Convex Hulls in a 3-dimensional Space

Vladimir Kovalevsky[1] and Henrik Schulz[2]

[1] Berlin, Germany
kovalev@tfh-berlin.de
[2] Dresden University of Technology, Dresden, Germany
hs24@inf.tu-dresden.de

Abstract. This paper describes a new algorithm of computing the convex hull of a 3-dimensional object. The convex hull generated by this algorithm is an abstract polyhedron being described by a new data structure, the cell list, suggested by one of the authors. The correctness of the algorithm is proved and experimental results are presented.

1 Introduction

The convex hull is a good tool to economically describe a convex object. We present here a new method to compute the convex hull of a three-dimensional digital object using a new data structure, the two-dimensional cell list, suggested by one of the authors [Kov89]. The cell list is well suited to efficiently perform intermediate steps of the algorithm and to economically encode the convex hull.

Since the convex hull is often used in an great number of applications it is an often treated problem in many articles and books on computational geometry. Preparata and Shamos [Pre85] describe a 3D convex hull algorithm of the devide-and-conquer type. The first step in this approach consists in sorting the given points by one of their coordinates. After sorting, the convex hull is computed by a recursive function consisting of two parts: generation of the convex hull of a small subset of points and merging two convex hulls. Since the set of points is sorted, every two subsets are non-intersecting polytopes.

Other approaches such as [Ber00,Cla92] use an incremental method. According to [Ber00] an initial polyhedron of four points is created and then modified by taking the given points into the polyhedron in a random order until it contains the whole set of points. The algorithm connects each point P with the edges of the "horizon", i.e. of the boundary of the subset of faces visible from P. The convex hull is described by a doubly connected edge list. It should be mentioned that the most of these approaches use only simplicial polyhedrons, i.e. the convex hull of the point set is a triangulation

The algorithm described in this paper can construct the convex hull of any finite set of points which are given by their Cartesian coordinates. However, in the case of a set of voxels, consisting of a few connected components, the algorithm may be essentially accelerated. The idea of this acceleration is based on the fact, that the convex hull of the whole set is equal to the convex hull of a relatively small subset of "distiguished" voxels or of their "distinguished" vertices.

Let us present some basic definitions necessary for the following sections.

The description of the algorithm and the proof of its correctness are based on the theory of abstract cell complexes (AC complexes) [Kov89]. To remind the reader some basic notions of this theory we have gathered the most important definitions in the Appendix.

Let V be a given set of voxels in a Cartesian three-dimensional space. The voxels of V are specified by their coordinates. In particular the set V may be specified by labeling some elements of a three-dimensional array of bytes or bits. Our aim is

to construct the convex hull of the set V while remaining in the frame of abstract cell complexes without using notions from the Euclidean geometry. We consider the convex hull as an abstract polyhedron according to the following definition:

Definition AP: An *abstract polyhedron* is a three-dimensional AC complex containing a single three-dimensional cell whose boundary is a two-dimensional combinatorial manifold without boundary (see Appendix). The two-dimensional cells (2-cells) of the polyhedron are its *faces*, the one-dimensional cells (1-cells) are its *edges* and the zero-dimensional cells (0-cells) are its *vertices* or points.

An abstract polyhedron is called a *geometric* one if coordinates are assigned to each of its vertices. We shall call an abstract geometric polyhedron an AG-polyhedron. Each face of an AG-polyhedron PG must be planar. This means that the coordinates of all 0-cells belonging to the boundary of a face F_i of PG must satisfy a linear equation $H_i(x, y, z) = 0$. If these coordinates are coordinates of some cells of a Cartesian AC complex A then we say that the polyhedron PG is embedded into A or that A contains the polyhedron PG.

Definition CP: An AG-polyhedron PG is called *convex* if the coordinates of each vertex of PG satisfy all the linear inequalities $H_i(x, y, z) \leq 0$ corresponding to all faces F_i of PG. The coefficients of the linear form $H_i(x, y, z)$ are the components of the outer normal of F_i.

A cell c of the complex A containing the convex AG-polyhedron PG is said to *lie in* PG if the coordinates of c satisfy all the inequalities $H_i(x, y, z) \leq 0$ of all faces F_i of PG.

Definition CH: The *convex hull* of a finite set S of points is the smallest convex AG-polyhedron PG containing all points of the set S. "Smallest" means that there exists no convex AG-polyhedron different from PG which contains all points of S and whose all vertices are in PG.

The aim of this paper is to describe the algorithm which constructs the convex hull of an arbitrary set of voxels given by their Cartesian coordinates in a three-dimensional space.

2 The Algorithm

In this section we describe the algorithm of constructing the convex hull of a set V of voxels given as a three-dimensional array in which the voxels of V are labeled. As already mentioned in the Introduction, the algorithm may be essentially accelerated if the set V consists of a few connected components. We shall demonstrate in the next section that the convex hull of the whole set V is equal to the convex hull of a relatively small subset of the so called *local corners*. It is also possible, if desired, to construct the convex hull of the vertices of the voxels, while considering each voxel as a small cube. The latter convex hull is slightly greater than the first one: it contains the cubes completely. From this point of view the first convex hull is that of the centers of voxels. Fig.1 illustrates the idea of these two convex hulls for the 2D case.

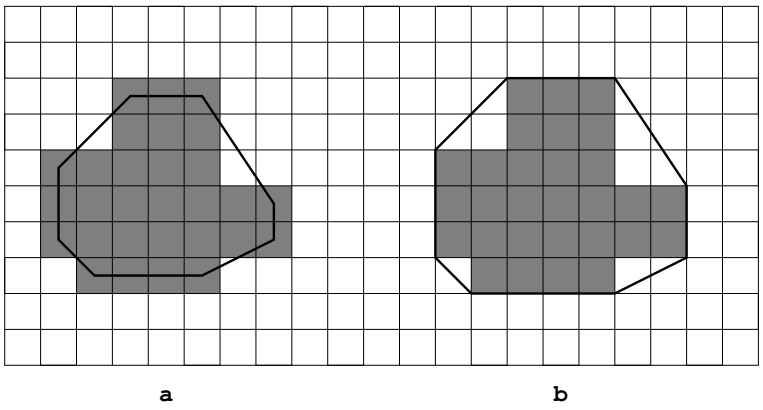

Fig.1: The convex hull of pixels (a) and that of their vertices (b)

Our algorithm for constructing the convex hull consists of two parts: in the first part a subset of vectors (pointing either to voxels or to the vertices) must be found which are candidates for the vertices of the convex hull. The coordinates of the candidates are saved in an array L. The second part constructs the convex hull of the set L.

