The Visual Computer

Hierarchical approximation and localization

Remco C. Veltkamp

Utrecht University, Department of Computer Science, Padualaan 14, 3584 CH Utrecht, The Netherlands
e-mail: remco.veltkamp@cs.uu.nl

This paper introduces a representation scheme for the boundaries of polygonal and polyhedral objects without holes. The scheme represents a hierarchical approximation to support processing at multiple levels of detail, together with hierarchical bounding volume information to support efficient set and search operations. The approximation is boundary based and intrinsic to the surface. The bounding areas contain the boundary segments of the object, not its volume. These bounding areas consist of intersections of 2D circles and 3D spheres, which makes the representation storage efficient, and set and search operations computationally efficient.

Key words: Polygonal and polyhedral approximation – Hierarchical object representation – Bounding volumes

1 Introduction

Complex polygonal (2D) or polyhedral (3D) object boundaries consist of many segments which are line segments or faces. To manipulate objects with many boundary segments, we need facilities to speed up the processing and avoid unnecessary operations. Two facilities are often used for efficient manipulation of complex polygonal or polyhedral objects consisting of many faces: approximation and localization. Both can be performed hierarchically. The purpose of both techniques is to avoid unnecessary processing of detail.

Approximation. It is not always necessary to process an object in full detail. If the approximation object consists of many fewer faces than the original object, the processing is usually much faster. For example, the interactive manipulation of a 3D object requires real-time display. During animated rotation, an approximation object allows faster display without much loss of sense of reality. Another example is the perspective display of distant 3D objects that are mapped to only a few pixels. Yet another application is hierarchical feature classification used in object recognition.

Localization. Localization provides information about the position of the volume or the boundary of the object by means of a set of bounding volumes that, together, contain (the boundary of) the object. If the bounding volumes allow efficient testing, we can locate points and test for intersections efficiently.

Both approximation and localization can take place at multiple levels of detail. If the successive levels are such that a segment at one level is refined at the next level, there is a *hierarchy* of successively more detailed levels. A hierarchy is naturally stored in a tree structure, where the root contains the most coarse level of approximation and the children of a node represent the refinement of the parent at the next level of approximation. Algorithms operating on the hierarchy try to work at the root. If this is not possible, the algorithm proceeds to successively more detailed levels.

2 Related work

This section discusses other methods as far as they are relevant. I do not explain other approximation schemes in detail if they are not applicable to bounding volumes.

The *iterative end-point fit* method (Duda and Hart 1973) is an approximation algorithm for 2D polygons that starts by connecting two initial end-points. If all the distances from the points to the line segment are less than a chosen error bound, the process is finished. Otherwise, the point farthest from the line segment is connected to the two end-points, yielding two new line segments. The process is repeated for each new line segment. This method is also known, especially in the cartography community, as the *Douglas-Peucker algorithm* (Douglas and Peucker 1973). There are many approaches in the cartography and GIS literature for approximation (map generalization), but not for localization. A '3D iterative end-point fit' algorithm selects triangles that approximate a part of a polyhedral surface (Faugeras et al 1984).

The *strip tree* (Ballard 1981) is a localization scheme represented by a binary tree that stores all intermediate approximations resulting from the iterative end-point fit method, together with some bounding area information: a strip. A strip that covers $v_0,\ldots, v_n$ represents the smallest rectangle with sides parallel to the line through v_0 and v_n. The *prism tree* (Ponce and Faugeras 1987) is a generalization of the strip tree to three dimensions, suitable for the hierarchical representation of polyhedral objects. A truncated pyramid (called a prism) enclosing a part of the surface is stored in each node of the tree.

The *arc tree* (Günther 1988) is a balanced binary tree representing a hierarchy of approximations and localizations of arbitrary planar curves. The localization is achieved by bounding ellipses. Let a curve C with length l be parameterized by its arc length fraction $t \in [0, 1]$. Then the length of the curve from $C(0)$ to $C(t_0)$ is $t_0 l$. The kth approximation consists of 2^k line segments. Line segment i connects $C(i/2^k)$ and $C((i+1)/2^k)$ and is the approximation of the arc of C between $C(i/2^k)$ and $C((i+1)/2^k)$, which has length $l/2^k$. The arc tree stores the hierarchy of approximations in a binary tree.

Some approximation methods only work for terrain surfaces, in which a height attribute is associated with vertices of a 2D triangulation. The *Delaunay pyramid* (DeFloriani 1989) is a representation of a sequence of 2D Delaunay triangulations. Starting with an initial Delaunay triangulation, a number of data points are added to obtain a more accurate terrain surface approximation. Alternatively, starting with a Delaunay triangulation at full detail, vertices can be deleted at the next levels (Berg 1997). Both methods are only approximation methods, not localization methods. Since all triangulations are done in the plane, such schemes are unsuitable for approximating boundary surfaces of 3D objects.

A number of other approximation schemes work only for surfaces in three dimensions. Schroeder's (1992) method iteratively removes vertices from the surface according to some selection criterion, followed by retriangulation. Hoppe et al. (1993) iteratively remove edges, but this may introduce new vertices into the surface. Hoppe (1996) successively shrinks edges until they collapse. Turk's (1992) scheme first generates new vertices into the surface and then discards the old vertices to create a new mesh. View-dependent simplifications are presented by Hoppe (1997) and Luebke and Erikson (1997). None of these schemes is hierarchical in the sense that each facet at one level is refined at the next, and they do not provide hierarchical bounding volume information.

Some optimal approximation algorithms by Imai and Iri (1988), minimize either the error, given a number of vertices, or vice versa. They are not usually hierarchical. There seem to be no efficient algorithms for optimal approximation in three dimensions, so that we are dependent on heuristics.

Finally, some approximation methods work only for vertices on a regular triangular (Schmitt and Gholizadeh 1986) or a rectangular (DeHaemer Jr. and Zyda 1991) grid, which limits the use to special applications.

To summarize, most of these schemes are approximation schemes without the ability to provide hierarchical localization. The arc tree is both an approximation and a localization scheme, but there is no 3D analogue, and the bounding ellipses are computationally expensive. The iterative end-point fit procedures are approximation methods only, but can be combined with the the strip tree and prism tree for localization. This paper introduces a scheme in two and three dimensions that is hierarchical and is both an approximation and a localization scheme. Furthermore, its bounding areas/volumes are computationally cheaper than the strip and prisms in hierarchical operations such as point location and object intersection. Sections 3 and 4 present this scheme for the 2D and 3D cases. Section 5 presents some hierarchical operations that use this scheme. Section 6 concludes the paper.

3 Hierarchical approximation and localization in two dimensions

In this section we consider the approximation and localization of a 2D polygon of N_v vertices $v_0 \dots v_{N_v-1}$.The approximation algorithm is based on the way that a part of the polygon, a polyline $v_p \dots v_s$, is localized.

3.1 Localization

The definition of the bounding area is based on the notion of a disc with a signed radius, and the definition of the ± operator. The following definition is illustrated in Fig. 1.

Definition 1. Disc with signed radius: let a_1, a_2, and b be points not lying on one line. The radius $R(a_1, a_2; b)$ of the disc $D(a_1, a_2; b)$ touching these points has a *negative sign* if $\angle(a_1, b, a_2)>\pi/2$, i.e., the center of the disc and b lie at opposite sides of the line through a_1 and a_2; otherwise it has a *positive sign*.
The following definition is illustrated in Fig. 2.

