent** is a line segment joining the two points of contact of a line touching the object at two different places. It is called **free** if in addition it does not further meet the object transversally. The endpoints of these free bitangents split the boundary $\partial \mathcal{S}$ of the objects into a number of arcs. The tangent visibility graph is a graph whose nodes are the points joining two arcs and whose edges are the arcs and the free bitangents.

A first algorithm for computing the tangent visibility graph in $O((k + n) \log n)$, where $k$ is the number of arcs of the graph, has been presented by Pocchiola and Vegter.$^{21}$ A second algorithm is given which is optimal in both time - $O(n \log n + k)$ - and space - $O(n)$. These algorithms assume that the complexity of the objects are $O(1)$ (*i.e.* the bitangents of pairs of objects can be computed in constant time).

#### 3.3. Rotational Sweep

The method for computing the tangent visibility graph we shall present here,
which is well suited to our problem, is based on a rotational sweep of free space. In that paper, the authors handle the case of strictly curved objects possessing at most a finite number of inflection points on their boundaries. Here, we extend the method to account for a scene $S$ made of a number of objects $S_i$ pairwise disjoint, each object being the interior of an injective and closed Jordan curve. Finally, for the sake of making things coherent, we suppose the scene $S$ is completely surrounded by a circle (say the circle at infinity).

3.3.1. Notations

We shall refer to the singular points of $\partial S$ as vertices and when a vertex is the endpoint of a non-planar piece of curve, we will call it a curved vertex. The term bitangent will refer to a “generalized” bitangent, i.e., a segment joining two points of the boundary, each point being a vertex or a point of tangency.

A non-singular point of the boundary separating a non-planar piece of curve from a planar one will be called a flat point. The directions of the lines of support of free bitangents, of inflectional tangents, of limiting tangent lines at curved vertices and of tangents at flat points are called critical directions. Given a direction $u$, a non-singular point of the boundary having a tangent of direction $u$ is called an extremal point with respect to $u$. We shall assume that the horizontal direction is not a critical direction. Extremal points with respect to the horizontal direction (extremal points for short), inflection points, curved vertices and flat points cut the boundary of the objects in a number $n$ of arcs that we shall take as measure of the complexity of the scene. Finally, we shall suppose that the objects $S_i$ are in general position, which simply means that no two critical directions with the same direction fall at the same point.

3.3.2. Visual events

Given a non-critical direction $u$, let $V(u)$ be the union of the extremal points with respect to $u$ and of the singular points of the boundary of the objects. The set of maximal free line segments with direction $u$ and containing a point of $V(u)$ partitions free space in a number of quadrangular regions (each containing two points of $V(u)$). Associated to $V(u)$, one can construct a graph $\Gamma(u)$ whose vertices are the points of $V(u)$ and an edge is drawn between two vertices if they belong to the same quadrangular region of the partitioning. Figure 7 displays an example.

The idea behind the algorithm of Pocchiola and Vegter is to rotate the direction $u$ over an angle of $\pi$ (here: from 0 to $\pi$) while maintaining the graph $\Gamma(u)$. It is fairly clear that, as we have defined it, the graph $\Gamma(u)$ is essentially identical for all directions $u$ between two consecutive critical directions. We need to know then how this graph evolves in the neighborhood of a critical direction. The cases of bitangents and inflectional tangents are tackled by Pocchiola and Vegter. The evolution of $\Gamma(u)$ near these critical directions is displayed in Figure 8. From now on, an inner (resp. outer) bitangent will be a bitangent such that the objects touched lie locally on opposite sides (resp. on the same side) of the bitangent. A split bitangent will be
Figure 7: Partition of free space in quadrangular regions for a general curved object.

a bitangent that is neither of the inner nor of the outer type.

Figure 8: Updating the graph $\Gamma(u)$. a. Sweeping an inflectional tangent. b. Sweeping an inner bitangent. c. Sweeping an outer bitangent.

There is no conceptual difficulty in handling the other types of visual events*. The way the graph $\Gamma(u)$ evolves when sweeping a bitangent of the types induced by the objects we are considering (vertex - vertex and vertex - point of tangency) is similar to the classical bitangent case. The two really new visual events happen when sweeping a flat point or a curved vertex, and are displayed in Figure 9.

3.3.3. The algorithm

Let us define a mapping on $\Gamma(u)$. Let $e$ be some edge of $\Gamma(u)$ with endpoints $v_1$ and $v_2$. If $v_i$ ($i = 1, 2$) is not a vertex of $S$ (i.e., it is a point of tangency), it belongs

---

*This is the term we shall use from now on for such phenomena. It was coined several years ago in the field of aspect graphs - see Section 5.3.
Figure 9: New visual events. a. Sweeping the tangent to a curved vertex. b. Sweeping the tangent to a flat point.

to some arc $a_i$ of the partitioning of the boundaries of $S$ induced by the points of $V(u + \pi/2)$.

Now we let $\text{death}(c)$ be:

- the segment joining $v_1$ and $v_2$ if $v_1$ and $v_2$ are vertices;
- the segment with minimal $u$-slope (slope with respect to $u$) going through $v_1$ (resp. $v_2$) and tangent to $a_2$ (resp. $a_1$) if $v_1$ is a vertex and $v_2$ is not (resp. $v_2$ is a vertex and $v_1$ is not);
- the bitangent with minimal $u$-slope between $a_1$ and $a_2$ if neither $v_1$ nor $v_2$ are vertices.

Let $E(u)$ be the union of the images of the edges of $\Gamma(u)$ by $\text{death}$ and of the critical directions at inflection points, curved vertices and flat points having a positive $u$-slope. It should be fairly clear that $E(u)$ is unchanged when $u$ moves on the interval $[u_1, u_2]$, where $u_1$ and $u_2$ are two consecutive critical directions.

The algorithm of Pocchiola and Vegter is as follows:

1. Pick up some initial non-critical direction (here, the horizontal direction) and initialize $\Gamma := \Gamma(0)$ and $E := E(0)$.

2. While $E$ is not empty, do the following:
   
   (a) Compute the edge of $\Gamma$ where the next update occurs.
   (b) Update $\Gamma$.
   (c) Update $E$.