A vector $\boldsymbol{v}$ is obviously not suitable as a candidate for a vertex if in the given set V there are two other vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ such that $\boldsymbol{v}$ lies on the straight line segment connecting $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, i.e. the vector $\boldsymbol{v}$ is a convex combination of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. A vector $\boldsymbol{v}$ is also not suitable as a candidate for a vertex if it may be represented as a convex combination of more than two, say of n voxels $\boldsymbol{v}_i$. Then the vector $\boldsymbol{v}$ may be represented as

$$\boldsymbol{v} = \sum \alpha_i \cdot v_i \quad \text{with } 0 \leq \alpha_i \leq 1 \text{ and } \sum \alpha_i = 1; \quad i = 1, 2, ..., n \tag{1}$$

It is hardly reasonable to test each vector of V whether it is a convex combination of some other vectors, since this problem is equivalent to that of constructing the convex hull. Really, if one knows the convex hull of V, then one also knows the vertices of the convex hull. The vertices are the only vectors which are no convex combinations of other vectors.

However, it is reasonable to test each vector, whether it is a convex combination of vectors *in its neighborhood*. We call a vector which is no convex combination of vectors in its neighborhood a *local corner* of V. The greater the tested neighborhood the less the number of local corners found. Thus, for example, a digital ball B of diameter 32 contains 17077 voxels. The number of local corners of B is 776 when testing 6 neighbors and 360 when testing 26 neighbors of each voxel. We have decided to test 26 neighbors.

When testing a vector $\boldsymbol{v}$ with coordinates (x, y, z) the procedure calculates the coordinates of 13 pairs of vectors from the neighborhood of $\boldsymbol{v}$. The vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of each pair are symmetric with respect to $\boldsymbol{v}$, e.g. $\boldsymbol{v}_1 = (x-1, y-1, z-1)$ and $\boldsymbol{v}_2 = (x+1, y+1, z+1)$. If both vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are in the set V then the vector $\boldsymbol{v}$ is not a local corner and will be dropped. Only if in each of the 13 pairs the number of vectors belonging to V is 0 or 1, the vector $\boldsymbol{v}$ is recognized as a local corner. The procedure saves the coordinates of each local corner in an array L.

We prefer to consider the problem of recognizing the local corners from the point of view of AC complexes. This gives us the possibility to uniformly examine two different problems: that of constructing the convex hull either of a set of voxels or of the vertices of the voxels.

From the point of view of AC complexes the given set V is the set of three-dimensional cells (3-cells) of a subcomplex M of a three-dimensional Cartesian AC

complex A. The complex A represents the topological space in which our procedure is acting. It is reasonable to accept that M is homogeneously three-dimensional (see the Appendix). This means that each cell of M whose dimension is less than 3 is incident to at least one 3-cell of M. With other words, M has no "loose" cells of dimensions less than 3.

Under these assumptions our problem consists in constructing the convex hull either of the interior $Int(M)$ or of the closure $Cl(M)$ of M. The first case corresponds to the convex hull of the set of voxels and the second case to that of the vertices of voxels.

The development of algorithms in digital geometry, to which our problem obviously belongs, and the proof of their correctness becomes easier and more descriptive when using topological coordinates (see Appendix and [Kov01]) rather than standard ones. Using topological coordinates enables one to recognize the dimension of a cell from its coordinates, to easily calculate the coordinates of cells incident to a given cell etc. However, the representation of a Cartesian AC complex in a topological raster [Kov01] demands 8 times more memory space than its representation in a standard raster, where only the voxels get memory elements assigned. The best way to proceed consists in using the standard raster in the computer program, while thinking about the problem in terms of a topological raster.

Consider now the problem of finding the 0-cells which are local corners of the closure $Cl(M)$. When considering only 6 neighbors of a 0-cell then it is easily seen that a 0-cell is a local corner iff it is incident to a *single* 3-cell of M.

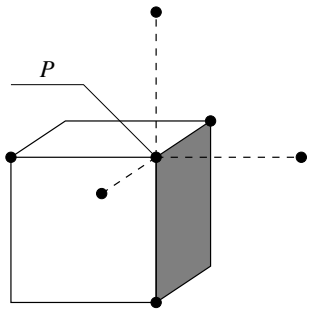

Fig.2: A 0-cell is a local corner iff it bounds a single 3-cell of M.

Really, consider Fig.2. Here we represent the 0-cells by small dark disks, the 1-cells by line segments, the 2-cells by squares and the 3-cells by cubes. There is only one 3-cell in Fig.2 and it is in M. If one of the three 1-cells shown in Fig.2 by dashed lines would be in M then its end points would be also in M since M is closed. In such a case the 0-cell P would be a convex combination of the end points of two 1-cells and thus it would be not a local corner. To each 1-cell of M there must be an incident 3-cell of M since M is homogeneously three-dimensional. Thus in this case there must be at least two 3-cells incident to P. Inversely, if there are more than one 3-cell of M incident to P, then the 0-cells incident to them are also in M, since M is closed. Then there is at least one pair of 0-cells of M such that P is a convex combination of the cells of the pair.

Thus testing the 6-neighborhood of P is rather simple in the topological raster where both the 0-cells and the 3-cells are represented: it is sufficient to test the eight 3-cells incident to P and to count those which are in M. The procedure testing 26 neighbors of P is more complicated.

We have found another solution which needs no topological raster and uses the same procedure which we have described above as the procedure for testing 26

neighbors of a vector. The procedure uses the possibility to interpret the elements of a standard raster as the 0-cells rather than the 3-cells. It is possible to uniquely assign each 0-cell P (except some 0-cells lying in the border of the space) to a 3-cell, e.g. to the farthest one from the coordinate origin among those incident to P. This idea is illustrated by the arrows in the 2D example of Fig.3a. Thus the given set of voxels may be converted into the set of vertices of these voxels. As the result of the conversion of the given set V some additional elements of the standard raster at the boundary parts of V most remote from the origin become labeled (black squares in the example of Fig.3b). After this conversion the procedure of testing 26 neighbors of each element of the raster may be applied without any change.

