Creating triangle strips from clustered point sets

Peter Jenke
September 2010

Bachelor Thesis, 15 hp, C
Computer Science

Computer Science program
Examiner: Anders Hast
Co-Examiner: Stefan Seipel
Supervisor: Anders Hast
Creating triangle strips from clustered point sets

Peter Jenke

Akademin för teknik och miljö
Högskolan i Gävle
S-801 76 Gävle

E-mail: nfk07pje@student.hig.se

Abstract

To create a digital model of the surface of some object from a set of points, representing positions on the surface of this object, requires information about the relationship between the points. This information is not immediately accessible. Thus, for creating such a model it is necessary to establish relationships between the points of the set.

In addition, it should be possible to render the resulting model as efficiently as possible. Modern graphics cards offer to send vertex informations as triangle strips; by using triangle strips the information about the triangles can be compressed.

This work is about a method for retrieving information about the relations between points in an unstructured spatial point set and transforming this information into triangle strips. It is based on the convex layers of a planar point set and an algorithm for triangulating the annuli of the convex layers, which uses the Rotating Calipers.

Keywords: Surface reconstruction, triangle strip, convex layers, Rotating Calipers
Acknowledgments

I would like to thank my supervisor Doc. Anders Hast, who invited me to this work. I am very grateful that he introduced me into the wonderful world of computational geometry.

I would also like to thank my teachers at the University College of Gävle who helped me to finish the studies, especially Fredrik Bökman and Prof. Stefan Seipel.

My family stood beside me during the last three years. Thank you for your love and your patience!
1 Introduction

To create a digital model of the surface of some object from a set of points, which represents positions on the surface of this object requires information about the relationship between the points. This information is not immediately accessible, if only the point positions are known. Thus, the first task this work has to deal with is to establish relationships between the points of a point set.

On the other hand, a digital model should be rendered as efficiently as possible to make it as useful as possible for the user. Modern graphic cards offer to send vertex information in a special order to save time for communication. This order is called triangle strip; by using triangle strips the information about the triangles can be compressed, because the information of the most vertices sent to the GPU is used in three triangles. Ideally, the amount of data can be little more than 1/3 of the number of vertices needed for rendering single triangles.

Several methods for constructing triangle strips from a point cloud are known: Some of them operate directly on the point set; others work indirectly, by first computing a triangulation of the point set (usually a mesh) and then applying a stripification algorithm on the result.

A method which creates triangle strips from a mesh is presented by Gopi and Eppstein [9]: This algorithm searches for a perfect matching in the dual graph of the given triangulation. By the matching, every triangle in the mesh is paired with exactly one of the adjacent triangles. From these pairs, a cycle of connected triangles through the mesh is constructed; the result is a triangle strip.

Boubekeur et al. propose a method called surfel strips [4], which creates a number of small triangle strips. This method processes the point set directly. It partitioned it into a number of subsets with the help of an Octree based datastructure. For a subset, an average plane is determined and the points in the set are projected onto this plane. The triangulation is performed by Delauney triangulation, followed by stripification.

Arkin presents a method, which works directly on the point set [2]. It is based on recursive subdivisions of the set: In a first step, the convex hull of the point set is computed. After this, one point in the interior is chosen and edges from this point to the points in the convex hull are created. The edges are subdividing the interior of the hull. If a created triangle is not empty, one of the points inside is chosen, and edges to the vertices of the surrounding triangle are created. This process is repeated until all triangles are empty.
A wellknown property of a point set is its convex hull. If the points of a convex hull are removed from the point set, the hull of the remaining subset can be constructed. By repeating this "onion peeling", a structure called convex layers is computed.

This work investigates an algorithm for creating triangle strips from a point cloud. It is based on the triangulation of convex layers and comprises the following steps:

- Sorting the points.
- Transforming the points into 2D space by projection on a fitting plane.
- Computing a set of convex layers from the planar point set.
- Triangulating the space between the different layers ("annuli").
- At the end: creating triangle strips from this triangulation.

The remaining part of the work is organized as follows: Section 2 is about the theoretical background of the used algorithms. In section 3, the implementation of the algorithms is shortly presented and the achieved results in section 4. The work is completed by the discussion of the results in section 5.
2 Theory

2.1 Prerequisites

The faces of a planar graph $G = (V,E)$ are the areas completely enclosed by edges plus the area which encloses $\mathbb{R}^2 \setminus G$, i.e. the area outside $G$. We define a vertex $f$ inside every face of the graph $G$ and connect the $f_i$ by a set of vertices $B$, such that for every edge $e \in E$ (which separates two vertices $f_a$ and $f_b$) a new vertex $b$ (connecting $f_a$ and $f_b$) is inserted into $B$. If an edge is related to only one face, a loop is inserted. The resulting graph $D = (F,B)$ is called the dual of the graph $G$. [8](74, 91)

Given a graph $G$. A path through $G$ which visits each vertex exactly once is called a Hamilton path. [8](211)

A point set $P$ is the union of a number of distinct points (i.e. with distinct positions) $p_i$ ($i = 1 \ldots n$) in a $k$-dimensional space. In this work, we concentrate on spaces with two and three dimensions.