Definition 2. Let D_1 and D_2 be two discs with signed radii R_1 and R_2 respectively. The ± operator is defined as

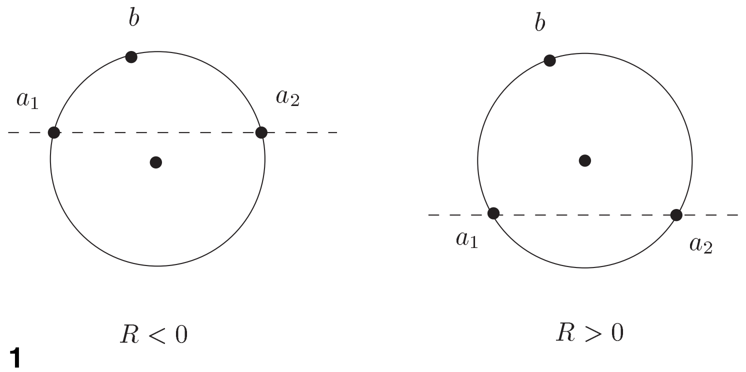
$$D_1 \pm D_2 = \begin{cases} D_1 \bigcap D_2 & \text{if } R_1, R_2 < 0, \quad \text{or} \\ & R_i < 0, R_j > 0, |R_i| > R_j, (i,j) \in \{(1,2),(2,1)\}, \\ D_1 \bigcup D_2 & \text{if } R_1, R_2 > 0, \quad \text{or} \\ & R_i < 0, R_j < 0, |R_i| < R_j, (i,j) \in \{(1,2),(2,1)\}. \end{cases}$$

The definition may look unnecessarily complex, but is used to make the definition of the bounding area simple.
The half-plane that contains point c and whose boundary passes through a_1 and a_2 is denoted by $H(a_1, a_2; c)$. A half-plane is considered as a disc with a radius of $-\infty$.
The bounding volume that is the basis of our approximation algorithm is defined in terms of discs with signed radii and the ± operator. Informally, the bounding volume of a polyline $v_p \dots v_s$ is the smallest intersection or union of two discs touching v_p and v_s that contains the polyline, and is denoted by $BV(v_p \dots v_s)$. The following definition is illustrated in Fig. 3 for the case in which the BV is an intersection of two discs or a disc and a half-plane.

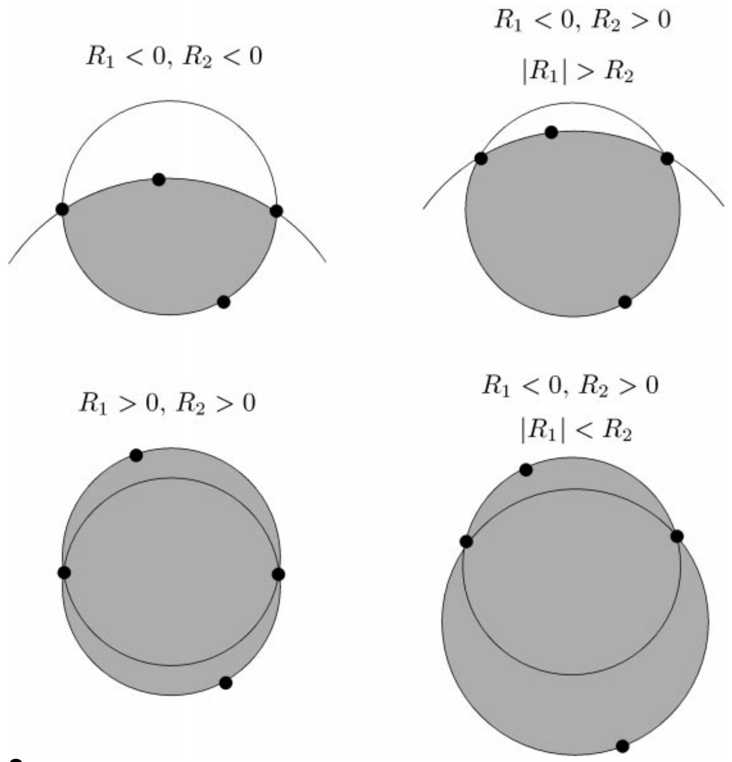
Definition 3. Let P be polyline $v_p \dots v_s$, $s>p+1$. If points of P lie both to the left and right of the directed line through v_p and v_s, then let v_q be such that $D(v_p, v_s; v_q)$ contains all vertices lying in $H(v_p, v_s; v_q)$; let v_r be such that $D(v_p, v_s; v_r)$ contains all vertices lying in $H(v_p, v_s; v_r)$, and define $BV(P)=D(v_p, v_s; v_q)\pm D(v_p, v_s; v_r)$. Otherwise, let v_q be such that $D(v_p, v_s; v_q)$ contains all vertices of P, and define $BV(P)=D(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$.
It is easily verified that $P\subset BV(P)$, so that $BV(P)$ is a bounding volume for P by construction.
Note that v_q and v_r minimize $\angle(v_p, v_i, v_s)$ for all v_i lying at one side. Note further that such a vertex v_q always exists. After all, there is a disc touching v_p and v_s that contains all vertices of P in $H(v_p, v_s; v_i)$ for some $p<i<s$. The smallest possible disc touches at least one vertex v_j, $p<j<s$. We call one of these vertices v_q. Symmetrically, if not all vertices of P lie in $H(v_p, v_s; v_q)$, then the v_r in the definition also exists.

3.2 Approximation

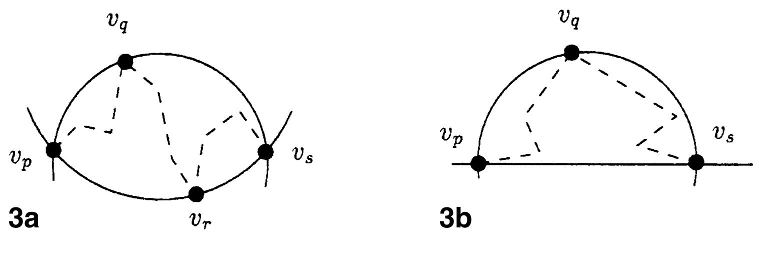
The hierarchical approximation of a polygon of N_v vertices starts with the calculation of the smallest bounding disc (SBD), that is, the smallest disc that contains all vertices. This disc touches at least two vertices, say v_i and v_j, $i<j$. If more than two vertices lie on the boundary of the smallest bounding disc, we take the two vertices that are farthest apart. Edge v_iv_j is the zeroth order approximation of the polygon, dividing it into two polylines $v_i v_{i+1} \dots v_j$ and $v_jv_{j+1} \dots v_i$ (here and in the rest of Sect. 3 the indices are taken modulo N_v).
For the next level in the hierarchy of approximations, let us consider a polyline $P=v_p \dots v_s$. The approximation depends on the bounding volume $BV(P)$. If $BV(P)$ is $D(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$, then P is approximated by the edges v_pv_q and v_qv_s. If $BV(P)$ is $D(v_p, v_s; v_q)\pm D(v_p, v_s; v_r)$ and if the radius of $D(v_p, v_s; v_q)$ is larger than the radius of $D(v_p, v_s; v_r)$, then P is approximated by the edges



1



2

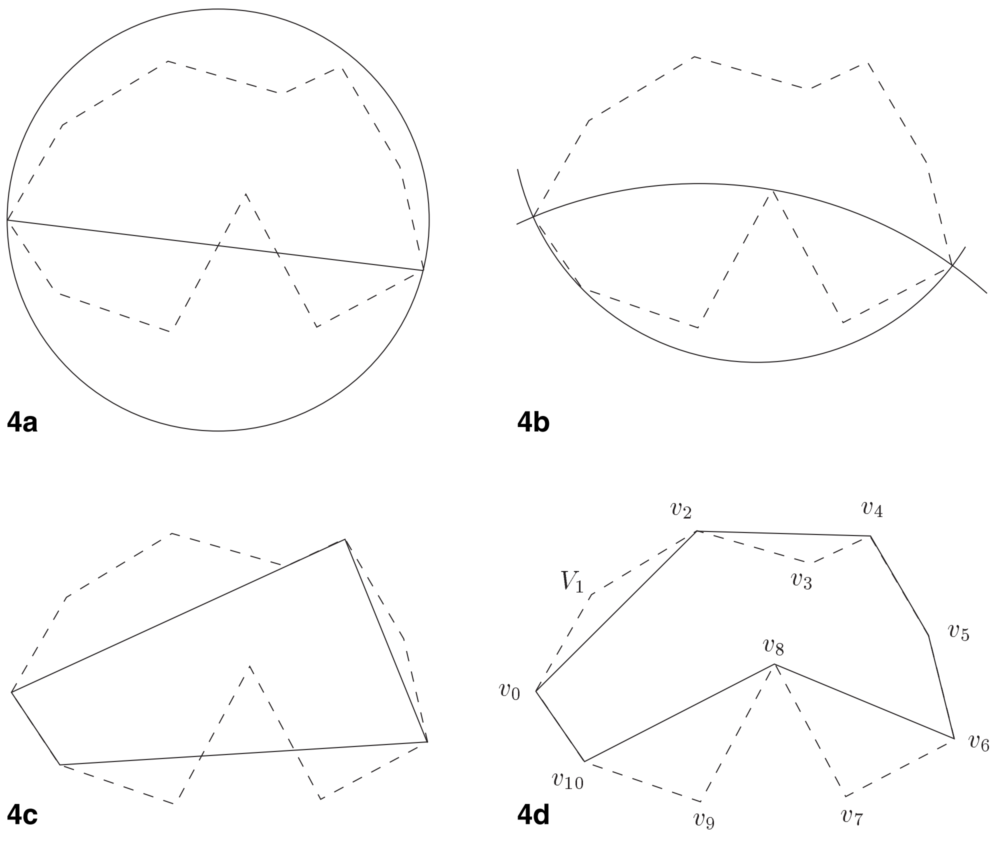


3a 3b

Fig. 1. Disc $D(a_1, a_2; b)$ with signed radius

Fig. 2. The ± operator on two discs D_1 and D_2

Fig. 3a, b. 2D bounding volumes: **a** $F(P)=D(v_p, v_s; v_q)\pm D(v_p, v_s; v_r)$. **b** $F(P)=D(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$