Step 1 is achieved through a standard sweep line algorithm. With the definition we have given for $n$, it is clear that the number of quadrangular regions of the partitioning (and thus the number of vertices of $\Gamma$) is bounded above by a linear function of $n$. This step thus takes $O(n \log n)$. Step 2.b can be done in constant time. Step 2.c requires including at the proper place an integer in a sorted list, so can be done in $O(\log n)$.

Step 2.a needs some special attention. When the edge to locate corresponds to a classical bitangent, then this step may be done in $O(1)$ since the mapping $\text{death}$ can be represented by bidirectional pointers. For other types of bitangents, there is no
additional difficulty (provided each vertex of the graph “knows” of what type it is). A difficulty occurs for inflectional tangents since an inflection does not correspond to a vertex of the graph. Locating the edge where the update occurs can however be done in $O(\log n)$ using balanced binary search trees. A similar argument shows that the case of flat points can be tackled in $O(\log n)$ as well. Finally, curved vertices are no big deal since they appear as nodes of the graph. On the whole, the algorithm has complexity $O((k + n) \log n)$, where $k$ is the number of bitangents of the scene.

3.4. Reporting Vri and IVri Active Bitangents

We need to prune those bitangents that are Vri or IVri inactive. As we have seen, a bitangent is Vri active iff its line of support is free. It is IVri active iff it is inner and one of the two half-lines containing the bitangent and originating at one of the endpoints of the bitangent is free. We thus need to report the intersections of the line of support of the bitangent with the scene. Rather than doing that as a post-processing on the results of the above algorithm, it is a simple matter to modify the rotational sweep so as to report these intersections.

So with every node of the graph $\Gamma(u)$ we associate two fields $\mathbf{v}_{\text{vis}}$ and $\mathbf{b}_{\text{vis}}$ giving the forward (resp. backward) visibility of the half-line with direction $u$ (resp. $u + \pi$) originating at the associated point of $\partial S$. Each field is set to 0 if the half-line is free, to 1 otherwise.

3.4.1. Initializing the forward and backward visibility

Before doing the rotational sweep, we need to initialize the $\mathbf{v}_{\text{vis}}$ and $\mathbf{b}_{\text{vis}}$ fields. To do this we proceed by traversing the oriented graph $\Gamma(0)^\circ$ from the bottommost to the topmost node, recalling that each node has at most a left (backward) child and a right (forward) child. Consider the situation depicted in Figure 10. Starting with node $A$, we set the visibility fields to 0. On the figure, the two fields are displayed below each node.

We then go on by propagating the fields among the nodes of increasing altitude. When a node of degree $1^d$ is met (concave point), we set both fields to 1. For a node corresponding to an extremal point (ordinary tangency, degree 3), the inheritance is as displayed in Figure 11, where $a$ and $b$ stand for the visibility fields. $A +$ sign above a node means that the corresponding point of the boundary is on the top of its object, otherwise it is marked with a $-$. Figure 11.a shows the case when the node has two children, while Figure 11.b displays the case of a single child. On the latter figure, the second row presents particular cases where both the node and one of its parents are on the same object (labeled z). The case of degree 3 vertices is similar. For a vertex giving birth to a node of degree 2, the node inherits the
visibility fields from its father.

Figure 10: Initializing visibility fields.

Figure 11: Propagation of the visibility fields for an extremal point or a degree 3 vertex. a. Two children. b. Only one child.

Looking back at Figure 10, node B matches Fig. 11.a, second diagram. Its $bvis$ field is set to 1 and its $fvis$ field is equal to that of $A$. Moving to node $C$, we are in the situation of Fig. 11.b second row, second diagram (both $A$ and $C$ are on the
same object. Its $bvis$ field is equal to that of $A$, its $fvis$ field is set to 1. Now node $D$: it matches the third diagram of the second row of Fig. 11.b - both $B$ and $D$ are on the same object - so its $bvis$ field is that of $C$ and its $fvis$ field is that of $B$. And so on.

3.4.2. Updating $\Gamma(u)$

Now that the visibility fields of $\Gamma$ have been initialized, the graph $\Gamma(u)$ needs to be updated each time a visual event is crossed. For this, we simply rework a bit the visual event diagrams, indicating how the visibility evolves. For instance, when sweeping an ordinary bitangent, the evolution is as in Figure 12.c if the bitangent is inner, as in Fig. 12.b for an outer bitangent and as in Fig. 12.a for a curved vertex. We then keep on a separate list the visibility fields of each bitangent at the moment of its sweep.

![Figure 12: Updating visibility fields. a. Sweeping a curved vertex. b. Sweeping an outer bitangent. c. Sweeping an inner bitangent.](image)

The conclusion of the rotational sweep is that a bitangent is $\forall\forall$ active iff both of its visibility fields are set to 0. A bitangent is $\forall\forall$ active iff it is inner and at least one of its visibility fields is set to 0. Note that an inner bitangent can bound $\forall\forall$ if only one of its visibility fields is set to 0.

4. Constructing External and Internal Visual Hulls

We can now wrap things up on the construction of the internal and external visual hulls of the collection $S$ of planar figures.

4.1. The Algorithm

The different steps of the construction are as follows:
Step 1. Compute the tangent visibility graph of $S$ and prune the inactive bitangents. Complexity: $O((k + n) \log n)$, where $k$ is the size of the visibility graph and $n$ the complexity of the scene.

Step 2. Merge the arcs of $\partial S$ that are endpoints of inactive bitangents. If we want the internal visual hull, eliminate outer active bitangents. Complexity: $O(k)$.

Let $k'$ be the size of the (VH or IVH) active tangent visibility graph. Now that the visibility graph is constructed, we need to build the partition of the plane induced by active segments. First we throw in the arcs of $\partial S$ and there are $O(k')$ of them. If we are interested in the internal visual hull, then for each active bitangent of the graph, we throw in a segment and a half-line according to Figure 5. If we want the external visual hull, then throw in either two half-lines or one segment as in Figure 4. Globally, we have $O(k')$ active segments and $m = O(k'^2)$ vertices of the partition.