A point set $P$ can be triangulated. The result is a triangulation $T$, in 2D space a planar multigraph $T = (P,E)$ with the points of $P$ as vertices and a set of edges $E$ between the vertices.

A triangle strip is a sequence of vertices $v$ such that the vertices $v_i$, $v_{i+1}$, $v_{i+2}$ define one triangle $T_k$ and the vertices $v_{i+1}$, $v_{i+2}$, $v_{i+3}$ the successor $T_{k+1}$. This means, that the vertices of subsequent triangles are ordered in their triangles alternating in clockwise and counterclockwise order.

![Figure 1: Triangle strip](image)

Both triangles are sharing the edge $(v_{i+1}, v_{i+2})$; $v_{i+1}$ and $v_{i+2}$ are vertices of triangles $T_1$ and $T_2$. If $T_2$ has a successor, $v_{i+2}$ is vertex in three triangles; if a number of triangles are sharing edges in the same manner, all the points are vertices in three triangles with the exception of the first and the last vertex [1](549). That means, a triangle strip consists of $n + 2$ points, if it contains $n$ triangles. On the other hand, for representing $n$ isolated triangles, $3n$ points would be necessary.
A triangle strip can be represented by a list of vertices. Consider as example the triangles in figure 1: The list representing the strip contains the vertices in the order \( v_0, v_1, v_2, v_3, v_4, v_5, v_7, v_8, v_9 \).

In a generalized triangle strip, a triangle can have a neighbor with the same order of vertices (CW or CCW). Consider the triangles in figure 2: Here, \( T_4 \) and \( T_5 \) are neighbors with the same orientation. A strictly ordered list for this strip is \( v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9 \). The problem here is, that a nonexisting triangle \( v_6, v_7, v_8 \) is created and the triangle \( v_5, v_7, v_8 \) disappears. To solve this, a swap vertex is inserted in the list: With a swap vertex, an empty triangle is introduced. That means, the geometry doesn’t change, but the order of the represented triangles is corrected [1](551).

In the example, \( v_5 \) has to be sent two times - after \( v_4 \) and after \( v_6 \), and the order in the list is \( v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_5, v_7, v_8, v_9 \).

![Figure 2: Generalized triangle strip](image)

For a finite point set, the convex hull can be computed. Informally, the convex hull can be thought as a rubberband spanned around a set of nails, sitting in a plane. Several definitions of convex hulls were formulated. Because in this work convex hulls are only used in relation to 2D point sets, the following one, given by O’Rourke seems useful:

"The convex hull of a set of points \( S \) in the plane is the smallest convex polygon \( P \) that encloses \( S \), smallest in the sense that there is no other polygon \( P' \) such that \( P \supseteq P' \supseteq S \)." [10](124)

Let \( P \) be a set of points in the 2D-space and \( H_0 \) the convex hull of \( P \).

It is possible to compute the set difference of \( P \) and \( H_0 \): \( P_1 = P \setminus H_0 \). Then, the convex hull of \( P_1 \) can be computed; the result is the convex polygon \( H_1 \), which is strictly inside \( H_0 \). This can be repeated until \( P_i = P_{i-1} \setminus H_{i-1} \) is empty. The resulting convex hulls are called the convex layers of \( P \) and denoted as \( C(P) \) [6].

Two convex hulls \( A \) and \( B \), consecutive in \( C(P) \), and the region between \( A \) and \( B \) are denoted as annulus.

---

\(^1\)This process is in the literature even known as 'onion peeling'.

10
2.2 Creating triangle strips from a planar point set

The input data used in this work are points in 3D space, possibly representing points on a surface of an object. These points are sorted in a preprocessing step. Then, a fitting plane is computed by a Principal Component Analysis for every subset and the points, belonging to this set, are projected onto this plane. In this way, a 2D representation for a set of points in 3d space is created.

To create triangle strips from the resulting planar point sets is the core of this work and the methods used to achieve this are the content of this section.

Let \( P \) be a planar point set. The following steps are necessary to create triangle strips from \( P \):

1. Compute the convex layers \( C(P) \).
2. Triangulate the annuli of \( C(P) \); result is a triangulation \( T_i \) for every annulus.
3. Build triangle strips by using the triangulations \( T_i \). Each strip is represented by a sequence of points as explained in section 2.1.