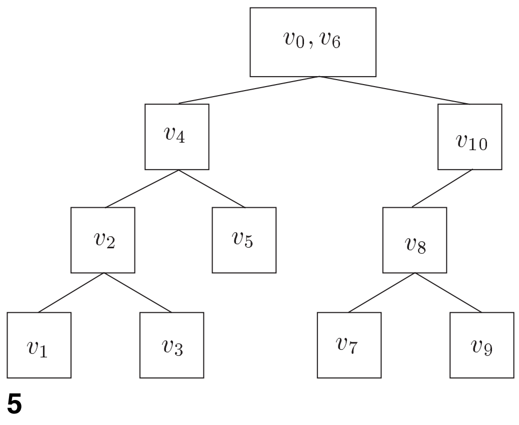


Fig. 4. **a** Smallest bounding disc and zeroth order approximation; **b** bounding volume of $v_6 \ldots v_0$; **c, d** first and second order approximations

Fig. 5. HAL tree of the example in Fig. 4

$v_p v_q$ and $v_q v_s$; otherwise, by $v_p v_r$ and $v_r v_s$. Remember that the radii of $D(v_p, v_s; v_q)$ and $D(v_p, v_s; v_r)$ are signed.

An edge $v_p v_s$ is not subdivided if $p+1=s$, in which case this edge is in the original polygon. In order to construct the complete hierarchy, the iteration continues until all vertices are contained in the approximation polygon; the last approximation coincides with the original polygon. By definition, an edge $v_p v_{p+1}$ has no bounding volume. Figure 4 gives a simple example of a polygon and the approximations of levels zero, one, and two.

A binary tree is a natural data structure for storing the hierarchical approximations. The root, level zero of the tree, contains vertices v_i and v_j. The left subtree stores vertices $v_{i+1},\dots, v_{j-1}$ so that the symmetric or infix order traversal yields the successive vertices of the polygon. The right subtree stores vertices $v_{j+1},\dots, v_{i-1}$ analogously. A level-ℓ approximation simply corresponds to the levels $0,\dots, \ell$ of the tree. Figure 5 shows the complete hierarchy tree of the example in Fig. 4.

The bounding areas are stored as follows. The root stores the smallest bounding disc. A node containing vertex v_q at level ℓ, $\ell\geq 1$ of the tree, contains $D(v_p, v_s; v_q)$ and $D(v_p, v_s; v_r)$, where v_p is the predecessor and v_s, the successor of v_q at approximation level ℓ. For example, the node containing v_2 in Fig. 5 stores $D(v_0, v_4; v_2)$ and $D(v_0, v_4; v_3)$.

The hierarchical approximation and localization scheme is referred to as the HAL scheme, and the tree that stores the hierarchy of approximations and bounding volumes, a HAL tree, also in three dimensions.

The following theorem shows that all *BV*s are the *intersections* of two discs or a disc and a half-plane, and never a union of them.

Theorem 1. *Starting with a polygon, all the BVs in the HAL tree are the intersection of two discs or a disc and a half-plane.*

Proof. Let the polygon be $v_0 \dots v_{N_v-1}$. The initial *BV* is the intersection of the SBD with itself. At every next level, if there is a disc touching v_p and v_s that contains $v_{p+1} \dots v_{s-1}$, then the bounding volume can be made smaller by intersecting two discs. By construction of the approximation, each approximated polyline is contained in a disc. Thus, each *BV* is the intersection of two discs or a disc and a half-plane. In particular, the case in which a *BV* is the union of a half-plane and a disc simply cannot occur. That would be the case if a vertex v_i, $p<i<s$ were to lie on the line through v_p and v_s, outside the line segment v_pv_s, but by construction of the approximation, this never happens. □

The algorithm to construct the complete HAL tree is summarized in Algorithm 1 in pseudo C code. The time complexities for the best and worst cases, as well as the storage complexity, are given by the following theorems.

Algorithm 1. 2D HAL tree construction algorithm.

```
HAL2()
{ bool Completed=FALSE;

  compute SBD($v_i$, $v_j$);                    // smallest bounding disc
  store $v_i$, $v_j$, and SBD in root;
  while (!Completed)
  {
     Completed=TRUE;
     for (each segment $v_pv_s$ approximating $P$, $p+1<s$)
                                                 // segment to be split
     {
        Completed=FALSE;
        determine $F(P)$;
        if ($F(P)$==$D(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$)
          store $v_q$, $D(v_p, v_s; v_q)$, and $D(v_p, v_s; v_r)$ in new node;
        else                  // $F(P)$ is $D(v_p, v_s; v_q)\pm D(v_p, v_s; v_r)$
          if ($R(D(v_p, v_s; v_q))>R(D(v_p, v_s; v_r))$)
            store $v_q$, $D(v_p, v_s; v_q)$, and $D(v_p, v_s; v_r)$ in new node;
          else
            store $v_r$, $D(v_p, v_s; v_q)$, and $D(v_p, v_s; v_r)$ in new node;
     }
  }
}
```

Theorem 2. *Let N_v be the number of vertices in the original polygon. The best-case time complexity for constructing the complete 2D HAL tree is $\Theta(N_v \log N_v)$. The worst-case time complexity is $\Theta\left(N_v^2\right)$.*

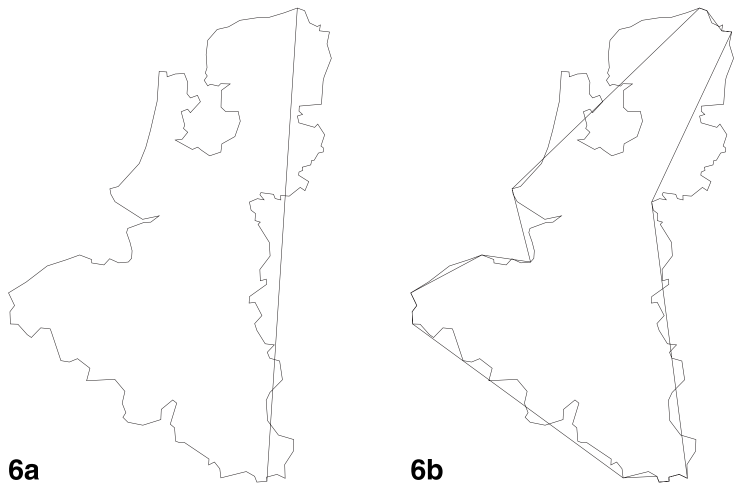
Proof. The smallest bounding disc can be found in $\Theta(N_v)$ time (Megiddo 1983). Two vertices on the boundary of the disc that are farthest apart can be found in $\Theta(N_v \log N_v)$ time (Preparata and Shamos 1985). The new vertex to be included in the approximation polygon can be found done in $\Theta(n)$ time for a polyline of n segments by testing the vertices sequentially.

