NONLINEAR INTERPOLATION BETWEEN SLICES

RESEARCH THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Master of Science in Computer Sciences

Amir Vaxman

Submitted to the Senate of
the Technion- Israel Institute of Technology

Kislev, 5767 Haifa
November 2006
Acknowledgement

This Research Thesis was done under the supervision of Prof. Gill Barequet in the Faculty of Computer Science.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GREATLY ACKNOWLEDGED.
Contents

Acknowledgement iii

Abstract 1

1 Overview 3
   1.1 Background .................................................. 3
   1.2 Preliminaries ................................................. 4
      1.2.1 Basic differential geometry of curves and surfaces .... 4
      1.2.2 Straight skeletons ....................................... 8
   1.3 Overview of Our Method .................................... 9
      1.3.1 Problem definition ....................................... 9
      1.3.2 Our contribution ......................................... 10
      1.3.3 Previous work ............................................ 11
      1.3.4 Outline of the algorithm ................................ 12
   1.4 Structure of the Dissertation ............................... 12

2 Straight-Skeleton-Based Matching 15
   2.1 Matching Vertices in Active Cells ........................ 15
   2.2 Properties of the Matching .................................. 17
   2.3 Lifting Up in Space .......................................... 21
   2.4 Surface Orientation .......................................... 22

3 Creating a Curve Network 27
   3.1 Extracting First-Order Data ................................. 27


3.1.1 Horizontal direction ............................................. 28
3.1.2 Vertical direction ................................................. 29
3.2 Creating the curve network ......................................... 30

4 Building Surface Patches ........................................... 35
  4.1 Interpolating Curve Networks ..................................... 35
  4.2 Interpolating Quadrilateral and Triangular Gregory Patches ............................................. 36
    4.2.1 Quadrilateral Gregory patches .......................................... 36
    4.2.2 Triangular Gregory patches .............................................. 38
    4.2.3 Interpolation along a common boundary .................................... 39
  4.3 Interpolation with $k$-Sided Patches ............................ 40

5 Results ................................................................. 43
  5.1 Complexity of the Algorithm ...................................... 43
  5.2 Experimental Results ............................................. 44
    5.2.1 Complex examples ................................................... 44
    5.2.2 Synthetic examples ................................................ 45

6 Conclusions and Future Work ........................................ 55
  6.1 Summary ............................................................ 55
  6.2 Shortcomings of the Algorithm ................................... 55

Bibliography ............................................................ 59
List of Tables

5.1 Time measurements for several inputs. Excluded are irrelevant visualization methods (such as patch tesselation). Curve network time includes first-order data extraction .......................... 44
List of Figures

1.1 An example of a polygon ABCDE and its straight skeleton (The propagation is in dashed lines). The edges EA and CD vanish in edge events, and B splits the edge DE ....................................................... 9

1.2 An example of surface reconstruction from two slices ................................. 13

2.1 An example of a partial matching following some events in the active cell. The skeleton is shown by solid lines, the progression of cell edges by dashed lines, and the matching by dashed-dotted lines. The matching GH is due to an edge event at $S_3$, while IB is actually two matching edges: a split event at $S_2$ followed by an edge event at $S_3$. KI is due to a split event of A and I at $S_1$, and KA is due to an edge event at $S_4$. ........................................... 17

2.2 An example of lemma 2.2.2. The active-cell edges are in solid lines, the bisectors (and the line of sight between $p$ and $q$) are in dashed-dotted lines, and the distances to the edges are in dashed lines. Here, $b_r$ forms an event with $b_q$, because $d_{e_2} < d_{e_p}, d_{e_q}$, and so prevents any event of $p, q$ that would have created the matching. ....................................................... 19

2.3 An example of a polygon with holes. Here, every path the matching $m_{rs}$ takes, as a trace of the movement of its vertices, cannot exist; The path exemplified crosses the matching of the vertex $x$ with any vertex of the outer boundary from its right, as the propagation of $x$ and the outer boundary would cause. ....................................................... 20
2.4 Matching decisions in intersection vertices. Figure 2.4(a) shows the top view of a pure intersection, that is also shown in isometric views by Figures 2.4(c) to 2.4(b) which demonstrate the twist in the surface. Figures 2.4(d) and 2.4(e) exemplify intersection vertices that are meeting points.