2.2.1 Step 1: Compute the convex layers

The algorithm for computing the convex layers of a point set \( P \) consists of several parts. First, the complete algorithm is presented and after this, its parts.

\[
\text{Algorithm ConvexLayers:} \\
\text{Takes a planar point set } P \text{ as input data.} \\
\begin{itemize}
\item Set } P' = P.
\item Do while } P' \neq \emptyset:
\begin{itemize}
\item Compute the convex hull } H(P').
\item Calculate } P' = P' \setminus H(P')
\end{itemize}
\end{itemize}
\]

Because the functions for computing the convex hull of a point set included in CGAL [5] don’t offer the required functionality (see subsection 3), an own function had to be implemented.
A number of algorithms with different time complexity are available. Some examples are QuickHull with a time complexity of $O(n^2)$  [10](77 ff.) or a Divide-and-Conquer algorithm with $O(n \log n)$  [10](103 ff.)

These algorithms also differ in "implementation complexity". Because Graham’s Scan is simple to implement and has a time complexity of $O(n \log n)$, it is used in this work [7](949), [10](80 ff.). The original algorithm was slightly modified: The points in the computed hull are ordered clockwise.

---

**Algorithm ConvexHull:**

Takes a planar point set $P$ as input data. Uses a stack as datastructure.

- Determine the leftmost lowest point $p_0 \in P$.
- Sort the other points of $P$ around $p_0$ in clockwise order$^a$.
- Push $p_0$ on the stack
- Push $p_1$ on the stack
- Push $p_2$ on the stack
- Do for $i = 3$ to $n$:
  - Do while the angle formed by the points next to top on the stack, top on the stack and $p_i$ makes a left turn:
    - Pop the topmost point from the stack.
  - Push $p_i$ on the stack

$^a$The implementation of the algorithm uses QuickSort.

---

An example may illustrate, how the algorithm works. Consider figure 3:

The content of the stack in the beginning of this step is 11, 9, 14, 8 and the next point in the sorted order is 5$^2$. Since the angle defined by points 14, 8 and 5 makes a right turn, point 5 is saved on the stack and the content of the stack is 11, 9, 14, 8, 5.

The next point after 5 is 7 – see figure 4. Now, 5 is removed from the stack, because points 8, 5 and 7 define a left turn (stack: 11, 9, 14, 8).

Instead, point 7 is saved on the stack and is now the first point on the stack (figure 5); the edge between points 8 and 7 defines a part of the convex hull of the point set (stack: 11, 9, 14, 8, 7).

$^2$If two points and $p_0$ are collinear, the sorting algorithm defines the point with the lowest distance as first in the order.
Figure 3: Computing the convex hull, step 1

Figure 4: Computing the convex hull, step 2

Figure 5: Computing the convex hull, step 3
The difference of two sets $C = A \setminus B$ is defined as

$$C = \{c : c \in A \land c \notin B\}$$

The method used in this work follows this definition.

Algorithm `SetDifference`:

- For all elements $a$ in $A$:
  - For all elements $b$ in $B$:
    - If $a = b$: Break and go to the next element in $A$
    - Add $a$ to $C$

### 2.2.2 Step 2: Triangulate the annuli of the convex layers

After computing the convex layers of $S$, it remains to triangulate the annuli. In the triangulation algorithm, a method called ”Rotating Calipers” [13] is used:

Given a convex polygon $P$. Two parallel lines of support $l_1, l_2$ through two points $p_1, p_2$ of $P$ are rotated until both touch another point: $p_1$ touches $p'_1$ and $p_2$ touches $p'_2$ (figure 6). Then the angles by which the lines were rotated are compared: The smaller one determines the new direction of the calipers. The lines are moved to the next point pair and the process is repeated until $p_1, p_2$ are reached again.

![Figure 6: Rotating Calipers](image)

Toussaint shows that a number of problems can be solved with the Rotating Calipers; finding the minimum-area rectangle enclosing a polygon is one of them, another is merging polygons in a convex hull finding algorithm.
He even proposed the algorithm, used in this work [14].

An annulus of a set of convex layers has an outer hull $A$ and an inner hull $B$. The algorithm creates a sequence of points, describing the edges of the triangulation where one point is member of $A$ and the other belongs to $B$. This set of edges is denoted as $R$.

As a first step, the leftmost points in $A$ and in $B - a_1$ and $b_1$ – are found. Two vertical lines of support are defined: $l_1$ goes through $a_1$, $l_2$ through $b_1$.