Step 3. Construct the partition of the plane induced by active segments, by a standard sweep line algorithm, and traverse it. Complexity: $O(m \log m)$.

For each region of the plane thus constructed, we need to compute its visual number. To achieve this, we start with a region of zero visual number. As we have explained, if we want to construct the internal visual hull, we do not want to cross the boundary of $S$ since the internal visual number can assume any value immediately outside an object.

Step 4. Traverse the partition and compute the visual numbers. Complexity: $O(m)$.

Step 5. Merge the regions of visual number 0. Complexity: $O(m)$.

We end up with an algorithm that has complexity $O((k + n) \log n + m \log m)$, knowing that $m \leq k'^2 \leq k^2$ and that $k$ may be $O(n^2)$ in the worst case. If the scene is made of convex objects, then using the visibility graph algorithm of Pocchiola and Vegter\textsuperscript{21} reduces the complexity to $O(n \log n + k + m \log m)$. Note that the best algorithm known up-to-now (described by Laurentini\textsuperscript{11}) has complexity $O(n^3 + m \log m)$ and applies only to general polygonal objects.

In the worst case, our algorithm does not perform better than that of Laurentini, since in this case $m = O(n^3)$ and the $m \log m$ term dominates the time complexity of both algorithms. It seems natural to think however that in practical situations $m$ is much smaller than in the worst case, but more work has to be done to assert the validity of this idea. We may benefit in this context from work done on arrangements of line segments that share endpoints,\textsuperscript{4} since this is what happens for active segments.

We are in the process of fully implementing the above algorithm. Our current implementation is limited to the case of convex obstacles and uses the pseudo-triangulation approach of Pocchiola and Vegter.\textsuperscript{21}
4.2. Additional Comments

In the plane, the external visual hull of $S$ is the same as the external visual hull of the convex hulls of the $S_i$. Thus, in scenes containing a large number of concave objects, an alternative to the above algorithm would be to first compute the convex hulls of the objects (which may be done in $O(n)$ for polygons for instance) and then construct the visibility graph of this simplified scene. This may potentially decrease the value of $k$ by an important factor and thus lower the computation time. Note however that in higher dimensions the external visual hull of a connected object is not equal to the convex hull of that object, so the argument holds only in 2D. But if some efficient algorithm was known for computing the visibility graph of a single object in 3D, then we could apply a similar method.

As for internal visual hulls, a similar strategy may be applied. Constructing the visibility graph of a single polygon can be done in $O(k)$ [8]. Among those $k$ bitangents, we are only interested in the $k'$ inner ones. First, note that the supporting line of an inner bitangent necessarily cuts the boundary of the object. What we ought to know is whether one of the two half lines containing the bitangent and starting at one of the endpoints of the bitangent falls in free space (in which case it is active) or not - see Figure 13. Testing this may be done in an obvious way in global time $O(k'n)$. For each of the remaining active bitangents, the point of intersection of the supporting half-line with the boundary of $S$ is inserted, an operation that globally takes $O(k'n \log n)$. Marching on this boundary and alternating between edges of $S$ and visually active segments then allows to finish the construction of the internal visual hull.

![Figure 13: Constructing the internal visual hull of a single polygon. Example of an active and an inactive inner bitangent.](image)

Up-to-now, we have used the same visibility graph algorithm for computing both the internal and external visual hulls, without taking into account that not all bitangents of the scene are to be used to build the hulls. Split bitangents may occur in scenes containing angular points. But while the visibility graph algorithms compute all kinds of bitangents, only the inner ones are needed for the internal visual hull and only the inner and outer ones for the external visual hull. It is thus meaningful to look for visibility graph algorithms that compute only the kinds of bitangents that we need.

First, we may already note that the algorithms of Pocchiola and Vegter [21] on
convex sets - the rough idea of which is to first pseudo-triangulate free space and then to flip the diagonals of the pseudo-triangulations to discover all bitangents - may be easily modified if only inner bitangents are needed. Indeed, it turns out that the flip operation transforms an inner bitangent into an inner bitangent and similarly for outer ones. After the initial pseudo-triangulation is computed, only the inner bitangents need to be flipped and thus the complexity may be lowered to $O(n \log n + k')$ for the optimal algorithm and to $O((n + k') \log n)$ for the other algorithm, $k'$ being the number of bitangents of the type(s) wanted.

As for the the rotational sweep algorithm we have used in this paper, it is not too difficult to modify it so as not to compute split bitangents, but it appears to be harder to distinguish between inner and outer bitangents. This will be the subject of future research. It would also be interesting to examine in detail the relations between the number of bitangents of each type.

4.3. Answering Queries

We may also be interested in answering queries with respect to visual hulls. A given scene may be preprocessed (visibility graph computation and visually active bitangents determination) in time $O(k \log n)$. Then the following queries (among others) may be answered in time $O(k')$:

- **What is the visual number of a point?** For each of the $k'$ visually active bitangents, increase the visual number by 1 according to Figures 4 and 5.

- **Is a point inside the visual hull?** Answer is yes if its visual number is 0.

- **Is a line segment inside the visual hull?** Answer is yes if both endpoints have visual number 0 and if the segment does not cross any visually active segment.

- **Is a point on the boundary of the visual hull?** Answer is yes if it lies on some visually active segment and if it has visual number 0 (assumption is made that the visual number is increased only when the point is in the correct open half-plane).

- **Is a point a vertex of the visual hull?** Yes if it has visual number 0 and if it lies on two visually active segments.

Finally, we may wonder, given two unknown scenes, whether their visual hulls are equal or not. This is the same as telling if they can be distinguished from silhouettes alone. In 2D, we may do that by circling around the scenes, computing a volume each time a visual event is encountered and intersecting all volumes. This is a $O(n^2)$ operation. If the intersection is the same then the visual hulls are equal. In 3D, this operation is far less obvious (see Section 3.2 for more). If the objects are assumed to be transparent, then we may get help from differential geometry. Pointet\(^{23}\) gives a minimal criterion for a set of viewpoints $\mathcal{W} \subset S^2$ to separate two compact surfaces embedded in $\mathbb{R}^3$, i.e., such that one can tell from the contours observed from viewpoints of $\mathcal{W}$ whether the two surfaces are equal or not. Namely, $\mathcal{W}$ should be $3$-omnidirectional. It is not necessary to give the formal definition of
this notion: an example of such a set \( W \) is the union of 4 great circles such that each point of the union belongs to at most 2 great circles. Since there are \( O(n^3) \) visual event surfaces (see Section 6), \( O(n^3) \) visual events are observed on each great circle so the global time complexity is \( O(n^3) \). In the opaque case, not much is known. There are however strategies for reconstructing the visual hull of an object known to be equal to its visual hull, as we shall see.