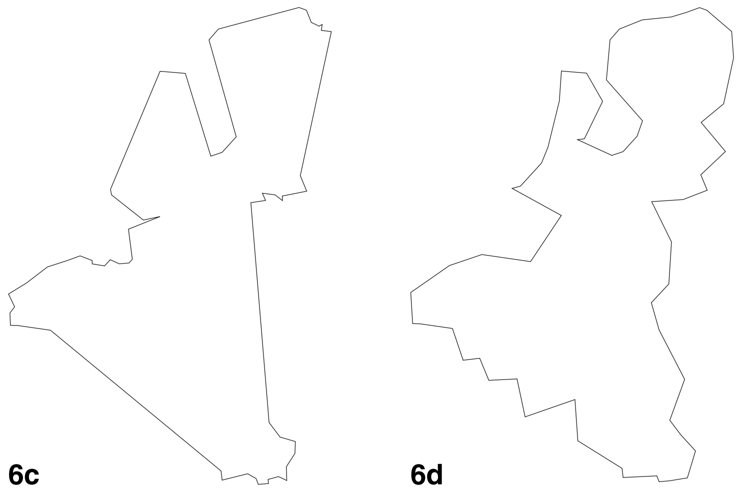
In the best case, the polygon is split into equally sized parts, giving $\lceil \log N_v \rceil$ iterations. The ith iteration then treats 2^i polylines of size $N_v/2^i$. The best-case time complexity is thus

$$\Theta(N_v) + \Theta\left(\sum_{i=0}^{\lceil \log N_v \rceil} 2^i \left(N_v/2^i\right)\right) = \Theta(N_v \log N_v).$$

In the worst case, a polyline of n segments is split into one polyline of size one and another of size $n-1$. The worst-case time complexity is therefore

$$\Theta(N_v) + \Theta\left(\sum_{i=1}^{N_v} i\right) = \Theta\left(N_v^2\right). \quad \square$$





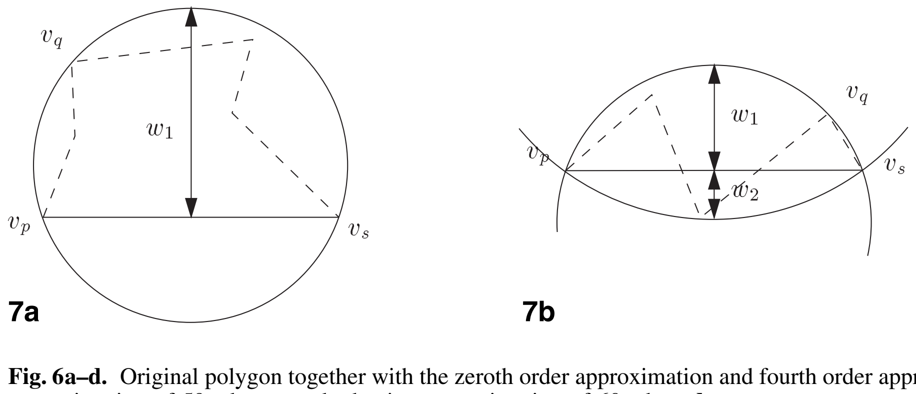


Fig. 6a–d. Original polygon together with the zeroth order approximation and fourth order approximation (**b**); higher order approximation of 59 edges **c** and adaptive approximation of 60 edges **d**

Fig. 7. **a** Width $w_1=R_1+h_1$ if $R_1>0$. **b** Width $w_1=-(R_1+h_1)$ if $R_1<0$

Theorem 3. *The storage complexity of the complete 2D HAL tree is* $\Theta(\mathrm{N}_v)$.

Proof. The HAL tree stores each vertex exactly once. The root stores two vertices and all other nodes store one vertex so that there are N_v-1 nodes. The HAL tree thus requires $\Theta(N_v)$ storage space. □

3.3 Adaptive approximation

The approximation algorithm described in the previous section replaces each edge in the current approximation by two new edges, unless the edge cannot be refined. One can apply this procedure a fixed number of times, or stop the iteration when the approximation polygon is within a specified error bound with respect to a chosen criterion. However, the error of one part of the approximation polygon can be very different from another part. This is clearly visible in Fig. 6, which shows the original polygon (the outer border of the mainland of the Benelux) with the zero-order level of approximation and two other levels of approximation. Some parts of the approximation polygon exhibit much more detail than other parts. The approximation algorithm can overcome this unevenness by only refining an edge if the error of *that edge* (rather than the whole polygon) is larger than a given bound. Such an algorithm is said to be adaptive.
In principle, any error criterion can be used. An approximation error tailored to a covering by *BV*s is based on the two widths associated with a *BV* (illustrated in Fig. 7):

Definition 4. Let $P=v_p \ldots v_s$ have an associated bounding volume $BV(P)=D(v_p, v_s; v_q)D(v_p, v_s; v_r)$ or $BV(P)=\mathrm{D}(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$. The widths w_1 and w_2 of the *BV* are defined as follows. Let w_1 be the largest distance from edge v_pv_s to a point on the boundary of $D(v_p, v_s; v_q)$ at the same side of v_pv_s as v_q. If $BV(P)=\mathrm{D}(v_p, v_s; v_q)\cap D(v_p, v_s; v_r)$, then w_2 is the largest distance from edge v_pv_s to a point on the boundary of $D(v_p, v_s; v_r)$ at the same side of v_pv_s as v_r. If $BV(P)=\mathrm{D}(v_p, v_s; v_q)\cap H(v_p, v_s; v_q)$, then $w_2=0$.
Note that the zero width associated with $H(v_p, v_s; v_q)$ agrees with the interpretation that the half-plane is a disc with a radius of $-\infty$. Verbally, the widths of a *BV* are the largest distances between the boundaries of the discs and the line through v_p and v_s. The covering approximation error of a single *BV* is now defined as $\max\{w_1, w_2\}$. The error of approximation of a polyline by BV_i, $i=0,\ldots, m$ with widths $w_{i,1}$ and $w_{i,2}$ is the maximum over all widths:

$$\max_{0\le i\le m} \max\left\{w_{i,1}, w_{i,2}\right\}. \tag{1}$$

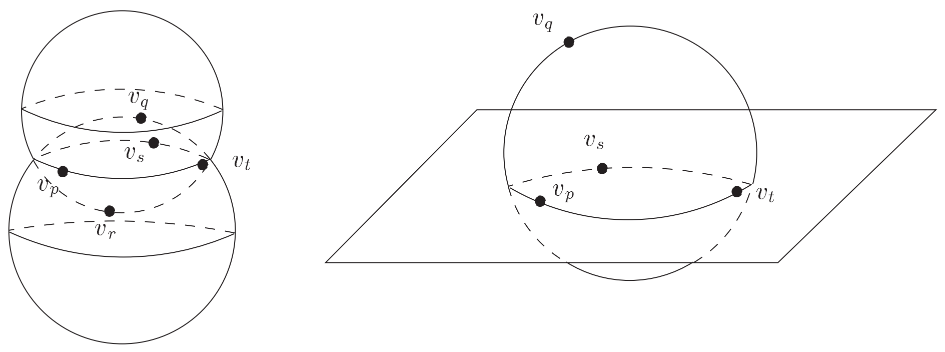
The adaptive approximation in Fig. 6 is constructed with this error criterion.
The tree representation of an adaptive approximation is a subgraph of the tree of the nonadaptive approximation of the same level. An adaptive approximation polygon often consists of fewer edges than a nonadaptive one of the same level of approximation. Alternatively, an adaptive approximation polygon often is of a higher approximation level than a nonadaptive one of about the same number of edges, usually resulting in a better approximation, i.e., a smaller approximation error. This is illustrated in the bottom row of Fig. 6 – the adaptive approximation and the nonadaptive one have about the same number of edges, but the adaptive approximation has a smaller error and is of a higher approximation level.

4 Hierarchical approximation and localization in three dimensions

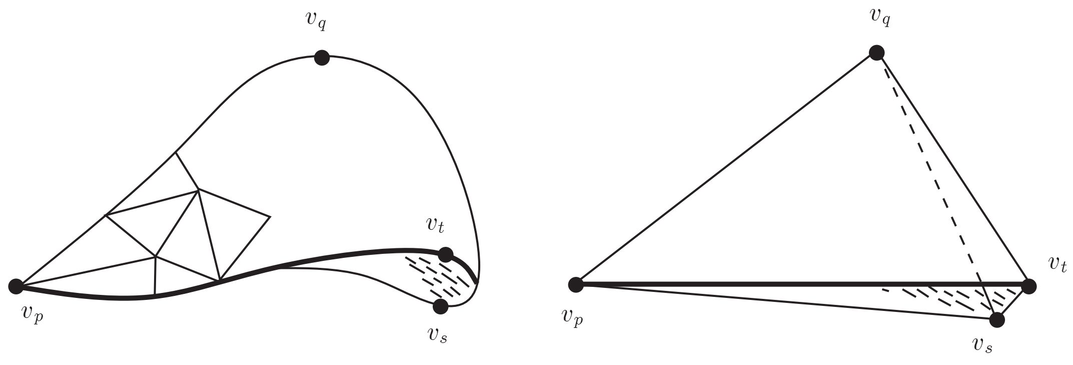
In this section we consider the approximation and localization of a 3D polyhedral solid object without voids and through-holes. That is, they have genus zero. In the following, the term ‘polyhedral surface’ denotes a part of the polyhedral boundary of the solid. N_v is the number of vertices of the polyhedron. The approximation algorithm is based on the way the polyhedral surfaces are localized.