Three angles are together the criterion, to determine between which points the next edge of the triangulation is placed (figure 7):

- $\alpha$ - enclosed by the line $a_1 - a_2$ and $l_1$
- $\beta$ - enclosed by the line $b_1 - a_2$ and $l_2$
- $\gamma$ - enclosed by the line $b_1 - b_2$ and $l_2$

![Figure 7: Triangulating annuli: Startconfiguration](image)

The next edge of the triangulation is chosen depending on the angles $\alpha$, $\beta$ and $\gamma$:

If $\beta \leq 0$, the point which can be ’hit’ from $a_1$ is $a_2$ and from $b_1 - b_2$. The new edge depends on the first touched point: If $\alpha < \gamma$, $l_1$ hits $a_2$ before $l_2$ hits $b_2$ and the next edge is $a_2 - b_1$ (figure 8). In this case, the rotation center in $A$ is moved to $a_2$, i.e. $l_1$ is in the next step rotated around $a_2$. The line $l_2$ is still rotated around $b_1$. The slope of both lines is now the same as the slope of line $a_1 - a_2$.

\[\text{Cited by Pirzadeh [11](53ff)}\]
Figure 8: Triangulating annuli: $\beta \leq 0$ and $\alpha < \gamma$

If $\alpha > \gamma$, $b_2$ is hit first by $l_1$. Now, the triangulation takes the opposite direction – the new edge is $a_1 - b_2$ and the center of rotation for $l_2$ moves from $b_1$ to $b_2$ (figure 9).

Figure 9: Triangulating annuli: Second case, if $\beta \leq 0$ and $\alpha > \gamma$

In the third case is $\alpha = \gamma$; both $a_2$ and $b_2$ are hit at the same time by $l_1$ and $l_2$, respectively. Two new edges are added ($a_2 - b_2$ and $b_1 - a_2$) and the center of rotation is moved for both lines.

If, on the other side, $0 < \beta$, the triangulation depends on the angles $\beta$ and $\gamma$. The line $l_2$ can in this constellation reach $a_2$ and $b_2$, and because $l_1$ is left from $l_2$, it cannot hit any point at all.

As first case, consider $\beta < \gamma$ (figure 10): The first point, which $l_2$ reaches, is $a_2$. Here, the rotation is stopped and the new edge $a_2 - b_1$ is added to the triangulation. The rotation center in $A$ is moved to $a_2$.

If $\beta > \gamma$, the first reached point is $b_2$. The new edge $a_1 - b_2$ is added and the rotation center in $B$ is moved to $b_2$ (figure 11).

The last possibility here is $\beta = \gamma$; $l_2$ hits $a_2$ and $b_2$ at the same time. As new edges are $a_1 - b_2$ and $a_2 - b_2$ added and the center of rotation is moved for both lines.

This process is repeated until both startpoints are reached again. In figure 12, a summary of the algorithm is given.
Consider now the resulting triangulation $T^4$: According to a definition given by Arkin [2](1), a triangulation $T$ is called Hamiltonian if the dual of $T$ contains a Hamilton path. This property is important, because in a Hamiltonian triangulation exists a perfect ordering, that means, it is possible to find a triangle strip in this triangulation.

Arkin gives also another important property:

"The dual of any triangulation is a tree, where the leaf nodes of the dual represent ears of the polygon, triangles with two of its three faces defined by the boundary of the polygon. A triangulation is Hamiltonian if and only if it contains exactly two ears – i.e., if and only if the dual graph is a path.” [2](2)

We show, that the triangulation $T$, computed by the given algorithm, is Hamiltonian:

The triangulation starts and ends in the points $a$ and $b$ as described above. We define, that the triangles, which contain both $a$ and $b$, don’t share an edge, i.e., that the edges $(a, b)$ in the first and in the last triangle are distinct.

---

[4] "Triangulation” means here: The graph $T = (V, E)$ with $V = V(A) \cup V(B)$ (the union of the vertices in $A$ and $B$) and $E = E(A) \cup E(B) \cup R$ (the union of the edges in $A$ and $B$ and in the resultset of the triangulation).
Algorithm TriangulateAnnulus:
Let $A$ and $B$ be the points of the outer and the inner hull, respectively and $T$ the triangulation of the annulus.

1. Choose a point $a_1 \in A$ and its neighbor $b_1 \in B$ as pair of startpoints.
2. Add $a_1 - b_1$ to $T$.
3. Create two lines of support as vertical lines through $a - l_1$ and through $b - l_2$.
4. Compute the angles $\alpha$, $\beta$ and $\gamma$.
5. Add vertices and edges to the triangulation and move the center of rotation for the support lines, depending on the angles:
   - If $\beta \leq 0$:
     - If $\alpha < \gamma$: Add edge $(a_{i+1}, b_i)$, move $l_1$ to $a_{i+1}$.
     - If $\alpha > \gamma$: Add edge $(a_i, b_{i+1})$, move $l_2$ to $b_{i+1}$.
     - Else: Add edges $(a_{i+1}, b_{i+1})$ and $(a_{i+1}, b_i)$, move $l_1$ to $a_{i+1}$ and $l_2$ to $b_{i+1}$.
   - Else:
     - If $\beta < \gamma$: Add edge $(a_{i+1}, b_i)$, move $l_1$ to $a_{i+1}$.
     - If $\beta > \gamma$: Add edge $(a_i, b_{i+1})$, move $l_2$ to $b_{i+1}$.
     - Else: Add edges $(a_i, b_{i+1})$ and $(a_{i+1}, b_{i+1})$, move $l_1$ to $a_{i+1}$ and $l_2$ to $b_{i+1}$.
6. Repeat from 4, until $a_1$ and $a_2$ are reached.