5. Possible Extensions and Other Issues

In this section, we examine some issues raised by the definition of visual hulls. First, we examine the case of a curved object in the plane, for which an infinite number of viewpoints are needed for exact reconstruction. If some reconstruction error is allowed, we compute the number of viewpoints needed to achieve such an error. Second, we discuss how to efficiently reconstruct a scene knowing that it is equal to its visual hull. Third, visibility graph and visual hull algorithms apparently need to know the types of visual events that can happen as a function of the geometry of the scene. These events have been classified in the field of aspect graphs and we present some results established there.

5.1. Reconstructing Curved Objects From a Finite Number of Views

Consider a curved object in the plane. It is clear that exactly reconstructing by volume intersection that object will take an infinite number of viewpoints. In a real application, we may very well be happy with a reconstruction of that object that is accurate up to some precision \( \varepsilon \). In this section, we want to relate this precision to the number of viewpoints needed to achieve it and to the distance between the object and the viewpoints.

Note that schemes for globally reconstructing objects have recently been developed. For instance, Kutulakos and Dyer have presented a technique for purposefully adjusting viewpoint by aligning the viewing direction with a principal direction at a curved surface point.

5.1.1. Global reconstruction

So let the object \( S \) be a unit circle centered at the origin and \( R \) the object reconstructed from \( n \) viewpoints situated on some geometric figure at distance \( r \) of the origin (to be defined later). The global reconstruction error is defined as:

\[
\varepsilon = \frac{A(R) - A(VH(S))}{A(VH(S))},
\]

where \( A() \) means “area of”. Here, the object is convex and its area is \( \pi \). Now, to get a grab at the relation between \( \varepsilon \) on one side and \( n \) and \( r \) on the other side, we need to compute the area of the polygon reconstructed. Let us denote by \( f(r,n) \) the area of \( R \). To achieve this, we have considered several situations for the positions of the viewpoints. Our first experiment was to place them on a circle of radius \( r \) centered at the origin. The situation is as depicted in Figure 14. For small values
of $n$, we were able to find the formulas for $f(r, n)$ with the help of the Maple\textsuperscript{14}
symbolic platform:

\[
\begin{align*}
    f(r, 3) &= \frac{6r^2}{3r^2-\sqrt{3}}, \\
    f(r, 4) &= \frac{4r^2}{3r^2} \left( 1 - \frac{1}{\sqrt{3}} \right), \\
    f(r, 6) &= \frac{6r^2}{3r^2-4} \left( \sqrt{3} - \frac{1}{\sqrt{3}} \right).
\end{align*}
\]

Unfortunately, it is hopeless to look for a general formula for $f(r, n)$ because the
coordinates of the viewpoints (sines and cosines of nontrivial angles) generate
complicated exact values. Anyway, we have plotted on Figure 15.a the error made on
the reconstruction for $n = 6$ as a function of $r$. It turns out that the error is large
for small values of $r$, is minimal (0.02\% ) for $r = 2\sqrt{2} + \sqrt{3} \approx 3.86$ and is asymptotically
equal to $(2\sqrt{3} - \pi)/\pi \approx 10.3\%$. $f(r, 4)$ has the same general behavior and we
may conjecture that the reconstruction error always admits a minimum at a value
depending on $n$.

Our second experiment consisted in placing the viewpoints on a vertical line at
distance $r_1$ of the origin, the $n$ viewpoints being equally spaced between heights
$y = -r_2$ and $y = r_2$ (see Figure 16.a). We could obtain values for the area of the
reconstructed polygon for low values of $n$. For instance,

\[
f(r_1, r_2, 2) = \frac{2 \frac{r_2(r_1^2 + r_2^2 - 1)}{r_1^2 - 1}}.
\]

For fixed values of $r_2$ and $n$, there is a minimum of the function $f(r_1, r_2, n)$ that
happens for some value of $r_1$ depending on $r_2$ and $n$, as before. For fixed values of
$r_1$ and $n$, a similar behavior happens.

Our final example is the most interesting one, because we have been able to
handle it completely. Suppose that the viewpoints are placed on a square at distance

\[
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{circle_views.png}
\caption{Reconstructing a unit circle from a few viewpoints on another circle of
radius 4. a. 4 viewpoints and 8\% of reconstruction error. b. 11 viewpoints and
less than 1\% of reconstruction error.}
\end{figure}
\]
Figure 15: Measuring reconstruction errors. a. Area as a function of the radius $r$ of a circle on which 6 viewpoints are equally spaced. b. Error in % as a function of $n$, where $4n$ is the number of viewpoints placed on a square at infinite distance.

Figure 16: Reconstruction from points on a square of “radius” 4 or on a line at distance 3. a. 4 viewpoints equally spaced on a line (height between -4 and 4) with a reconstruction error of 10.4 %. b. 8 viewpoints on a square (i.e., $n = 2$) with 1.42 % of reconstruction error.