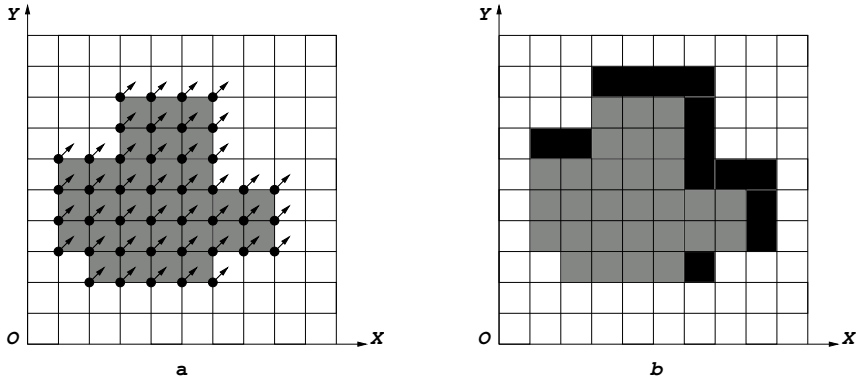

Fig.3: Converting the coordinates of pixels (gray squares) into those of their vertices; the assignment (a) and the additionally labeled elements (black squares in b)

The second part of our algorithm is that of constructing the convex hull of the set L of the local corners found by the first part.

To build the convex hull of L we first create a simple convex polyhedron spanning four arbitrary non-coplanar local corners of L. It is a tetrahedron. It will be extended step by step until it becomes the convex hull of L. We call it the *current polyhedron* CP.

The surface of the current polyhedron is represented with the data structure called the *two-dimensional cell list* [Kov89] which is now generalized to represent complexes in which a 0-cell may be incident to *any number* of 1-cells. In the original version the list was designed for block complexes [Kov89,Kov01] embedded into a Cartesian complex, where a 0-cell is incident to at most four 1-cells.

The cell list of a two-dimensional complex consists in the general case of three sublists. The kth sublist contains all k-dimensional cells (k-cells), $k = 0, 1, 2$. The 0-cells are the vertices, the 1-cells are the edges, the 2-cells are the faces of the polyhedron. Each entry in the kth sublist corresponds to a k-cell c^k. The entry contains indices of all cells incident to c^k. The entry of a 0-cell contains also its coordinates.

The contents of the cell list is illustrated in Tables 1 to 3 for the case of the surface of a tetrahedron.

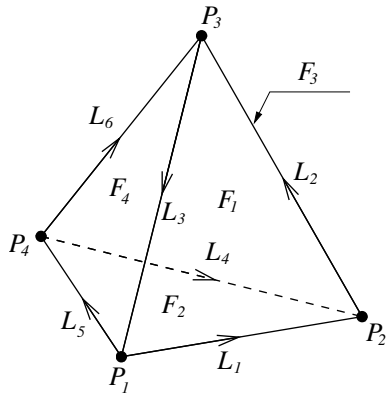

Fig.4: The tetrahedron and its cells

Table 1: Vertices of the tetrahedron

Vertex	Coordinates	Edges
1	(x_1, y_1, z_1)	-1, 3, -5
2	(x_2, y_2, z_2)	1, -2, 4
3	(x_3, y_3, z_3)	2, -3, 6
4	(x_4, y_4, z_4)	-4, 5, -6

Table 2: Edges of the tetrahedron

Edge	StartP	EndP	LeftF	RightF
1	1	2	1	2
2	2	3	1	3
3	3	1	1	4
4	4	2	2	3
5	1	4	2	4
6	4	3	3	4

Table 3: Faces of the tetrahedron

Face	n	Pairs (P, L)
1	3	$(1,1), (2,2), (3,3)$
2	3	$(1,5), (4,4), (2,-1)$
3	3	$(2,-4), (4,6), (3,-2)$
4	3	$(1,-3), (3,-6), (4,-5)$

Let us explain the contents of the cell list. The surface is considered as a two-dimensional non-Cartesian complex consisting of vertices (0-cells), edges (1-cells) and faces (2-cells). The sublist of the vertices (Table 1) contains in the first column the identifiers of the vertices, which are simultaneously their indices in the corresponding array.

The second column contains three integers (x_i, y_i, z_i) for each vertex. These are the coordinates of the vertex. The third column contains in each row the indices of all edges incident to the corresponding vertex. The indices of the edges are signed: the minus sign denotes that the oriented edge points away from the vertex while the plus sign corresponds to an edge pointing to the vertex.

The second sublist (Table 2) is that of edges. Its first column contains the indices. The subsequent columns contain the indices of the starting vertex, of the end vertex, of the left and of the right face, when looking from outside of the polyhedron.

The third sublist (Table 3) is that of faces. The topological relation of a face to other cells is defined by the boundary of the face. The surface of a convex polyhedron

is a combinatorial manifold. Therefore the boundary of each face is a 1-manifold, i.e. a simple closed polygon. It is a closed sequence of pairs each of which contains a vertex and an edge. The sequence of pairs stays in a linked list whose content is shown in the third column. The second column contains the number of pairs, which may be different for different faces.

This version of the cell list is redundant because it contains for a pair of two incident cells c^k and c^m both the reference from c^k to c^m and from c^m to c^k. Therefore, for example, the sublist of edges may be computed starting from the sublist of faces. Also the content of the third column of Table 1 may be computed from that data. The redundancy makes the calculation of the convex hull faster because cells incident to each other may be found immediately, without a search. When the calculation of the convex hull is ready, the redundancy of the cell list can be eliminated to save memory space. To exactly reconstruct a convex object from the cell list of its convex hull it suffices to have the coordinates of the vertices and the sublist of the faces where the indices of the edges may be omitted. This is the economical encoding of the convex hull.

The next step in constructing the convex hull is to extend the current polyhedron while adding more and more local corners, some of which become vertices of the convex hull. When the list of the local corners is exhausted the current polyhedron becomes the convex hull of M. The extension procedure is based on the notion of visibility of faces which is defined as follows.