Figure 12: Algorithm for triangulating annuli

Furthermore, both triangles contain exactly one edge which is member of $A$ or $B$ (a second edge laying on one of the convex hulls is impossible, because the triangle is not closed in this case). The third edge is constructed between two points in $A$ and $B$, respectively. That means, that the triangle at the start of the triangulation shares only one face with its successor in the triangulation. The same is true for the triangle at the end and its predecessor. So, the triangles at the start and the end of the triangulation are ears of the triangulation.

We assume, that the triangulation is not Hamiltonian. Therefore, it has to contain more than two ears. Because the algorithm visits all points in
A and B and adds only edges to T which are shared by two triangles, a third ear requires a point which is not member of A or B. If such a point exists, it can only have a position (a) between A and B or (b) to the left of A or (c) to the right of B (the orientation is CW). In the last two cases, the point can’t be a member of a hull $A'$ around A or $B'$ inside B, otherwise the triangle belongs to triangulations of $A'$ and A or B and $B'$, respectively.

We consider the three cases:

- Case (a): The convex layers are constructed in such a way that no point can be found between A and B.
- Case (b): A point to the left of A belongs to another convex hull or doesn’t exist (if A is the convex hull of $P$).
- Case (c): A point to the right of B belongs to another convex hull or doesn’t exist (if B is the innermost layer of $C(P)$).

That means, a third ear doesn’t exist. Therefore, the algorithm constructs a Hamiltonian triangulation.

Pirzadeh shows, that the time complexity of this algorithm is $O(m + n)$, where $m$ is the number of points in A and n the number of points in $B$ [11](53ff).

2.2.3 Step 3: Build the triangle strips

The triangulation algorithm returns a list of point pairs $T$; in a pair, the first point belongs to A, the second to B. The sequence of points $S$, which represents a triangle strip as described in 2.1, has to be extracted from this list.

Consider the triangulation in figure 1: The triangulation for this polygon would be represented by the list in figure 13. The pair $(v_2, v_1)$ is the diagonal of the quadrilateral with cornerpoints $v_0, v_1, v_2, v_3$. In $S$, only the cornerpoints are necessary, therefore the pair $(v_2, v_1)$ can be ignored.

Because a diagonal is the line between the third and the second corner of the quadrilateral (the other diagonal between the first and the last point is not created by the triangulation algorithm), it can be detected in the list by comparing the points in the pair $p$ with the points in $p_{i-1}$ and $p_{i+1}$: If the first point in $p_i$ is identical with the first of $p_{i+1}$ and the second point in $p_i$ is identical with the second in $p_{i-1}$, then the pair represents a diagonal and can be discarded.
If the triangulation contains a single triangle, which requires a swap (as in figure 2), $T$ has the form as in figure 14.

If $p_i$ is the swap pair, it is obvious, that this pair will not be discarded: The first point in $p_i$ is not identical with the first of $p_{i+1}$ or the second point in $p_i$ is not identical with the second of $p_{i-1}$.

The following algorithm creates a sequence of points, which represents a triangle strip, from a triangulation list $T$ as described above:

**Algorithm StripSequence:**
Takes a list of point pairs $T$ and returns a sequence $S$.

- For all elements $p_i$ in $T$:
  - If $p_i(0) = p_{i+1}(0)$ and $p_i(1) = p_{i-1}(1)$: Discard $p_i$.
  - Else:
    * Add $p_i(0)$ to $S$.
    * Add $p_i(1)$ to $S$.  

Figure 13: Triangle strip

Figure 14: Triangle strip with swap
This algorithm runs in linear time: It visits all elements in the list $T$ only once, therefore the time complexity is $O(n)$ with $n$ - number of elements in the list.
3 Method

The algorithm presented in subsection 2.2 was implemented and tested with different datasets, containing both planar points and points in 3D space.

For the implementation, C++ is used as programming language; the implementation consists of a number of classes with the template classes PointSet_2D and PointSet_3D as core. An Octree is used as data structure for sorting the points of a 3D dataset. It is implemented by a template class, too.

In PointSet_3D, a number of functions provided by CGAL [5] are used to create the fitting plane for a 3D point set and to project the points onto this plane (to transform the points from the 3d to 2d space as mentioned in section 2.2). They are contained in the package for computing a Principal Component Analysis.

The most important class of the implementation is contained in PointSet_2D. It offers a set of functions for creating triangle strips from a planar point set: Computing the convex layers, triangulating the annuli and creating triangle strips from the triangulation.