$r$ of the origin (i.e., its vertices are $(r, r), (r, -r), (-r, -r), (-r, r)$), such that there are $n$ viewpoints on each side of the square (total number is $\nu = 4n$) - see Figure 16b. We started by computing the area $f(r, n)$ of the reconstructed polygon for small values of $n$. For instance,

$$f(r, 3) = \frac{4}{r^2 - 1} \left( \frac{\sqrt{(2r^2 - 1)(10r^2 - 9)}}{3\sqrt{10r^2 - 9}} - 2(11r^2 - 9) \frac{3r^2 - 2}{\sqrt{2r^2 - 1}} - \frac{11r^2 - 9}{3} \right).$$

This function has the same general behavior as previously, i.e., it has a minimum at some value of $r$ depending on $n$. More interesting is that we were able to give
the general case formula. Let us start by setting \( f_i = (n^2 + 4i^2)r^2 - n^2 \) and \( g_i = (n^2 + 8i^2)r^2 - n^2 \). Then for \( n \) even and for large values of \( r \):

\[
f(r, n) = \frac{4}{n(r^2 - 1)} \left( -\sqrt{f_i} + \sqrt{f_{i+1}} \right) + \sum_{i=1}^{n/2} \left( \sqrt{f_i} f_{i+1} - \frac{2g_i}{\sqrt{f_i}} \right) \frac{n^2((2n - 3)r^2 - n + 1)}{\sqrt{f_{n/2}}} - \frac{2n(n^2 - 1)r^2}{3} + \frac{n^3}{2} \). \tag{1}
\]

With the restriction on \( r \), we want to look at the case when \( r \) goes to infinity (which in the context of computer vision is very much like assuming that the projection model is the orthographic projection, a fair approximation to true perspective). When \( r \) tends to infinity, we have:

\[
f(\infty, n) = \frac{4}{n} \sum_{i=1}^{n/2} \sqrt{(n^2 + 4i^2)(n^2 + 4(i - 1)^2)} - \frac{2n(n^2 - 1)}{3})
\]

Let us first look at the summation. To find its series expansion when \( n \) tends to infinity, we write it as follows:

\[
\sqrt{(n^2 + 4i^2)(n^2 + 4(i - 1)^2)} = (n^2 + 4i^2)\sqrt{1 - \frac{4(2i - 1)}{n^2 + 4i^2}}
\]

Using the fact that around \( x = 0 \)

\[
\sqrt{1 - x} = 1 - \frac{x}{2} - \frac{x^2}{8} - \frac{x^3}{16} - \frac{5x^4}{128} + O(x^5),
\]

and since \( x = \frac{4(2i - 1)}{n^2 + 4i^2} = O(\frac{1}{n}) \) when \( n \to \infty \), we can compute the expansions using the Euler-MacLaurin summation library of Maple. We find that

\[
\sum_{i=1}^{n/2} \sqrt{(n^2 + 4i^2)(n^2 + 4(i - 1)^2)} = n \left( \frac{\pi}{4} - \frac{2}{3} \right) + \frac{2n^3}{3} + \frac{1}{4n} \left( \frac{1 + \pi}{3} \right) + O \left( \frac{1}{n^3} \right).
\]

Thus,

\[
f(\infty, n) = \pi + \frac{1}{n^2} \left( \frac{1}{3} + \frac{\pi}{8} \right) + O \left( \frac{1}{n^3} \right).
\]

In turn, this confirms that \( f(\infty, n) \) converges to \( \pi \) and gives an asymptotic behavior of the error:

\[
\varepsilon = \frac{1}{n^2} \left( \frac{1}{3\pi} + \frac{1}{8} \right) + O \left( \frac{1}{n^3} \right).
\]

We have plotted in Figure 15.b this reconstruction error for a square at infinite distance as a function of \( n \). Since in our case the number of viewpoints \( \nu \) is equal to \( 4n \), we see that the error goes asymptotically as \( O(\frac{1}{\nu^2}) \). Conversely, the number of viewpoints needed to have an error of \( \varepsilon \) is asymptotically equal to:

\[
\nu \approx \frac{1.92}{\sqrt{\varepsilon}}.
\]

20
5.1.2. Local reconstruction

The global definition of reconstruction error given above, though correct, may seem quite unpractical in real situations where only local information are available. For instance, Boyer and Berger show how to obtain a measure of the error on the depth (with respect to the camera) of the reconstructed point. This measure may be linked to the distance between the reconstructed point and the correct point on the object.

First we define the notion of $\varepsilon$-$\mathcal{V}(-r)$ to be the union of $\mathcal{V}$(-r) and of the set of points outside of $\mathcal{V}$(-r) at a maximum distance $\varepsilon$ of $\partial\mathcal{V}$. Now, given a reconstructed object $\mathcal{R}$, we may define a local reconstruction error $\varepsilon'$ as the smallest real number such that $\mathcal{R}$ lies entirely inside $\varepsilon'$-$\mathcal{V}(-r)$.

Getting back to the case of a unit circle examined in the previous section, computing $\varepsilon'$ amounts to computing the maximum distance of a vertex of the reconstructed polygon from the circle. If the viewpoints are placed uniformly on a square at distance $r$, then keeping the notations that we have already introduced, we find that:

$$\varepsilon' = \sqrt{h(r, n)} - 1$$

where

$$h(r, n) = \frac{1}{2(r^2 - 1)} \left( 2\sqrt{f_0} - 2\sqrt{f_1} - \sqrt{f_0f_1} + f_1 \right).$$

When $r$ goes to infinity, we get that:

$$h(\infty, n) = \frac{1}{2}(n^2 + 4 - \sqrt{n^2(n^2 + 4)}).$$

This in turn means that when $n$ tends to infinity,

$$h(\infty, n) = 1 + \frac{1}{n^2} + O\left(\frac{1}{n^4}\right),$$

and finally that the reconstruction error is:

$$\varepsilon' = \frac{1}{2n^2} + O\left(\frac{1}{n^4}\right).$$

Some comments are in order. First, the orders of magnitude of the local and global reconstruction error are the same. This gives an indication that we may not be misled by only enforcing that local reconstruction errors are kept to a minimum. Second, we have assumed here that the camera is allowed to make a full rotation around the object but this is usually not the case in practice. A finer analysis is needed to assert the validity of the above result for instance in the case when the camera is kept within some polygonal region. Another issue raised by the above study is where to place the viewpoints and how many are needed to minimize reconstruction error.
5.2. Computing the Visual Hull from Silhouettes

In this section, let us assume that the scene $S$ is polyhedral and such that it is equal to its external visual hull, i.e. that it is \textit{VH}-equal ($\epsilon$-reco in the terminology of Laurentini\cite{laurentini}). Let also $n$ be the complexity of this scene. We are interested in reconstructing such a scene from silhouettes alone and discussing what strategy may be adopted to place the viewpoints in an efficient way.