4.1 Localization

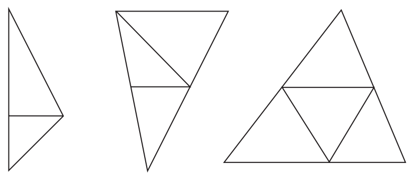
The definition of the bounding volume is based on the notion of a ball with a signed radius. $B(a_1, a_2, a_3; b)$ denotes the ball touching a_1, a_2, a_3, b with a signed radius, defined analogously to Definition 1. The ± operator for balls



8



9



10

Fig. 8. 3D bounding volume

Fig. 9. Triangle $v_pv_sv_t$ is split into three at a point inside the polyhedral surface

Fig. 10. Schematic splitting at sides

with a signed radius is defined by Definition 2 with 'ball' substituted for 'disc'. The half-space containing c whose boundary passes through a_1, a_2, and a_3, is denoted by $H(a_1, a_2, a_3; c)$. A half-space is considered as a ball with a radius of $-\infty$. Before presenting the approximation algorithm, the basic bounding volume must be defined. The following definition is illustrated in Fig. 8 for the case that the bounding volume is an intersection of two balls or a ball and a half-space.

Definition 5. Let P be a polyhedral surface, and v_p, v_s, and v_t, three distinct vertices on the polygonal boundary of P. Let v_q be a vertex of P such that $B(v_p, v_s, v_t; v_q)$ contains all vertices lying in $H(v_p, v_s, v_t; v_q)$. If $P \subset H(v_p, v_s, v_t; v_q)$, then the bounding volume BV of P is defined as $BV(P)=B(v_p, v_s, v_t; v_q) \cap H(v_p, v_s, v_t; v_q)$; otherwise, $BV(P)=B(v_p, v_s, v_t; v_q) \pm B(v_p, v_s, v_t; v_r)$, where v_r is a vertex of P not in $H(v_p, v_s, v_t; v_q)$ such that $B(v_p, v_s, v_t; v_r)$ contains all vertices in $H(v_p, v_s, v_t; v_r)$.

Note that v_q and v_r minimize the solid angle $\angle(v_i, v_p, v_s, v_t)$ at v_i for all v_i lying at one side.

Note that, analogous to the 2D situation, such a v_q always exists. If not all vertices of P lie in $H(v_p, v_s, v_t; v_q)$, such a v_r also exists. It is easily verified that $P \subset BV(P)$, so $BV(P)$ is a bounding volume for P by construction.

4.2 Approximation

The hierarchical approximation of a polyhedron starts with the calculation of one of the smallest balls that touches at least three vertices, say v_i, v_j, and v_k, and contains all vertices. Such a ball always exists, but it need not be the smallest bounding ball, which may touch only two vertices.

Next, we determine the three shortest paths of the edges in the polyhedron running between v_i and v_j, v_j and v_k, and between v_k and v_i. Note that a shortest path between two vertices v_i and v_j may not be unique. However, we always use the same shortest path-finding algorithm in the same direction, i.e., from v_i to v_j if $i<j$. Note further that pairs of shortest paths may partly coincide.

The three paths cut the polyhedron boundary open. They form the polygonal boundary of one or two polyhedral surfaces. Typically, there is a part of one of these paths that does not coincide with a part of one of the two other paths, and the polyhedron boundary is divided into two polyhedral surfaces; otherwise, only one polyhedral surface results. Triangle $v_iv_jv_k$ is the zeroth order approximation of the polyhedron.

At all next levels of the hierarchical approximation, we consider a polyhedral surface P of more than three vertices, and three distinct vertices v_p, v_s, and v_t on the boundary of P. If $BV(P)=B(v_p, v_s, v_t; v_q)\cap H(v_p, v_s, v_t; v_q)$, then P is approximated by the three triangles $v_pv_qv_s$, $v_qv_sv_t$, and $v_pv_qv_t$. If $BV(P)=B(v_p, v_s, v_t; v_q)\pm B(v_p, v_s, v_t; v_r)$ and if the signed radius of $B(v_p, v_s, v_t; v_q)$ is larger than the radius of $B(v_p, v_s, v_t; v_r)$, then P is approximated by $v_pv_qv_s$, $v_qv_sv_t$, and $v_pv_qv_t$, as illustrated in Fig. 9. Otherwise P is approximated by $v_pv_rv_s$, $v_rv_sv_t$, and $v_pv_rv_t$.

At all levels of approximation, there is a polyhedral surface, part of the original polyhedron boundary, associated with each approximation triangle $v_pv_sv_t$. The boundary polygon of this polyhedral surface consists of three shortest paths between v_p, v_s, and v_t. Note again that pairs of shortest paths may partly coincide, especially near the vertices v_p, v_s, or v_t. This may even result in a situation in which only the boundary vertices are associated with the approximation triangle $v_pv_sv_t$. This has no effect on the combinatorial and topological integrity, however, and the refinement procedure will just continue at the following levels.

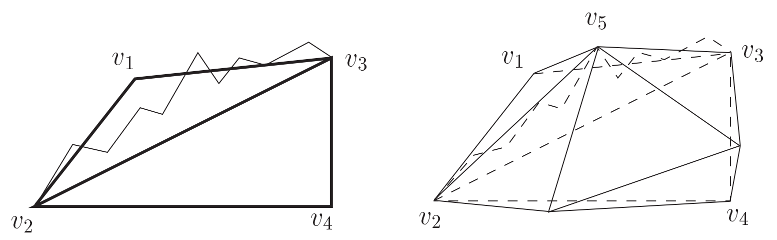
If the triangles are always split into three new ones as described, the new triangles become more and more elongated (‘slivers’), because each time the angles at the vertices are divided. Moreover, the edges are retained in all next levels, which will generally not lead to a good approximation. To avoid too thin triangles, the angle is not split if it is less than some chosen value, e.g., arbitrarily set to $\pi/4$. In such a case, the triangle is split at two sides as in Fig. 10 (middle). A triangle can also have two angles that are too small to be split, in which case the triangle is split at one side, as in Fig. 10 (left). If a triangle is split at a side, its neighboring triangle sharing that side is also split at the same point, thus avoiding cracks in the approximation polyhedron. A triangle can thus be forced to split, even if it has no small angles. Therefore, a triangle can also split at all three sides into four new triangles as illustrated in Fig. 10 (right).

The position where a side should be split is at a vertex on the shortest path between the end points. Such a path is part of the boundary polygon of the polyhedral surface that is approximated by the triangle. The vertex for splitting is the one that gives the smallest difference between the lengths of the new sides. If the shortest path consists of a single edge, there is no such vertex, and no splitting occurs. Thus, no new vertices are introduced. Such a split of a triangle into two new ones, together with its neighboring triangle, is shown in Fig. 11.

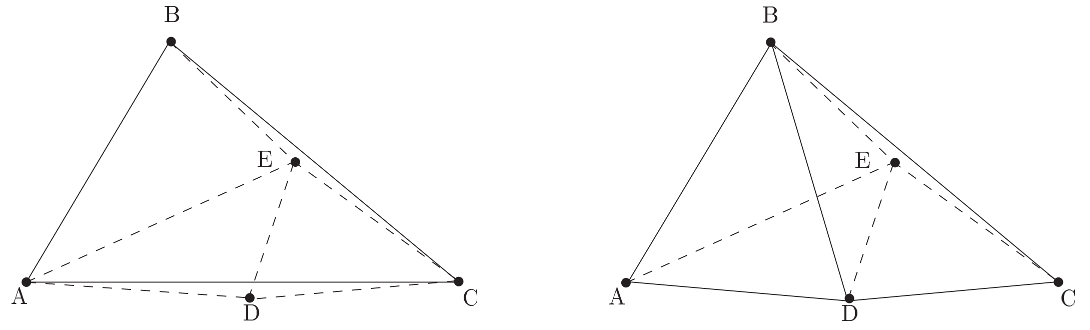
Because a triangle can be forced to split at a side by its neighbors, the effect of splitting at a side propagates through the current approximation polyhedron. Therefore, all edges in the current approximation polyhedron that must be split should be determined first. Only then is it known how all triangles should be split.