Definition VI: The face F of a convex polyhedron is *visible* from a point P, if P lies in the outer open half-space bounded by the face F, i.e. if the scalar product (N, W) of the outer normal N of the face F and the vector W pointing from a point Q in F to P, is positive. If the scalar product is negative then F is said to be *invisible* from P. If the scalar product is equal to zero then F is said to be *coplanar* with P.

It should be mentioned that the choice of a point Q in F as the starting point of W does not influence the value of the scalar product, since all vectors lying in F are orthogonal to N and therefore their scalar product with N is zero.

To extend the current polyhedron the algorithm takes one local corner after another. For any local corner P it computes the visibility of the faces of the polyhedron from P. Consider first the simpler case when there are no faces of the current polyhedron, which are coplanar with P. The algorithm labels each face of the current polyhedron as being visible from P or not. If the set of visible faces is empty, then the point P is located inside the polyhedron and may be discarded. If one or more faces are visible, then the polyhedron is extended by the point P and some new faces. Each new face connects P with the boundary of the set of visible faces. A new face is a triangle having P as its vertex and one of the edges of the said boundary as its base. All triangles are included into the cell list of the current polyhedron while all visible faces are removed. Also each edge incident to two visible faces and each vertex incident only to visible faces is removed.

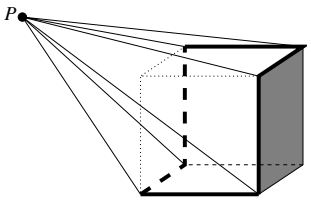

Fig.5: The current polyhedron (a cube) being extended by the vertex P

In Fig.5 the boundary of the visible subset is shown by bold lines (solid or dashed). The edges shown by dotted lines must be removed together with the three faces visible from P. The algorithm repeats this procedure for all local corners.

Consider now the problem of coplanar faces. In principle it is possible to treat a face coplanar with the new local corner P in the same way as either a visible or an invisible face. Both variants have its advantages and drawbacks.

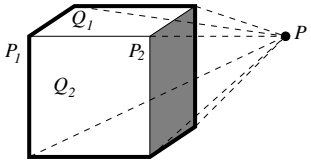

Fig.6: Treating coplanar faces as visible ones

So when considering faces coplanar with a new vertex P as visible ones (Fig.6) then unnecessary many new triangles are constructed some of which are coplanar with each other. They must be merged together later on, and this makes the procedure slower.

When, however, treating a coplanar face as an invisible one (Fig.7a) then the number of new faces being constructed may be made smaller: it is possible to connect P only with the end points P_1 and P_2 of the common boundary of the coplanar face Q_2 and of the visible region, rather than with all points of the common boundary. However, in this case some of the faces are no triangles. Their common boundary may consist of more than one edge. The procedure of merging such faces proved to be much more complicated than that of merging triangles. An example is shown in Fig.7a: the point P is coplanar with the square Q_2 and with the new quadrangle Q_3. Their common boundary consists of two edges.

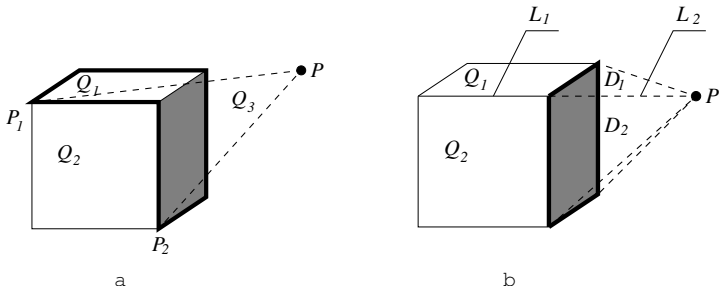

Fig.7: Treating coplanar faces as invisible; coplanar quadrangles Q_2 and Q_3 (a) and collinear edges L_1 and L_2 (b)

When considering coplanar faces as invisible ones some collinear edges may occur (L_1 and L_2 in Fig.7b). For merging them to a single edge some additional means are necessary.

The best solution seams to consist in considering coplanar faces neither as visible nor as invisible ones. In this case it is possible to extend a coplanar face towards the point P by a procedure looking like the construction of a two-dimensional convex hull (Fig.8).

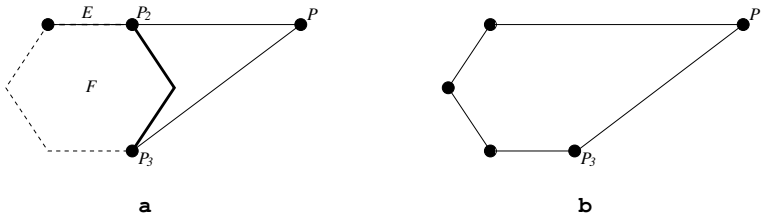

Fig.8: Constructing a 2D convex hull in the case of a coplanar face: the initial (a) and the resulting (b) face

The number of the new faces being constructed is small. However, the procedure of extending a face towards a coplanar point proved to be still more complicated than that of merging two polygons with more than one common edge. Also some collinear edges may occur in this case (the edges E and (P_2, P) in Fig.8a). The extension becomes especially complicated in the case when the point P lies on the crossing line of the planes of two adjacent faces (Fig.6b). Then both of these faces are coplanar with P.

After having tested all three variants we came to the decision that the best solution consists in treating coplanar faces as visible ones. In this case the program creates sometimes many coplanar triangles which must be merged together. But the procedure of merging triangles is rather simple and fast.

It should be noticed here that merging triangles is not always necessary: many programs of computer graphics work with sets of triangles. Our algorithm gives the possibility to represent the surface of the convex polyhedron as a simplicial complex whose all faces are triangles. To achieve this it suffices to switch off the subroutine of merging. For example, the convex hull of a digital ball of diameter 32 contains 1404 triangles. After merging coplanar triangles the hull contains only 266 faces.

The procedure of adding new faces to the current polyhedron ends after having processed all local corners.

3 Proof of the Correctness

The described algorithm of constructing the convex hull gets a set V of voxels as input and generates an AG-polyhedron K defined by the cell list of its surface. In this section we prove the correctness of the algorithm (Theorem PL). We need the following lemmas LZ, CH and BE.