Globally, two different strategies may be used. In the first one, we do not take into account the geometry of the scene and regularly sample a surface enclosing $S$. In the second, we look for “unstable” viewpoints at which visual events are observed. Indeed, at such viewpoints the camera center $C$ is in the supporting plane of a face of $S$ and thus potentially (but potentially only - see Fig. 17.a; this phenomenon may be avoided by considering topological changes on the silhouette of the object only) a portion of this face will be on the boundary of the volume computed from $C$ and thus on the boundary of the object reconstructed. Better yet, we can look for viewpoints belonging to two different supporting planes of faces of $S$, since in this case two faces (or two portions of faces) will be reconstructed at the same time.

It is known however that unstable viewpoints can never be reached (since they belong to a set of measure 0 in the space of camera positions) and that image computations are highly unreliable at these viewpoints. Some compromise has to be found between being close enough to such viewpoints so as to minimize the angle between the viewline and the supporting plane of the face under consideration and far enough to avoid unreliable data.

5.2.1. Approximate Reconstruction

Even if the object under consideration may be exactly reconstructed in a finite number of steps, the large number of viewpoints needed may be prohibitive in practical situations. We may be satisfied with a reconstruction that is accurate up to some $\epsilon$ but that needs a much smaller number of viewpoints to be obtained.

In 3D, the positions of the viewpoints may be naively obtained by regularly sampling the sphere $S^2$. We have not carried out the calculations as we have done in the 2D case for the observation of a curved object, but we may conjecture that $O(n^2)$ viewpoints on the sphere would yield a reconstruction error that behaves asymptotically as $\frac{1}{n}$.

A different technique may be to use viewpoints placed at the topological changes observed on a set of great circles satisfying omnidirectionality.\cite{laurentini} The number of such viewpoints is $O(k)$, where $k = O(n^3)$ is the number of visual event surfaces.

Finally, here is a third approach. Consider a set of $m$ great circles on the 2-sphere, picked either at random or regularly. Now walk on each great circle. When you meet a topological change of the contour, find the (unoriented) direction in which the face of $S$ inducing the visual event intersects the great circle. Walk in both (oriented) directions until another visual event happens and take these as viewpoints. This process is illustrated on Figure 17.b. Start with point $A$ on some great circle, moving upwards. At some place, we cross the plane supporting face $F$.
moving from $B$ in the directions induced by this plane we stop when a new visual event occurs (points $C$ and $D$) and we take these points as viewpoints. Doing this for all great circles, we end up with $O(mk)$ viewpoints. Of course, not all faces of the scene may have been uncovered this way.

![Diagram](image)

Figure 17: Approximate reconstruction of a Vit-equal polyhedron. a. Face $F$ is reconstructed from viewpoint $C_2$ but not from $C_1$, where $C_1$ and $C_2$ are in the supporting plane of $F$. b. A viewpoint placement strategy.

5.2.2. Exact reconstruction

In 2D, the visibility graph has worst case complexity $k = O(n^2)$. Thus the visual hull of $\mathcal{S}$ may have at most $O(n^2)$ edges. But if the object is Vit-equal, then its visual hull has only $O(n)$ edges. It suffices then to patrol on a circle enclosing the object. The lines supporting the $O(n)$ edges generate $O(n)$ points of intersection on the circle (the viewpoints). These viewpoints may be detected by observing changes in the topology of the contour.

In 3D, Lauringanti\textsuperscript{13} describes an algorithm for reconstructing a Vit-equal polyhedron with $n$ faces in $O(n^5)$ volumetric intersections. The algorithm works roughly as follows. Given a point $P$ of face $F$ and a free line $L$ through $P$, let $\mathcal{P}$ be the plane supporting $F$ and $\mathcal{L}$ the polygonal intersection of the polyhedron $\mathcal{S}$ and $\mathcal{P}$, excluding $F$ and any coplanar face. If $\mathcal{L}$ is empty, then $F$ belongs to the convex hull of $\mathcal{S}$ so a single viewpoint is needed to reconstruct it. Otherwise, rotate line $L$ several times in the plane $\mathcal{P}$ until it goes through two vertices of $\mathcal{L}$. The intersections of these lines bitangent to $\mathcal{L}$ provide viewpoints from which face $F$ can be reconstructed. Since the complexity of $\mathcal{L}$ is $O(n)$, there are $O(n^2)$ bitangent lines to $\mathcal{L}$ and $O(n^4)$ points of intersections of these lines. Doing this procedure for each face, we get the $O(n^5)$ bound for the number of viewpoints.

This algorithm suffers from several drawbacks. First, assumption is made that one knows a point inside each face of the polyhedron and one line through this point that does not intersect the object transversally. No mention is made however as how these points and lines could be obtained. Second, it is not clear how the
algorithm can deal with the situation presented in Fig. 18. With the point $P$ and the line $L$ given, only one part of $F$ will be discovered by continuous deformations (rotations) of the line. To discover the other branch of the cross, one needs to know either another point of the face and a free line through it, or how to jump to the new position. Third, we do not see why viewpoints have to be placed at the intersections of bitangents to $L$: any viewpoint on the line supporting the bitangent will be able to reconstruct the corresponding part of $F$. It thus suffices to take one viewpoint for each of the $O(n^2)$ bitangent lines: since there are $O(n)$ faces, only $O(n^3)$ viewpoints are needed.

![Figure 18. Two families of lines are needed to reconstruct face $F$. $F$ is represented in dashed lines, while $L$ is in plain.](image)

5.3. Relation to Aspect Graphs

The *aspect graph*\(^9\) is a popular viewer-centered representation that enumerates all the topologically distinct line drawings ("aspects") of an object. The viewpoint space considered (usually, under perspective projection, it is the entire space in which the object lives) is partitioned in view-equivalent cells by *visual event hypersurfaces* and one can associate a *canonical* (representative) view to each cell. A graph is constructed whose nodes are the canonical views and an arc links two nodes if the corresponding cells share a common boundary.