The refinement is iterated as long as the associated polyhedral surface, the boundary plus the interior vertices, consists of more than three vertices. Otherwise, the polyhedral surface contains just the three vertices forming the approximating triangle itself. This is the most detailed level of approximation, and the approximating triangle coincides with the original polyhedron facet. By definition, an approximation triangle at the lowest level has no bounding volume.

As stated before, two paths may partly coincide, which may affect the result of the splitting. If the vertex where a side is split is part of two paths,



11



12

Fig. 11. Two neighboring triangles are split at their common side

Fig. 12. Triangle *ABC* approximates original triangles drawn with dashed lines

then degenerate triangles will result. They approximate no part of the original surface. This, however, causes no problem, since the degenerate triangles can simply be recognized by their coalescing vertices.

Another result, however, is that the approximation may not be very accurate locally at an almost final approximation level. Look at the example in Fig. 12 (note that the dashed lines do not denote the method of triangle refinement, but represent the original triangles). A possible splitting sequence is the following. Triangle *ABC* approximates the original faces *ABE*, *ADE*, and *CDE*, and is split into *ABD* and *BCD*. In the next iteration, *ABD* is split into *ABE* and *ADE*, and *BCD* is split into *CDE* and *BDE*. However, the last triangle *BDE* approximates no part of the original surface. Triangles that do not approximate a part of the polyhedron have coalescing shortest paths between their vertices, which is easily tested. Such triangles are immediately discarded from the approximation.

Because no new vertices are introduced in splitting a triangle at a side, the shortest paths consist of edges of the original polyhedron. Because triangles that do not approximate a part of the polyhedron are discarded, all the final approximating triangles coincide with the original triangles. Thus, at the most detailed level of approximation, the original polyhedron is recovered. In order to avoid inaccurate approximations at an almost final level, as already illustrated, the split procedure should take the triangulation topology, and not only the geometry, into account. This is done in several other works mentioned in Sect. 2 that do not combine approximation with bounding volumes.

Let P be a polyhedral surface that is approximated by a triangle $v_p v_s v_t$. If a ball touching v_p, v_s, and v_t and containing P exists, then the '±' in the definition of the bounding volume is a '$\cap$'. If no sides of this triangle are split, the polyhedral surfaces associated with the new triangles are also contained in a single ball. However, if one or more sides are split, these polyhedral surfaces need not be contained in a single ball. Thus, unlike the 2D case, 3D bounding volumes are not always the intersection of two balls or a ball and a half-space, but may also be the union of two balls. There is no 3D analogue to Theorem 1. As a result, the bounding volume can degenerate to the union of a half-space and a ball. This will be the case when a vertex v_i of the approximated polyhedral surface lies in the plane through the vertices v_p, v_s, and v_t and outside the triangle $v_p v_s v_t$. In many data sets obtained from experimental applications, no four vertices are coplanar, so that this problem would not arise. By contrast, many synthetic data sets contain groups of four coplanar vertices.

In two dimensions, the HAL tree stores vertices, while the edges of successive approximations are implicitly defined. There is no simple analogous scheme in three dimensions that implicitly represents the triangles. They are therefore stored explicitly. The root contains the center and radius of the ball bounding the whole polyhedron. The two sons of the root contain v_i, v_j, and v_k, approximating the polyhedral surfaces P_1 and P_2, and the centers and signed radii of the balls that define the bounding volumes. All nodes except the root can have up to four sons, which are constructed as already described.

The algorithm to build the complete HAL tree is summarized in Algorithm 2 in pseudo C code. The time complexities of the algorithm for the best and worst cases, and the storage complexity are given by the following theorems.

Algorithm 2. 3D HAL tree construction algorithm

```
HAL3 ()
{ bool Completed=FALSE;

    compute BB-3 ($v_i, v_j, v_k$); // bounding ball touching 3 vertices
    store $v_i, v_j, v_k$ and BB-3 in root;
    while (!Completed)
   {
     Completed=TRUE;
     for (each $v_p v_s v_t$ approximating a $P$ of more than three
     vertices)
     {
        Completed=FALSE;
        determine the sides to split;
     }
     for (each $v_p v_s v_t$ approximating a $P$ of more than three
     vertices)
     {
        split triangle;
        divide $P$ accordingly;
        create children of $v_p v_s v_t$ and store new triangles and their
        bounding volumes;
     }
   }
}
```

Theorem 4. *Let N_v be the number of vertices in the original polyhedron. The best-case time complexity to construct the complete 3D HAL tree is $\Theta(N_v(\log N_v)^2)$. The worst case time complexity in 3D is $\Theta\left(N_v^2 \log N_v\right)$.*

Proof. The smallest bounding ball is found in $\Theta(N_v)$ time (Megiddo 1983). In three dimensions, this ball may touch only two vertices. Finding a third vertex such that the ball touching these three vertices contains them all takes another $\Theta(N_v)$ time. Finding the new vertex to be included in the approximation polyhedron takes $\Theta(n)$ time for a polyhedron of n vertices. Both operations can simply be performed by successively testing all candidates. Finding a shortest path in a polyhedron of n vertices takes $\Theta(n \log n)$ time (Dijkstra 1959).

In the best case, the polyhedral surface is split into equally sized parts, giving $\Theta(\log N_v)$ iterations. A polyhedral surface is split into at most four parts, but the order of complexity is not affected if we let the total number of polyhedra at the ith iteration be 2^i. The best-case complexity is therefore

$$\Theta(N_v) + \Theta\left(\sum_{i=0}^{\lceil \log N_v \rceil} 2^i \left(\left(N_v/2^i\right) \log\left(N_v/2^i\right)\right)\right)$$

$$= \Theta\left(N_v \sum_{i=0}^{\lceil \log N_v \rceil} i\right) = \Theta\left(N_v (\log N_v)^2\right).$$

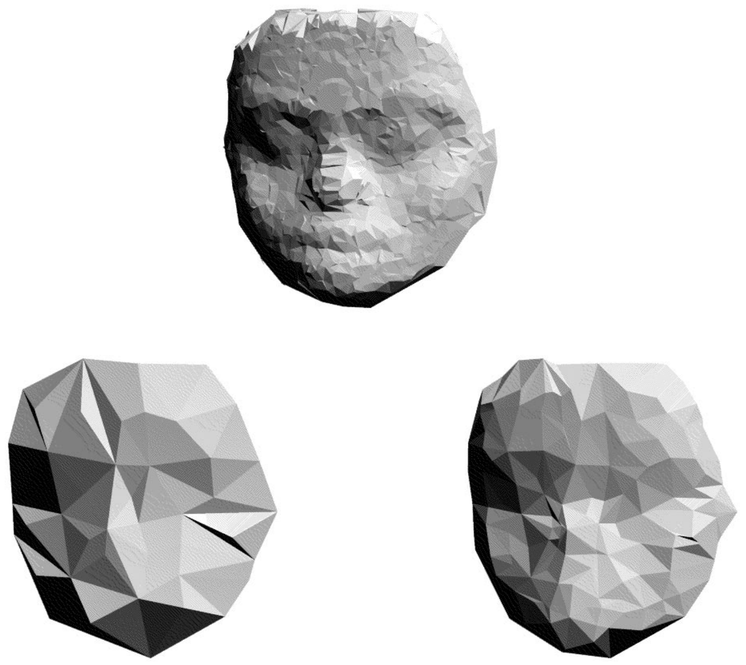
If there are only three polyhedron vertices associated with an approximation triangle, they form the approximation triangle, and no splitting occurs. If there are more than three vertices associated with an approximation triangle, then the subdivision will split off at least one vertex: v_q. In the worst case, the number of vertices of the polyhedral surface to be approximated decreases by one at each iteration.