The points in the planar point set are representations of the original points in the 3D point set, thus an algorithm, operating on the planar point set is indirectly operating on the 3D points. The relationship between original and representation is established by the index of the original point set – this causes, that in every step of processing planar points, the ID of the processed point has to be tracked. For this purpose, a structure, containing the point data itself and its ID is used.

CGAL offers several functions for computing the convex layers, but they can only process point objects and are not able to track the indices. The output of these functions is another set of points, so the relationship between input and output is lost. Therefore, it was necessary to create a ’tailormade’ function for computing convex hulls.

This function - computeHull - implements Graham’s Scan. It uses sortPointsByAngle for sorting the points around the reference point, here in clockwise direction. The sorting algorithm is QuickSort; the implementation is based on [16](99).

The function creates a convex hull which includes collinear points, laying on the borders of the hull (in figure 16, for example the points 3 and 1 between 16 and 6). The triangulation has to include all points of the point set, because of the aim of this method – to create a model of some object inherent in the point set. If the methods follows the definition given by O’Rourke\(^5\), a number of points would be missed.

\(^5\)O’Rourke defines, that hull points are only the corner points of a hull [10] (84).
For the triangulation of the convex layers, two functions were created. The first of them implements the triangulation of an annulus of a set of convex layers: The function `createTStrip` takes two convex hulls as arguments and returns the resulting triangulation in a vector of `int`-values. This function implements `TriangulateAnnulus`, presented in section 2.2.2.

The second one (`triangulateConvexPolygon`) is used, if the innermost convex hull contains more than two points and creates a triangle fan with the lowest leftmost point of the point set as center.

The comparison of angles in the algorithm (see section 2.2.2) in `createTStrip` is realized by computing a helper point $b_h$, which represents a line of support ($l_2$, see figure 7), and a test for a left or right turn of the points under consideration and the helper point. This point is computed from $b_1$ as starting point; it depends on the configuration of the calipers in the step before. When the algorithm is started, it is initialized by $b_h = b_1 + (1.0, 1.0)$.

An example can help to explain this: The positions of $a_2$, $b_1$ and $l_2$ determine, if $\beta > 0$ (figure 11). This is the case, if $p_2$ has a position at the right to the line of support $l_2$. Line $l_2$ is represented by $b_h$: That means, we have to test, if $p_2$ is right from the line $(b_2, b_h)^6$. The appropriate function to do this is `right_turn` with $b_i, b_h, a_{i+1}$ as arguments (in this order).

With a new point, the angles $\gamma$ and $\alpha$ can be compared: The vector $v_b$ from $b_1$ to $b_2$ is computed. Then, this vector is added to point $a_1$; the result is the point $g_h = a_1 + v_b$, which now can be used to determine the relationship between $\gamma$ and $\alpha$. For example: $\gamma < \alpha$, if $g_h$ is left from the line between $a_1$ and $a_2$.

![Figure 15: Comparing angles $\alpha$ and $\gamma$](image)

---

6 This technique is inspired by O'Rourke [10](30 ff.).

7 Tests, if the angle formed by $b_i$, $b_h$ and $a_{i+1}$ makes a right turn.
The other tests are performed in the same manner and, at the same time, $b_h$ is updated:

- If $\beta > 0$:
  - Right turn of $b_1$, $a_2$ and $b_2$: $\beta < \gamma$
    $$b_h = b_1 + a_2 - a_1$$
  - Left turn of $b_1$, $a_2$ and $b_2$: $\beta > \gamma$
    $$b_h = b_1 + b_2 - b_1$$
  - Else: $\beta = \gamma$
    $$b_h = b_1 + a_2 - a_1$$

- Otherwise: Here, $g_h$ is calculated for transforming the angle $\gamma$ to the point $a_1$ (figure 15).
  - Right turn of $a_1$, $a_2$ and $g_h$: $\alpha < \gamma$
    $$b_h = b_1 + a_2 - a_1$$
  - Left turn of $a_1$, $a_2$ and $g_h$: $\alpha > \gamma$
    $$b_h = b_1 + b_2 - b_1$$
  - Else: $\alpha = \gamma$
    $$b_h = b_1 + a_2 - a_1$$
4 Result

The algorithm for the triangulation of planar point sets (implemented in \texttt{PointSet\_2D}) was tested with a dataset from O’Rourke’s book \cite{10}\cite{95} with some new points added. The data for this point set can be found in appendix A.

The following figures illustrate the process:

During the first step, the convex layers are computed (figure 16).

![Figure 16: Convex layers](image)

After this, the layers are triangulated (figure 17).

The complete sequence of the triangulation of the first layer is visualized by the images in appendix C.

The final step is the creation of triangle strips, i.e. a sequence of point ID which can be used for defining triangle strips for rendering.