The theory of aspect graphs has reached a highly sophisticated level, dwelling deep into singularity theory\(^25\) and into algebraic geometry.\(^17\) Why so? Because before any construction scheme can take place, a classification of the possible visual events has to be made. Most importantly, one needs to know the possible visual event surfaces, the main partitioners of viewpoint space. But they highly depend on the types of objects in the scene.

Interestingly, it seems that algorithms to construct visibility graphs (or the more elaborate visibility complexes\(^22\)) have to have some knowledge of the visual event hypersurfaces developed in works on aspect graphs. For instance, Durand et al.\(^5\) use the $EEE$ (lines meeting three edges) and $EV$ (lines meeting an edge and a
vertex) events for their construction of the visibility complex of polyhedral scenes.

It is thus legitimate to ask for what types of objects is the largest classification of visual events known. Rieger\textsuperscript{26} gives a classification for objects in general position that are piecewise-smooth $C^\infty$, possessing at most a transversal curve $D$ of self-intersection and a finite number of triple points on $D$. A very recent preprint of Rieger\textsuperscript{24} gives a slight extension of this classification to account for piecewise-smooth $C^\infty$ objects possessing in addition a finite number of cross-caps (also known as pinchpoints), points of $D$ at which the two tangent planes of the surface coalesce. It is not the place here to describe this classification, but let us simply say that it consists of 19 different types of surfaces, each corresponding to some geometric situation involving contacts of lines and planes with the objects. For instance, the most complicated surfaces are the trilocal ones (contacts with the objects at three different places): one type is made of tritangent lines, another of lines bitangent and further cutting $D$, . . . . If the total degree of the surfaces defining the objects is $d$, then the most complicated visual event surfaces have degree $O(d^3)$, which means that the worst case complexity of the visibility graph is $k = O(d^3)$.

Now back to visual hulls. The classification of Rieger is the most complete known up-to-now. This implies that no visibility graph or visual hull algorithm can be given for more complicated objects (like for instance for CAD objects with two patches having some high order of mutual tangency). But Rieger goes further\textsuperscript{18}, he proves that for more complicated surfaces the very notion of aspect graph does not make sense. For such objects, the visual event surfaces may be region-filling, making it impossible to define a discrete graph structure in the space of camera positions like the visibility graph. Of course, for such classes of objects one could probably find open dense subsets of surfaces for which a notion of graph could be defined, but this is an open problem.

\section*{6. From 2D to 3D}

Even though this is not the main purpose of this paper, let us move up to the 3-dimensional case which is the main motivation behind the study of visual hulls. Roughly speaking, the techniques to be used in this case proceed from the same ideas. We only present here the main issues, the details being left for a future report.

Laurentini\textsuperscript{11} describes an algorithm for computing the visual hulls of polyhedral objects. Given a point $P$ and an object $S$ with a total number of vertices $n$, he defines the visual cone of $P$ as being the set of visual lines of $S$ through $P$. The \textit{3D visual number} of $P$ is the number of faces of its visual cone. The algorithm then goes on by partitioning viewpoint space in cells of constant visual number. The visually active surfaces are parts of the more general visual event surfaces known in the field of aspect graphs\textsuperscript{7}: those surfaces are ruled surfaces made of lines through three edges ($EEE$) and surfaces made of lines through a vertex and an edge ($EV$). Since in this case, there are $O(n^2)$ \textit{potentially active} surfaces, the partition has $O(n^2)$ cells, faces and vertices. The partition can then be constructed in $O(n^3 \log n)$. Computing the visual number of each cell can be done in $O(n^3)$, so
the algorithm has overall complexity $O(n^{12})$.

6.1. Visibility Complexes of Polyhedra and Convex Objects

If concerning visibility structures the 2D case is well-understood, it is by far different for the 3D case. The few works that we are aware of have been developed from the point of view of visibility complexes, which we introduce now.

Consider a scene $S$ of $\mathbb{R}^3$. We want to partition the set of maximal free line segments - segments for short - according to the objects they see (i.e. that their extremities see). The visibility complex of $S$ is a partition of the set of segments according to their visibility. It was introduced by Pocchiola and Vegter, who construct the visibility complex of $n$ planar convex figures in time $O(n \log n + k)$, where $k$ is the size of the complex. For a 2D scene, the complex is a 2-dimensional object embedded in a 3-dimensional space: a face corresponds to segments seeing two given objects, an edge to segments seeing an object and tangent to another, and a vertex to segments tangent to two objects. For a 3D scene, the visibility complex is a 4-dimensional object embedded in a 5D space: a face (4D) corresponds to segments seeing two objects, a tangency face (3D) to lines tangent to some object, a bitangency face (2D) to lines tangent to two objects, a tritangency edge (1D) to lines tangent to three objects and a vertex (0D) to lines four times tangent to $S$.

If $S$ is a polyhedral scene with a total of $n$ edges, there are $O(n^3)$ visual event surfaces of the type $EEE$, so the visibility complex has worst case complexity $O(n^4)$. Durand et al. present a theoretical algorithm for constructing the visibility complex in this case in time $O(n^4 \log n)$. On the other hand, Pocchiola (personal communication) proposes an algorithm for $n$ convex objects. The idea is to compute for each convex the sub-complex of rays emanating from this convex as the lower envelope of the family of functions that associate to each ray the object visible along this ray. Such a computation can be done in time $O(n^4 \log^* n)$, so repeating it for each convex gives a $O(n^5 \log^* n)$ algorithm. In addition, this approach may prove interesting since the lower envelope algorithm may be output-sensitive.

6.2. Visual Hulls of Polyhedral Objects

The visually active surfaces can be easily computed from the visibility complex: they are to be found among the 1-faces (tritangency edges) of the complex. Now back to the polyhedral case. Note that the first step of the algorithm of Laurentini is to compute the potentially active surfaces in time $O(n^3)$. Among the 1-faces computed, some have to be pruned because they further meet the scene $S$ transversally. To achieve this, we again use the information gathered in the complex: as mentioned by Durand et al., the complex can be stored using a polytope structure, where each $k$-face has pointers to its boundaries, to the larger faces to which it is adjacent and to the objects it sees. Pruning of inactive surfaces can thus be done by traversing the complex.