The worst-case time complexity is therefore

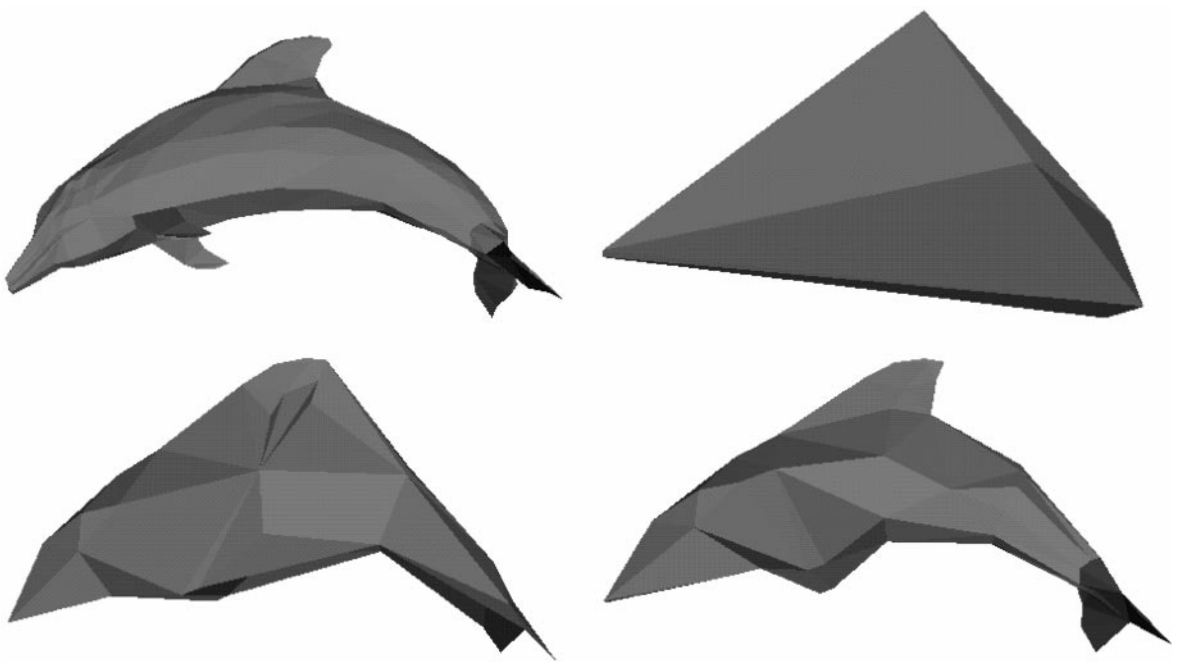
$$\Theta(N_v) + \Theta\left(\sum_{i=1}^{N_v} i \log i\right) = \Theta\left(\sum_{i=N_v/2}^{N_v} i \log i\right)$$

$$= \Theta\left(N_v^2 \log N_v\right).$$

Due to partly coinciding shortest paths, degenerate approximating triangles may be generated that do



13



14

Fig. 13. Polyhedral object of 2930 triangles and adaptive HAL approximations of 89 triangles and 277 triangles

Fig. 14. Polyhedral object of 595 triangles, and adaptive HAL approximations of 18, 109, and 162 triangles

not approximate a part of the original surface. If such triangles are treated at next levels, no vertex would be split off. Such triangles are discarded immediately, and do not contribute to the processing time at the next levels. At least one vertex is always split off from the approximated surface. Since they were created as one of the two to four new triangles, they do not make an extra contribution to the overall time complexity. □

Theorem 5. *Let N_t be the number of triangles in the original polyhedron. The storage complexity of the complete 3D HAL tree is $\Theta(N_t)$.*

Proof. Almost all internal nodes have two to four children. Only at the lowest levels are triangles possibly refined into one new triangle, so that only $\mathcal{O}(N_v)$ nodes have one child. Because the leaves of the tree contain the original N_t triangles, there are $\Theta(N_t)$ internal nodes. The total storage space is thus $\Theta(N_t)$. □

4.3 Adaptive approximation

The approximation error of a 3D covering is again $\max\{w_1, w_2\}$, and the widths are defined in complete analogy to the 2D case:

Definition 6. let a bounding volume BV of a polyhedral surface P be $BV(P)$=B$(v_p, v_s, v_t; v_q)\cap B(v_p, v_s, v_t; v_r)$ or $BV(P)=B(v_p, v_s, v_t; v_q)\cap H(v_p, v_s, v_t; v_q)$. The *widths* w_1 and w_2 are defined as follows. Let w_1 be the largest distance from the plane through $v_pv_sv_t$ to a point on the boundary of $B(v_p, v_s, v_t; v_q)$ at the same side of $v_pv_sv_t$ as v_q. If $BV(P)=B(v_p, v_s, v_t; v_q)\cap B(v_p, v_s, v_t; v_r)$, then w_2 is the largest distance from the plane through $v_pv_sv_t$ to a point on the boundary of $B(v_p, v_s, v_t; v_r)$ at the same side of $v_pv_sv_t$ as v_r. If $BV(P)=B(v_p, v_s, v_t; v_q)\cap H(v_p, v_s, v_t; v_q)$, then w_2=0.
The zero width associated with $H(v_p, v_s, v_t; v_q)$ agrees with the interpretation that the half-space is a ball with a radius of $-\infty$.
Figures 13 and 14 show examples of a polyhedral object and adaptive HAL approximations using this error criterion.

5 Hierarchical operations

This section shows two examples of operations that exploit the hierarchy of *BV*s and approximations. Where necessary, the representation is locally inspected at a level of more detail. In this section we use the term ball for both a 2D disc and a 3D ball unless we explicitly indicate otherwise. We also use the term ‘boundary segment’ for a 2D line segment and a 3D triangle.

5.1 Point inclusion

A point-in-polygon or point-in-polyhedron test determines whether a given point X is internal to a polygon or polyhedron P. One way to decide this is to count the number of intersections between P and any half-line emanating from X. See, for example, Preparata and Shamos (1985). If X is not on P, it is internal to P if the number of intersections is odd, and external otherwise. In order to count the number of intersections, one has to test the half-line against each boundary segment of P. This test is more efficient if we can use an approximation of P that yields the same answer to the test as P itself.
Let B be a part of P, A, an approximation of B connected with the rest of P without cracks, and BV, a bounding volume containing A and B (Fig. 15). If X is external to BV, then X is internal/external to P if and only if X is internal/external to P with B replaced by A. Indeed, if X is external to BV, then X does not lie between A and B. Therefore, the replacement does not change the result of the inclusion test.
The hierarchical point-location test starts at the root of the tree by testing if X is external to the bounding volume. If so, it is also external to P, otherwise the algorithm proceeds at the next level. If X lies outside a bounding volume, the approximation segment at that level can be used to test for the location; otherwise, the algorithm proceeds locally at the next level.
In two dimensions, the end points of a line segment and a neighboring segment at a more refined level always connect. However, in three dimensions the edge of a face and the edges of neighboring faces at a deeper level of approximation need not coincide, leaving a crack between them. To prevent the line through X from passing through the crack

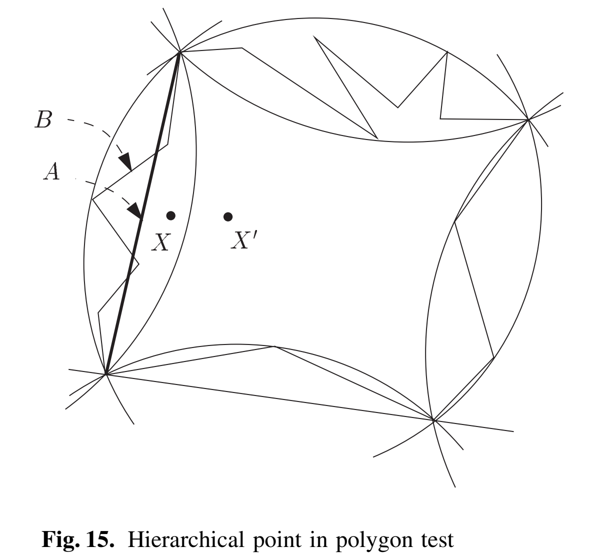


Fig. 15. Hierarchical point in polygon test

and failing to intersect the surface, the collection of approximation triangles must form a proper polyhedron.

For example, let A be an approximation triangle ($v_1v_2v_3$) (Fig. 11) with a neighboring triangle $A'=(v_2, v_3, v_4)$. Suppose that X lies inside the bounding volume of A, so that A cannot be used for the location test, and therefore ($v_1v_2v_5$) and ($v_1v_3v_5$) are tried. Even if X is not the bounding volume of A', we cannot use A' for the test, because there is a crack between ($v_1v_2v_5$), ($v_2v_3v_5$), and (v_2, v_3, v_4). Therefore, the refinement of A' is used. In this example, the other sides of A' are also split, so this has a propagating effect. Note that the actual refinements have already been done at the time of construction of the approximation tree.