The results are for Layer 1:

9 5 14 5 8 5 7 5 16 0 3 0 1 0 6 0 18 12 13 15 12 10 17 11 17 9 5

For layer 2:

5 4 0 20 12 19 17 2 5 4

For layer 3:

4 20 4 19 4 2

As one could expect (the number of points in the outer hull is very different from the number of points in the middle hull), several swaps are included in the first sequence as for instance 9 5 14 5 8 5. ‘Real’ stripified parts of the first sequence are, for example, 7 5 16 0 and 6 0 18 12. However,
the strip computed for the second layer is a 'real' triangle strip. Figure 18 shows the final result.

The algorithm for processing 3D point sets was tested with a dataset from the Stanford 3D Scanning Repository [12] (the reconstructed data of the Stanford Bunny with 35947 points). This data was read and, as explained above, sorted in an Octree (figure 19).

Every point set was then processed: With the help of functions, offered by CGAL [5], a fitting plane was created and the points projected onto this plane. By the x- and y-coordinates of the projections, a set of 2D points was created.

From this point set, triangle strips were created; the result set contains
the point-ID of the 3D points. This data was finally used to render the model. In figure 20, an overall view of the result is presented.

Figure 21 shows a detail of the rendering from the inside of the model. The triangle strips are visible, and a problem of the strips, too: Some regions are very long and narrow\(^8\). Besides this, some undesirable artifacts are visible (the blue structure in one of the patches).

The same region can be viewed in figure 22; now, the triangle strips are represented by lines. Here, the long, narrow triangles becomes even more visible; very clear is this at the set in the middle, where some triangles are almost as long as the whole set.

In the upper part of the image, the set with the big blue area is positioned, and one can see, that blue triangles are stretched over other areas inside the triangle strip. At the moment, the cause for this effect is not known.

One problem appearing during the tests of PointSet\_2D should be men-

\(^8\)Long and narrow triangles are not desired, because they cannot be rendered efficiently.
tioned at the end: The Bunny dataset contains very small numbers (example: 0.000420458:0.036172:-0.0244623). If these numbers are used, the sorting algorithm works no longer correct, because the computed area of a triangle is not correct: For a triangle, where two points are identical, the result was $\neq 0$ and the algorithm didn’t terminate. To solve the problem, it was necessary, to multiply the point data by a factor of 100 000.

A similar effect appeared under processing of some clusters from the 3D point set with positions nearby $x = 0$. The program execution was abandoned with the error message 'segmentation fault’. To solve this, all points of the set were translated by 1 unit in $x$, $y$ and $z$ direction and the coordinates scaled by a factor of 100000000.0.
5 Discussion

The results in section 4 show, that the algorithm for processing planar point sets works as expected, but more tests with different datasets have to be made to get more information about the quality of the triangulation, i.e. the relations between the sidelength of the created triangles.

Vaněček och Kolingerová stated, that the quality of a triangulation created by onion peeling is not very high [15](102). On the other hand, the
triangulation is determined by the shape of the input data:

The Octree partitions the space into small cubes, each containing one subset of the point set. The shape of a subset is determined by the distribution of the points in space and by the borders of the cube — by the cube, a patch is 'cut out', which results in quadrangular or triangular shapes as visible in figure 21.

If the point set is quadrangular in shape, the points in the corners are members of the convex hull, and if the other points are positioned in the interior, they are the only members of the outer hull. Then, the triangulation creates long triangles stretching over the whole length of a side.

Maybe the result would be better, if the point set has a disklike shape. Thus, we have to consider other clustering algorithms.

The algorithm for computing a set difference uses a double loop and has therefore a time complexity of \(O(n^2)\). This is the most inefficient part of the method which causes, that the overall time complexity is \(O(n^2)\).

One possibility would be to replace this algorithm by a more efficient one. Another alternative is to replace the complete algorithm for computing convex layers: Chazelle presented an algorithm with a time complexity of \(O(n \log n)\) [6]. With this algorithm, an overall time complexity \(O(n \log n)\) could be achieved.

A third algorithm is the "Spiraling Procedure" by Bose and Toussaint [3], which creates a spiral-formed triangulation of the given point set and has linear time complexity. The disadvantage of the last algorithm is that the resulting triangulation depends on the starting point and the working direction.

In the method used in this work, a number of sequences of points or point indices is returned, representing triangle strips. It is possible to go a step further: The triangle strips computed by the current method could be connected to build one strip from the pieces. A simple method for achieving this is to insert four degenerated triangles between two single strips [1](552). In this way, even strips which are not neighbors can be connected.

Another possibility for collecting the triangle strips is to divide all the strips in a lower and an upper part. Then, these parts can be connected in a meanderlike manner as in figure 23.