Lemma LZ: Let V be a set of three-dimensional vectors $\boldsymbol{v} = (v_x, v_y, v_z)$ (pointing to voxels or to vertices of voxels). Let T be a subset of V such that all vectors of T satisfy some given linear inequality, i.e. the inequality cuts T from V. Then T contains at least one vector which is no convex combination of vectors of V.

Proof. Let $H(x, y, z) \geq 0$ be the said inequality. Let us find all vectors of T having $H(x, y, z) = max$. We denote by $TH \subset T$ the subset of all vectors $\boldsymbol{c} = (c_x, c_y, c_z)$ having the maximum value of H:

$$\boldsymbol{c} \in TH \longrightarrow H(c_x, c_y, c_z) \geq H(v_x, v_y, v_z) \text{ for each vector } \boldsymbol{v} \in V. \tag{2}$$

The vector $\boldsymbol{c}$ satisfying (2) can only be a convex combination of vectors from TH since H is linear and its value for a convex combination

$$\boldsymbol{c} = \alpha \cdot \boldsymbol{v}_1 + (1 - \alpha) \cdot \boldsymbol{v}_2; \text{ with } 0 < \alpha < 1; \ \boldsymbol{v}_1, \boldsymbol{v}_2 \in V. \tag{3}$$

with other vectors would be less than the maximum value of H. If the subset TH contains a single vector $\boldsymbol{c}$ then we are done: $\boldsymbol{c}$ is no convex combination of vectors

of V. If, however, TH contains more than one vector, then consider the subset $TX \subset TH$ of vectors having the maximum value of the coordinate X. If there is a single such vector then we are done. If not, then all vectors of TX have the same coordinate $x = x_{max}$ and we consider the subset $TY \subset TX$ of vectors having the maximum value of the coordinate Y. This process ends at the latest with the subset $TZ \subset TY$ of vectors having the maximum value of the coordinate Z since there exits only one vector with $x = x_{max}$, $y = y_{max}$, $z = z_{max}$. This vector is no convex combination of vectors of V. □

Lemma CH: Let V be a set of three-dimensional vectors and K an AG-polyhedron satisfying the following three conditions:

1. K is convex;
2. K contains all local corners of V;
3. each vertex of K is identical with one of the local corners of V.

Then K is the convex hull of V according to the Definition CH given in the Introduction.

Proof. First of all let us show that K contains all vectors of V. Suppose there is a subset $T \subset V$ which is not contained in K. This means that the vectors of T do not satisfy at least one of the linear inequalities corresponding to the faces of K. According to Lemma LZ the subset T contains at least one vector which is not a convex combination of vectors of V and thus it is a local corner. This contradicts the second condition of the Lemma. Therefore there is no subset $T \subset V$ outside of K, i.e. K contains all vectors of V.

The polyhedron K is convex and contains all vectors of the set V. To fulfill all conditions of the Definition CH it remains to show that K is the smallest such polyhedron. With other words, we must show that each convex polyhedron K' different from K which is contained in K does not fulfill the conditions of Lemma CH.

If the convex polyhedron K' is contained in K and is different from K then at least one vertex of K' lies in the interior of K. Let us construct the polyhedron K' at first by moving a vertex P of K by a small amount into its interior. Then the original vertex P does not belong to K'. According to the conditions of Lemma CH the vertex P is a local corner of V. Thus there is a local corner of V which is not in K'. Therefore K' does not meet the conditions of Lemma CH.

Each other transformation of K to some polyhedron lying inside of K may be represented as a sequence of movements of vertices of K into its interior. It follows that such a polyhedron does not contain some vertices of K and hence is not in accordance with the conditions of Lemma CH. □

Lemma BE: If an edge E of a polyhedron K lies between a face F_v visible from a point P and another face F_n which is not visible from P then the polyhedron K lies completely in the closed half-space bounded by the plane (P, E) going through P and E.

This means that the coordinates of all vertices of K satisfy the linear inequality $H(x, y, z) \leq 0$ while the linear form $H(x, y, z)$ takes the value zero at the point P and at the endpoints of E. The coefficients of $H(x, y, z)$ are the coordinates of the normal of the plane (P, E), which points to the outside of K.

Proof. If the face F_v is visible from P then the outer normal N_v of F_v and the line segment (Q, P) compose an acute angle $\alpha_v < 90°$ while Q is an arbitrary point in E (Fig.9). Conversely, the outer normal N_n of the invisible face F_n and the line segment (Q, P) compose an obtuse angle $\alpha_n > 90°$. The angle β_v between the outer

normal N_H to the plane (P, E) and N_v is equal to $90° - \alpha_v$ and is acute. The angle β_n between N_H and N_n is equal to $\beta_n = \alpha_n - 90°$ and is also acute. Consequently the normal N_H to the plane (P, E) is a linear combination of the normals N_v and N_n with positive coefficients (proportional to $cos\beta_v$ and to $cos\beta_n$):

$$N_H = a \cdot N_v + b \cdot N_n;\ a, b > 0 \tag{4}$$

The vector W pointing from Q to an arbitrary point in the polyhedron K has a non-positive scalar product with both N_v and with N_n because K is convex and all its inner point lie on the inner side of each of its faces. According to (4) the scalar product of N_H with the vector W for an arbitrary point of K is non-positive. Thus all points of K lie in the closed half-space bounded by the plane (P, E) which proves the Lemma. □

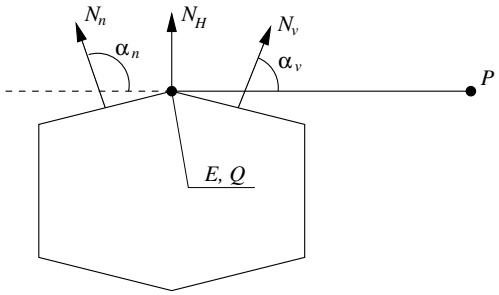

Fig.9: A section through the polyhedron K and the plane (Q, P)

An angle between two vectors whose end points are cells of a 3D Cartesian AC complex is specified by the scalar and vector product of these vectors.

Theorem PL: The algorithm described in the previous Section constructs the convex hull of the given set of voxels or of vertices of voxels.