5.2 Intersection

Another operation that can be efficient hierarchically is the intersection of two polygons or polyhedra. Detection and computation of polygonal or polyhedral intersections are fundamental problems in hidden surface elimination, motion planning, and linear programming. The HAL scheme can be exploited as follows. The hierarchical intersection algorithm first tests whether the bounding balls at the roots intersect. If they do not, then neither do the objects themselves. Otherwise, the algorithm proceeds with the next level of the deepest subtree and the same level of the other (sub)tree, testing all pairs of bounding volumes for intersection. This process continues locally for those pairs that intersect, and stops for those that do not. When bounding volumes at the lowest level are tested and found to intersect, the original boundary faces themselves are tested.

If bounding volume BV_1 is defined by balls B_1 and B_2, and bounding volume BV_2 by balls C_1 and C_2, the following combinations can occur:

1. $BV_1=B_1\cap B_2$, $BV_2=C_1\cap C_2$,
2. $BV_1=B_1\cup B_2$, $BV_2=C_1\cup C_2$,
3. $BV_1=B_1\cap B_2$, $BV_2=C_1\cup C_2$.

In these situations, BV_1 and BV_2 intersect in the following corresponding situations:

1. $B_1\cap C_1\neq\phi\wedge B_2\cap C_1\neq\phi\wedge B_1\cap C_2\neq\phi\wedge B_2\cap C_2\neq\phi$,
2. $B_1\cap C_1\neq\phi\vee B_2\cap C_1\neq\phi\wedge B_1\cap C_2\neq\phi\wedge B_2\cap C_2\neq\phi$,
3. $(B_1\cap C_1\neq\phi\wedge B_2\cap C_1\neq\phi)\wedge(B_1\cap C_2\neq\phi\wedge B_2\cap C_2\neq\phi)$.

For 2D discs, only situation 1 can occur.

Testing whether two balls intersect amounts to comparing the distance between their centers with the sum of the radii. If the distance is greater than the sum, they do not intersect; otherwise, they do intersect.

For intersection detection, the algorithm can stop as soon as a single intersection is detected. In order to actually compute the intersection, the iteration must be continued until all intersecting boundary faces are found, and then the actual intersection must be computed. After computing the intersections of the polygons or polyhedra, the solid object that is the intersection, union, or difference of the two objects can be determined in a manner similar to the methods described by Günther (1988) and Ponce and Faugeras (1987).

6 Concluding remarks

The HAL scheme has some advantages over other boundary-based schemes: it is hierarchical, it is an approximation as well as a localization scheme, and the bounding volumes are efficient in storage and for computations.

For both the point location and the intersection algorithm based on the HAL scheme, the basic operation is the computation of the distance between two points. This is very simple and computationally cheap. Especially in three dimensions it is much more efficient than other localization schemes. For example, a point-in-prism test needs to consider five faces, and a prism-prism intersection requires 25 polygon-polygon intersections. In two dimensions, the localization by ellipses in the arc-tree scheme was found to be more efficient than other schemes (Dominguez and Günther 1991). The point-in-*BV* test is as cheap as a point-in-ellipse test, which requires the distances of the point to the two focal points of the ellipse. However, the intersection of two *BV*s is cheaper than the intersection of two ellipses. It should be noted, though, that the performance of such hierarchical operations not only depends on the efficiency of calculations with a single bounding volume, but also on the quality of localization for the whole object.

The time complexity for the HAL tree construction is the same as for many of the schemes described in Sect. 2. The optimal approximations are more expensive, and the 2D Douglas-Peucker algorithm can be done in $\Theta(N \log N)$ time (Hershberger 1992) at the cost of an additional data structure. Testing for point inclusion and bounding volume intersection is done by calculating distances between points, which is computationally cheaper than the calculations on bounding volumes of many other schemes. For simplicity and efficiency of storage and computations a half-space in the definition of a *BV* can be represented by a ball of large radius. In order to compare the efficiency of the HAL scheme with the efficiency of other methods, these should also be implemented and applied to the same data sets. Dominguez and Günther (1991) make such a comparison of the arc tree, the strip tree, and approximation and localization by the Bézier scheme. Extending the comparison with the HAL scheme is a possible subject of future research.

Due to the nature of our scheme, it works only for objects without handles, i.e., of genus zero. For objects with holes, the inner boundary can be treated in same way separately. However, there is no guarantee that the approximated inner boundary will lie completely within the approximated outer boundary. Furthermore, even if the original object is a simple polygon or polyhedron (informally, the object boundary does not intersect itself), the HAL scheme approximation need not be simple. The polygon or polyhedron may intersect itself, especially at the first few approximation levels. At higher levels of approximation, however, this will rarely happen. All approximation schemes in Sect. 2 except the Delaunay pyramid, which can only be used for $2\frac{1}{2}$D polyhedra, suffer from the same problem. A future research direction is the development of a hierarchical approximation and localization scheme that always yields simple approximation polygons or polyhedra when the original is simple.

As mentioned in Sect. 4.2, to avoid inaccurate approximations at some levels, the splitting procedure should take the triangulation network topology, and not only the geometry, into account. This is an interesting topic for further work, but is more difficult in combination with bounding volumes than for approximation alone. The latter has already been done by several others mentioned in Sect. 2.

Acknowledgements. The constructive comments of the reviewers is much appreciated. They led to a better manuscript.

References

Ballard DH (1981) Strip trees: a hierarchical representation for curves. Commun ACM 24:310–321

Berg M de (1997) Visualization of TINs. In: Kreveld M van, Nievergelt J, Roos T, Widmayer P (eds) Algorithmic foundations of geographic information systems (Lecture Notes in Computer Science, vol. 1340). Springer, Berlin Heidelberg New York, pp 79–97

DeFloriani L (1989) A pyramidal data structure for triangle-based surface description. IEEE Comput Graph Appl 9:67–80

DeHaemer Jr. MJ, Zyda MJ (1991) Simplification of objects rendered by polygonal approximations. Comput and Graph 15:175–184

Dijkstra EW (1959) A note on two problems in connection with graphs. Numerische Mathematik 1:269–271

Dominguez S, Günther O (1991) Performance analysis of three curve representation schemes. In: Bieri H, Noltemeier H (eds) Computational geometry – methods, algorithms and applications, Proceedings of the International Workshop on Computational Geometry CG'91, Bern, Switzerland (Lecture Notes in Computer Science, vol 553). Springer, Berlin Heidelberg New York, pp 37–56

Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Canadian Cartog 10:112–122

Duda RO, Hart PE (1973) Pattern classification and scene analysis. John Wiley, New York

Faugeras OD, Hebert M, Mussi P, Boissonnat J-D (1984) Polyhedral approximation of 3-D objects without holes. Comput Vision Graph Image Processing 25:169–183

Günther O (1987) Efficient structures for geometric data management (Lecture Notes in Computer Sciences, vol 337). Springer, Berlin Heidelberg New York

Hershberger J, Snoeyink J (1992) Speeding up the Douglas-Peucker line simplification algorithm. Proceedings of the 5th International Symposium on Spatial Data Handling, IGU Commission on GIS, Charleston, South Carolina. University of South Carolina, Columbia, SC, 134–143

Hoppe H (1997) View-dependent refinement of progressive meshes. Comput Graph 31:189-198

Hoppe H (1996) Progressive meshes. Comput Graph 30:99–108

Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1993) Mesh optimization. Comput Graph 27:19–26

Imai H, Iri M (1988) Polygonal approximations of a curve – formulations and algorithms. In: Toussaint GT (ed) Computational morphology – a computational geometric approach to the analysis of form. North-Holland, pp 71–86

Luebke D, Erikson C (1997) View-dependent simplification of arbitrary polygonal environments. Comput Graph 31:199–208

Megiddo N (1983) Linear time algorithms for linear programming in R^3 and related problems. SIAM J Comput 12:759–776

Ponce J, Faugeras O (1987) An object centered hierarchical representation for 3D objects: the prism tree. Comput Vision Graph Image Processing 38:1–28

Preparata FP, Shamos MI (1985) Computational geometry: an introduction. Springer, Berlin Heidelberg New York

Schmitt F, Gholizadeh B (1986) Adaptive polyhedral approximation of digitized surfaces. Proceedings of the SPIE – The International Society for Optical Engineering 595:101–108

Schroeder W (1992) Decimation of triangle meshes. Comput Graph 26:65–70

Turk G (1992) Re-tiling polygonal surfaces. Comput Graph 26:55–64

REMCO C. VELTKAMP is Assistant Professor at Utrecht University. He obtained his PhD in 1992 at Erasmus University Rotterdam. His current research focuses on the application of computational geometry in computer vision and shape processing, such as shape matching, reconstruction, and curve and surface modeling. Aspects like algorithmic design and experimental verification play an important role in his work.