The advantage of such a structure is, that the strip of a point set starts and ends at the border of the point set, i.e. the convex hull. This would give the possibility to connect the triangle strips of a neighborhood of point sets. At the same time, the space between the point sets could be triangulated and, in the end, a complete model of the surface, given by the set of point sets, can be created.
To achieve this, more work is necessary in relationship to 3D point sets. To build valid and renderable models requires knowledge about the orientation of the point sets in space. With this knowledge, it would be possible to determine the directions in the triangles and to calculate the normal vectors for the created triangles. Thus, the presented algorithm should be completed by methods for determining the orientation of the parts of a clustered pointcloud and by datastructures for managing the informations about orientation, direction and normalvector of the computed triangles.

6 Conclusions

In this work, a method for creating triangle strips from point sets is presented and an implementation to demonstrate it. This method is based on the convex layers.

The main parts of the method are an algorithm for computing the convex layers of a point set and second algorithm for triangulating the annuli of a set convex layers, which employs the Rotating Calipers [13]. Some proposals were made to improve the algorithm.

At the end, a number of possibilities for further development of the method are discussed.
References


A  Testdata

1:  3.0, 3.0
2:  3.0, 5.0
3:  0.0, 1.0
4:  2.0, 5.0
5:  -2.0, 2.0
6:  -3.0, 2.0
7:  6.0, 5.0
8:  -3.0, 4.0
9:  -5.0, 2.0
10: -5.0, -1.0
11:  1.0, -2.0
12: -3.0, -2.0
13:  4.0, 2.0
14:  5.0, 1.0
15: -5.0, 1.0
16:  3.0, -2.0
17:  0.0, 5.0
18:  0.0, 0.0
19:  7.0, 4.0
20:  1.0, 1.0
21:  2.0, 2.0

B  Log

Filling point set with set 1
Computing hulls: Sorting points: ...OK
...OK
Set difference:
  0 => 3 : 3
  2 => 0 : 1
  4 => -2 : 2
  5 => -3 : 2
  12 => 4 : 2
  17 => 0 : 0
  19 => 1 : 1
  20 => 2 : 2
Computing hulls: Sorting points: ...OK
...OK
Set difference:
  2 => 0 : 1
  4 => -2 : 2
Computing hulls: Sorting points: ...OK
...OK
Set difference is empty
Convex hull 0:
Number of points: 13
Leftmost lowest point: id = 11 => -3 : -2
Rightmost highest point: id = 6 => 6 : 5
11 => -3 : -2
9 => -5 : -1
14 => -5 : 1
8 => -5 : 2
7 => -3 : 4
16 => 0 : 5
3 => 2 : 5
1 => 3 : 5
6 => 6 : 5
18 => 7 : 4
13 => 5 : 1
15 => 3 : -2
10 => 1 : -2
Convex hull 1:
Number of points: 4
Leftmost lowest point: id = 17 => 0 : 0
Rightmost highest point: id = 0 => 3 : 3
17 => 0 : 0
5 => -3 : 2
0 => 3 : 3
12 => 4 : 2
Convex hull 2:
Number of points: 4
Leftmost lowest point: id = 2 => 0 : 1
Rightmost highest point: id = 20 => 2 : 2
2 => 0 : 1
4 => -2 : 2
20 => 2 : 2
19 => 1 : 1
Triangulating convex hulls...
Stripify 0 of 2
edge 9:5
ddge 14:5
dge 8:5
dge 7:5
edge 16:5
edge 16:0
edge 3:0
edge 1:0
edge 6:0
edge 18:0
edge 18:12
edge 13:12
edge 15:12
edge 10:12
edge 10:17
edge 11:17
edge 9:17
edge 9:5

Stripify 1 of 2
edge 5:4
edge 0:4
edge 0:20
edge 12:20
edge 12:19
edge 17:19
edge 17:2
edge 5:2
edge 5:4

Triangulating convex polygon
edge 4:20
edge 4:19
edge 4:2

Triangle strips:
9 5 14 5 8 5 7 5 16 0 3 0 1 0
   6 0 18 12 13 12 15 12 10 17 11 17 9 5
5 4 0 20 12 19 17 2 5 4
4 20 4 19 4 2
C Figures

Figure 24: Triangulating the outer annulus, step 1

Figure 25: Triangulating the outer annulus, step 2
Figure 26: Triangulating the outer annulus, step 3

Figure 27: Triangulating the outer annulus, step 4

Figure 28: Triangulating the outer annulus, step 5
Figure 29: Triangulating the outer annulus, step 6

Figure 30: Triangulating the outer annulus, step 7

Figure 31: Triangulating the outer annulus, step 8
Figure 32: Triangulating the outer annulus, step 9

Figure 33: Triangulating the outer annulus, step 10

Figure 34: Triangulating the outer annulus, step 11
Figure 35: Triangulating the outer annulus, step 12

Figure 36: Triangulating the outer annulus, step 13

Figure 37: Triangulating the outer annulus, step 14
Figure 38: Triangulating the outer annulus, step 15

Figure 39: Triangulating the outer annulus, step 16

Figure 40: Triangulating the outer annulus, step 17