Proof. We consider the coordinates of voxels or of vertices of voxels as three-dimensional vectors composing the set V. At its first stage the algorithm saves the local corners of V. Only these vectors are used as candidates for the vertices of the polyhedron being constructed. Thus the third condition of Lemma CH is fulfilled. To fulfill the remaining two conditions it remains to show that the constructed polyhedron is convex and that it contains all local corners of V.

The algorithm starts with a polyhedron K_i, $i = 0$; which is a tetrahedron. It is convex. Then the algorithm takes the next not used local corner P and defines the set of faces which are visible from P (Definition VI in the previous Section). The boundary of the set of visible faces consists of vertices and edges while each edge bounds exactly one visible face. The algorithm constructs a new face for each edge of the boundary. The face spans the edge and the point P. The visible faces become deleted. According to Lemma BE the polyhedron K_i lies completely in the closed half-space defined by the plane of the new face F. It also lies in the closed half-space corresponding to the inner side of each old not deleted face. Thus the new polyhedron is an intersection of half-spaces and thus it is convex. This is true for any new added local corner P.

Let us show now that K contains all local corners of V. The algorithm processes all local corners of V. Some of them turn out to lie in the current polyhedron. They are not used for the construction of K, but they are nevertheless already contained in K. Other local corners are put into K as their vertices and thus they also become contained in K. All local corners being in K remain in K since the modified polyhedron contains the old one. Thus all local corners are contained in K, all three conditions of Lemma CH are fulfilled and therefore K is the convex hull of V. □

4 Results of Computer Experiments

We have implemented and tested the described algorithm of constructing the convex hull of a set of voxels or of vertices of voxels. To make a numerical comparison of the memory efficiency we have tested some examples with our algorithm and with the well known Marching Cubes algorithm [Lor87]. It should be mentioned here that we only compare the encoding efficiency of the surface, because the Marching Cubes algotithm does not produce the convex hull of an object.

We have compared the number of integer values necessary to save a non-redundant cell list with that necessary to save the triangulation of the same object. For the Marching Cubes triangulation method (MC-triangulation) we have assumed the following: one needs to save three coordinates for each vertex and three vertex indices for each triangle.

Since the number N_T of triangles is nearly twice the number N_V of the vertices, we must save on the average

$$size_{tr} = 3 \cdot N_V + 3 \cdot N_T = (3/2 + 3) \cdot N_T \tag{5}$$

integers, i.e. about 4.5 integers for each triangle.

As mentioned above in the Section "The Algorithm" the non-redundant cell list of the convex hull contains the coordinates of the vertices and the sublist of the faces where the indices of the edges are omitted. The later list contains for each face the sequence of the vertex indices in its boundary. In our experiments each face have had on the average about 5 vertices in its boundary. Thus the memory amount necessary to store the non-redundant cell list is approximately equal to

$$size_{cl} = 3 \cdot N_V + 5 \cdot N_F; \tag{6}$$

where N_V and N_F are the numbers of vertices and faces correspondingly.

A simple example is shown in Fig.10: the object has 3571 voxels and 1782 faces. The convex hull contains only 63 faces.

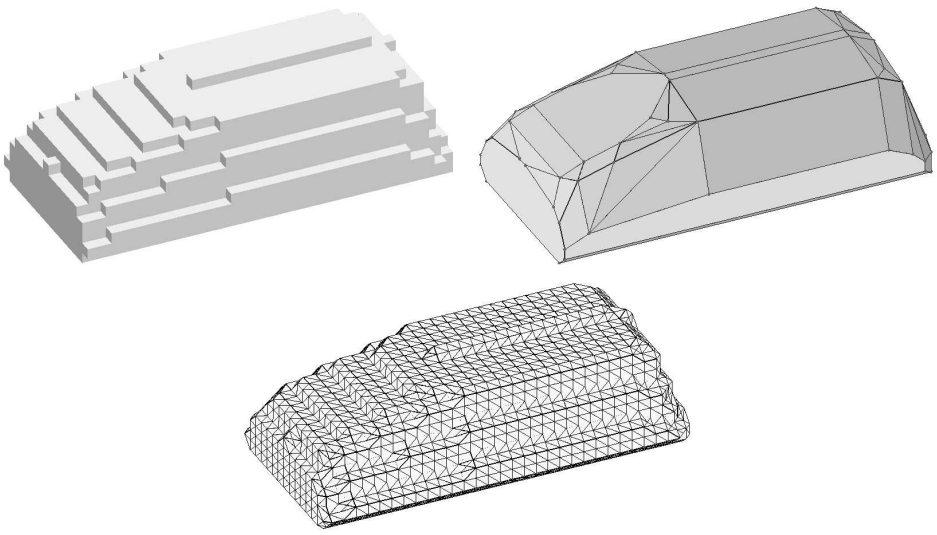

Fig.10: Example "future car". Top left: The voxel object. It has 3571 voxels and 1782 faces. Top right: Convex hull of this object. Bottom: Triangulation with the Marching Cubes method

The following values for the memory requirements have been computed:

Table 4: Comparison of the memory requirements of the cell list and of the triangulation for the example in Fig.10

	faces	vertices	integers to save
MC-triangulation	3560	∼1780	16020
convex hull	63	60	495

The same investigation was made for half-balls with diameters from 8 to 22 voxels (Fig.12).

Table 5: Experimentally acquired values for half-balls of various diameters

	MC-triangulation			convex hull		
diameter	triangles	integers to save	integers per triangle	faces	inegers in cell list	integers per face
8	504	2268	4.5	30	324	10.80
10	760	3420	4.5	62	454	7.32
12	1072	4824	4.5	50	442	8.84
14	1416	6372	4.5	30	426	14.20
16	1840	8280	4.5	118	1010	8.56
18	2296	10332	4.5	142	1070	7.53
20	2784	12528	4.5	94	938	9.98
22	3488	15696	4.5	118	1142	9.68

The following diagram shows the number of integers necessary to save either the results of the triangulation or the cell list.