For each of the remaining visual event surfaces (let us say there are $m$ of them, $m = O(n^3)$ in the worst case), we associate visually active patches as in the 2D
case. For instance, for a $VE$ surface, the portion of the surface to keep is the one extending between vertex $V$ and edge $E$. The next step is to actually construct the partition of space induced by these portions of visually active surfaces. This partition has $O(m^3)$ complexity and may be constructed in time $O(m^3 \log n)$ (see Plantinga and Dyer\textsuperscript{19}). Also as in 2D, there are visually active surfaces that do not correspond to boundaries of the visual hull and can be pruned beforehand.

Laurentini goes on by computing the visual numbers of each region of the above partitioning. But with the definition he chose for the 3D visual number, a lot of computations has to be done: crossing a visually active surface does not mean adding or subtracting one to the visual number. Each visual number has to be computed separately, and since each 3D visual number takes $O(n^3)$ to compute, the overall complexity of this step is $O(n^{12})$. The trouble with this definition is that it depends too much on the geometry of the scene: a point immediately outside a face of a pyramid does not have the same visual number as a point outside a face of a cube. We could say that the definition of 2D visual number is “topological” and this one is “geometrical”.

But we may very well take as definition for the 3D visual number of a point $P$ the number of different families of free lines going through $P$ (two lines being in the same family if they can be continuously moved one onto the other). It then suffices to traverse the partition, starting from some initial position (only those cells inside the convex hull of $\mathcal{S}$ need to be traversed). A good initial position is a point immediately outside a face of $\mathcal{S}$ belonging to the convex hull of the scene: its visual number is necessarily 1. If we assume that the scene is in general position (such that for instance no line can meet five edges), then traversing a visually active surface means incrementing or decrementing the visual number by 1 (this increment may be attached to the surface). If the surface is not in general position, then a preliminary work should be to check for partial (or global) overlapping between visually active surfaces - this should not be too hard since they are at most ruled quadric surfaces for which we know the generators. If two surfaces overlap, then its increment should be set to two.

The traversal of the partition, and the marking of regions with visual number 0, can thus be done in time proportional to its size, i.e. $O(m^3)$. The overall algorithm for computing the external visual hull of polyhedral objects is thus $O((n^4 + m^3) \log n)$, where $m$ is $O(n^3)$ in the worst case. This should be compared to the $O(n^{12})$ of the algorithm of Laurentini.

7. Conclusion

In this paper, we have dealt with a structure known in the field of computer vision and object reconstruction as the visual hull and due to A. Laurentini. Informally, the external (resp. internal) visual hull of an object is the largest possible figure that one may reconstruct by volume intersection, where the viewpoints are constrained to lie outside of the convex hull of the object (resp. in free space).

We have presented new, more efficient and more general algorithms for computing the visual hulls of general planar curved and polygonal objects, improving
on results of Laurentini. Our approach uses the recent algorithms published for constructing visibility graphs and complexes. Again looking at things from the visibility complex standpoint, we have shown how the 3D case may be handled. We have linked visibility issues in 3D to the highly mathematical notions of visual events developed in the field of aspect graphs and have shown how results of that field may both enhance and limit constructions of visibility structures. We have also presented results on the number of viewpoints needed to reconstruct objects equal to their visual hull and we have given an analysis of the number of viewpoints needed to reconstruct a curved object up to a given precision.

We hope to have proved that many questions concerning visual hulls can be approached from a computational geometry standpoint and using the results and techniques of that domain. Much is however left to be done. First, visual hull computation is highly related to visibility graph / complex computation, so progress in the former is very dependent on progress in the latter. Specifically, efficient output-sensitive algorithms for computing visibility complexes of various classes of objects would clearly help. Also, can one design efficient visibility graph algorithms that compute only inner bitangents? Can they be turned dynamic? Randomized?

Second, if visual hulls in 2D are pretty well understood, the 3D case is much more complicated. What exactly makes that a visually active surface does not participate to the visual hull? What is the value $k$ of the complexity of the visibility structures in real (practical) situations? What other properties can be proved about visual hulls that would help characterize their geometry? Can one give necessary and sufficient conditions (other than their definition) for an object to be $V_h$-equal? And $IV_h$-equal?

Third, more experiments have to be conducted to determine how to efficiently place viewpoints so as to minimize the reconstruction error. Also, we need to evaluate whether visual hulls can be useful in actual object recognition techniques, such as that of Wallack and Manocha\textsuperscript{27} using light beams.

Fourth, the complexity of the visual hull of a general polyhedral object having $n$ faces is known to be $O(n^4)$ in the worst case. One may wonder for which object is this complexity lower. Pellegrini\textsuperscript{16} proves that for a compact star-shaped polyhedron (of which a terrain is a particular case), the complexity of the set of lines missing the object is $O(n^3 \beta(n))$, where $\beta(n) = 2^{c \sqrt{\log n}}$ is a subpolynomial function. He proves also that a connected component of the set of missing lines (an isotopy class) has the same order of complexity. Is there a connection between objects the visual hull of which has complexity lower than in the worst case and $V_h$-equal objects?

Finally, the definition of visual hull suggests possible extensions and objects of study. For instance in the plane one can define a $k$-concave polygon (we were unable to find a correct term for such a polygon so let us throw one in): a polygon is said to be $k$-concave ($k \geq 1$) if from any point of $\partial S$ a chain of $k - 1$ line segments and a half-line may be drawn that does not further intersect $\partial S$ transversally. The "1-concave hull" is the internal visual hull. Note that this shares some resemblance with the notion of link distance.\textsuperscript{28} What then is the time complexity to check if an
object is k-concave? What is the complexity of computing the k-concave hull of an object? Of a single object? We plan to come back to these issues in future reports.

Acknowledgments

I am pleased to acknowledge the reviewers for their helpful comments. I am also grateful to Sylvain Cunzi for interesting discussions on some of the topics of this paper.

References

14. MAPLE V, 1990. Copyright (c) 1981-1990 by the University of Waterloo. All rights reserved. MAPLE is a registered trademark of Waterloo Maple Software.

29