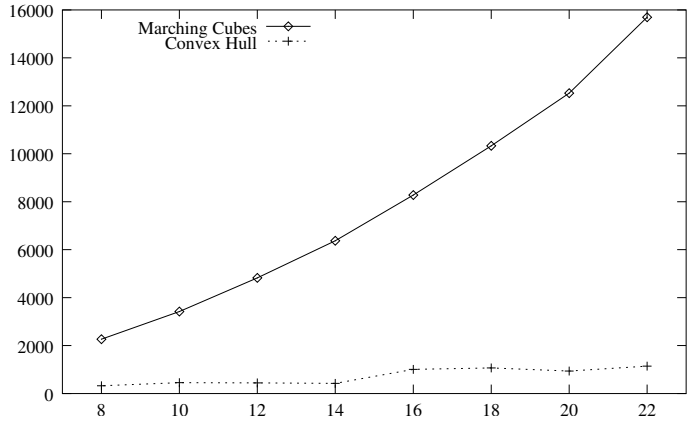

Fig.11: Number of integers to be saved for half-balls with diameters from 8 to 22 voxels

As it can be seen from Fig.12, the convex hull preserves the symmetry of digital objects. The polygons composing the surface are all symmetric. This is an important property of the convex hull, besides its property to be economic, and it is its great advantage as compared with other means representing surfaces of digital objects, e.g. the triangulation or the subdivision into digital plane segments [Kle01].

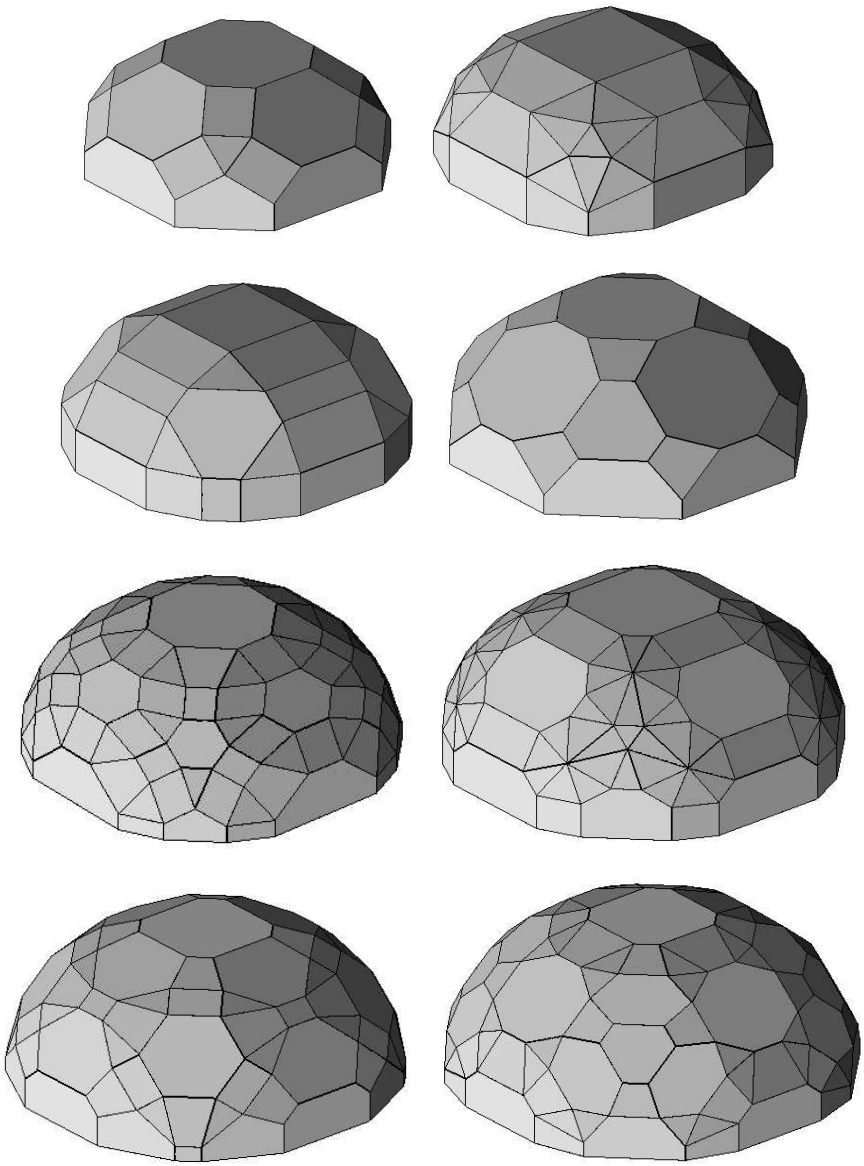

Fig.12: Convex hulls of the half-ball examples with diameters from 8 to 22 voxels

5 Conclusion

In this paper we present a new algorithm of computing the convex hull of a three-dimensional digitized object represented as a set of voxels. The computed convex hull is an abstract polyhedron which is a particular case of an abstract cell complex. The surface of the polyhedron is encoded by the data structure known as the two-dimensional cell list. This data structure is well suited both to handle the intermediate data during the computation of the convex hull as well as to economically encode the resulting hull. The cell list also provides the possibility to exactly reconstruct the original digitized object. The correctness of the presented algorithm has been proved. Numerous computer experiments demonstrate the memory efficiency of the cell list for convex objects as compared with the well known triangulation method.

References

[Ber00] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: *Computational Geometry - Algorithms and Applications*. Springer-Verlag. 2000.

[Cla92] Clarkson, K.L., Mehlhorn, K., Seidel, R.: *Four results on randomized incremental constructions*. Comp. Geom.: Theory and Applications, pages 185-221, 1993. Preliminary version in Proc. Symp. Theor. Aspects of Comp. Sci., 1992.

[Kle01] Klette, R., Sun, H.J.: *A Global Surface Area Estimation Algorithm for Digital Regular Solids*. University of Auckland, CITR-TR-69. 2001.

[Kov89] Kovalevsky, V. A.: *Finite Topology as Applied to Image Analysis*. Computer Vision, Graphics and Image Processing, Vol.45, No.2, pp.141-161. 1989.

[Kov93] Kovalevsky, V. A.: *Digital Geometry based on the Topology of Abstract Cell Complexes*. In Proceedings of the Third International Colloquium "Discrete Geometry for Computer Imagery". University of Strasbourg. pp.259-284. 1993.

[Kov01] Kovalevsky, V. A.: *Algorithms and Data Structures for Computer Topology*. In: Bertrand, G., Imiya, A., Klette, R. (Eds): Digital and Image Geometry. Lecture Notes in Computer Science, Vol.2243, pp.37-58. Springer-Verlag. 2001.

[Kov02] Kovalevsky, V.A.: *Multidimensional Cell Lists for Investigating 3-Manifolds*. Discrete Applied Mathematics, Vol. 125, Issue 1, pp.25-43. 2002.

[Lor87] Lorensen, W.E., Cline, H.E.: *Marching Cubes: A High-Resolution 3D Surface Construction Algorithm*. Computer Graphics, Vol. 21, No. 4, pp.163-169. 1987.

[Pre85] Preparata, F.P., Shamos, M.I.: *Computational Geometry - An Introduction*. Springer-Verlag. 1985.

A Abstract Cell Complexes

In this section we want to remind the reader the basic definitions of the theory of abstract cell complexes [Kov89,Kov93].

Definition ACC: An *abstract cell complex* (AC complex) $C = (E, B, dim)$ is a set E of abstract elements called *cells* provided with an antisymmetric, irreflexive, and transitive binary relation $B \subset E \times E$ called the *bounding relation*, and with a *dimension function* $dim : E \longrightarrow I$ from E into the set I of non-negative integers such that $dim(e') < dim(e'')$ for all pairs $(e', e'') \in B$.

The bounding relation B is a partial order in E. The bounding relation is denoted by $e' < e''$ which means that the cell e' bounds the cell e''.

If a cell e' bounds another cell e'' then e' is called a *side* of e''. The sides of an abstract cell e'' are not parts of e''. The intersection of two distinct abstract cells is always empty, which is different from Euclidean complexes.

If the dimension $dim(e')$ of a cell e' is equal to d then e' is called *d-dimensional* cell or a *d-cell*. An AC complex is called *k-dimensional* or a *k-complex* if the dimensions of all its cells are less or equal to k. Cells of the highest dimension k in an k-complex are called *ground cells*.

Definition SC: A *subcomplex* $S = (E', B', dim')$ of a given AC complex $C = (E, B, dim)$ is an AC complex whose set E' is a subset of E and the relation B' is an intersection of B with $E' \times E'$. The dimension dim' is equal to dim for all cells of E'. A subcomplex of a given complex is uniquely defined by the subset E'. Therefore it is usual to say "subset" instead of "subcomplex".

Definition OP: A subset OS of cells of a subcomplex S of an AC complex C is called *open in* S if OS contains each cell of S which is bounded by a cell of OS.

Definition SON: The smallest subset of a set S which contains a given cell $c \in S$

and is open in S is called *smallest neighborhood* of c relative to S and is denoted by $SON(c, S)$.

Definition CS: A subset CS of cells of an AC complex C is called *closed* if CS contains all cells of C bounding cells of CS.

Definition CL: The smallest subset of a set S which contains a given subset $M \subset S$ and is closed in S is called the *closure* of M relative to S and is denoted by $Cl(M, S)$.

Definition IN: The greatest subset of a set S which is contained in a given subset $M \subset S$ and is open in S is called the *interior* of M relative to S and is denoted by $Int(M, S)$.

Definition BD: The *boundary* ∂S of an n-dimensional subcomplex S of an n-dimensional AC complex C is the closure of the set of all (n-1)-cells of C each of which bounds exactly one n-cell of S.

Definition MA: An *n-dimensional combinatorial manifold* (n-manifold) without boundary is an n-dimensional complex M in which the $SON(P, M)$ of each 0-cell P is homeomorphic to an open n-ball with the cell P lying in the interior of $SON(P, M)$. In a manifold with boundary the $SON(P, M)$ of some 0-cell P may be homeomorphic to a "half-ball", i.e. the 0-cell P lies in the boundary of $SON(P, M)$ rather than in its interior.

Definition IC: Two cells e' and e'' of an AC complex C are called *incident* to each other in C iff either $(e', e'') \in B$, or $(e'', e') \in B$, or $e' = e''$. The incidence relation is symmetric, reflexive and non-transitive.

Definition CN: Two cells e' and e'' of an AC complex C are called *connected* to each other in C iff either e' is incident to e'' or there exists a cell $c \in C$ which is connected to both e' and e''. Because of this recursive definition the connectedness relation is the transitive hull of the incidence relation.

Definition HN: An n-dimensional AC complex C is called *homogeneously n-dimensional* if every k-dimensional cell of C with $k < n$ is incident to at least one n-cell of C.

Definition RG: A *region* is an open connected subset of the space.

Definition SO: A region R of an n-dimensional AC complex C is called *solid* if every cell $c \in C$ which is not in R is incident to an n-cell of the complement $C - R$.

Definition DHS: A *digital half-space* is a solid region of a three-dimensional Cartesian AC complex containing all voxels whose coordinates satisfy a linear inequality.

Definition TL: A connected one-dimensional complex in which all cells, except two of them, are incident to exactly two other cells is called a *topological line*.

Definition CA: By assigning subsequent integer numbers to the cells of a topological line L in such a way that a cell with the number x is incident to cells having the numbers $x-1$ and $x+1$, one can define *coordinates* in L which is a one-dimensional space. AC complexes of greater dimensions may be defined as Cartesian products of such one-dimensional AC complexes. A product AC complex is called a *Cartesian*

complex.

Definition TR: An n-dimensional array whose each element is assigned to a cell of an n-dimensional Cartesian AC complex while the topological coordinates of the cell serve as the index of the corresponding element of the array is called the *topological raster.*

In a topological raster it is possible to access each cell of any dimension and save a label of any cell. By means of topological coordinates it is easy to find all cells incident to a given cell without a search. It is also possible to specify the dimension of a cell by means of its topological coordinates.

Definition SG: A *standard raster* is an n-dimensional array whose elements represent only the ground cells (i.e. the n-dimensional cells) of an n-dimensional complex, e.g. only the pixels in the 2D case or only the voxels in the 3D case.

A cell c of some lower dimension gets in the standard raster the same coordinates as the ground cell incident to c and lying farther away from the origin of the coordinate system. The dimension of c cannot be specified by means of its coordinates, it must be specified explicitly. To save a label of a cell of some lower dimension (if necessary) some special means are necessary, e.g. it is possible to assign different bits of a byte in the standard raster to cells of different dimension having all the same coordinates.