2.5 Orientation of an overlay of two squares. The faces that are visible are oriented CCW. In addition, the original contours are oriented with only one (opposite) half-edge each.

3.1 The dissimilarity between $L_2$ and its adjacent slices is reflected in the matchings $m_{AB,1}, m_{AB,2}$ which matched vertices $A$ and $B$ in the overlay of slices $L_2$ and $L_3$, and the overlay of slices $L_1$ and $L_2$, respectively. The $z_{\text{diff}}$ component differentiates between these two matches, and produces a smoothly vanishing object on both sides.

3.2 An example of the effect of adding a component in the $Z$ direction for up- and down-self edges.

3.3 An example of the calculation of the normal $\hat{N}$ at the vertex $P$. Four directions are acknowledged from matching arcs, and the $z_{\text{diff}}$ component is added to $M_4$, as it is a self edge. The direction $\hat{H}$ is already given from the spline interpolation of the middle contour.

3.4 Projection of the vector $\vec{AB}$ into the tangent plane at vertex $A$, with normal $\hat{N}_A$. The projected vector $\hat{T}_A$ is then scaled to the length of $\vec{AB}$, and the result is $\hat{T}'_A$.

4.1 The structure of control points of a Gregory patch. The control polygon is in dashed lines, and the independent inner points which are blended are connected in dotted lines.

4.2 Determining inner points across a boundary between two patches.

4.3 Subdividing a 7-sided patch. The subdivision lines are produced in the following order: $v_1 - v_3, v_3 - v_5$ and $v_5 - v_7$, creating three triangular and one quadrilateral patch that are interpolated.
5.1 A reconstructed heart. 5.1(a) Shows an overlay of two adjacent slices in the input, 5.1(b) shows the curve network, showing splines of original contours (blue), matching curves (green) and subdivision curves (purple). 5.1(c) and 5.1(d) show the final results with shading .......................... 46
5.2 A pair of lungs ......................................................... 47
5.3 A Pelvis ................................................................. 48
5.4 A hip bone ............................................................... 49
5.5 An example of a reconstructed topographic map ................. 50
5.6 Timing results for the algorithm ...................................... 51
5.7 The trachea area of the reconstructed lungs ...................... 52
5.8 The major arteries of the reconstructed heart ...................... 52
5.9 A reconstruction of a synthetic example, featuring a smooth solution to the branching and correspondence issues ......................... 53
5.10 Another synthetic example, featuring a smooth solution to a considerable dissimilarity ....................................................... 54

5.1 A reconstructed heart. 5.1(a) Shows an overlay of two adjacent slices in the input, 5.1(b) shows the curve network, showing splines of original contours (blue), matching curves (green) and subdivision curves (purple). 5.1(c) and 5.1(d) show the final results with shading .......................... 46
5.2 A pair of lungs ......................................................... 47
5.3 A Pelvis ................................................................. 48
5.4 A hip bone ............................................................... 49
5.5 An example of a reconstructed topographic map ................. 50
5.6 Timing results for the algorithm ...................................... 51
5.7 The trachea area of the reconstructed lungs ...................... 52
5.8 The major arteries of the reconstructed heart ...................... 52
5.9 A reconstruction of a synthetic example, featuring a smooth solution to the branching and correspondence issues ......................... 53
5.10 Another synthetic example, featuring a smooth solution to a considerable dissimilarity ....................................................... 54

6.1 The result of the reconstruction of two shapes with a small intersection area. Even though the shapes are quite similar, the given overlay does not produce the best result possible ........................................ 56
6.2 The matching produces long skinny triangles, and the result shows unwanted “ripples” .............................................................. 57
Abstract

The topic of interpolation between slices has been an intriguing problem for many years, as it offers means to visualize and investigate a 3D object given only by its level sets. A slice consists of multiple non-intersecting simple contours, each defined by a cyclic list of vertices. An interpolation solution matches between a number of such slices (two or more at a time), providing means to create a closed surface connecting these slices, or the equivalent morph from one slice to another. Most interpolation methods either interpolate the surface between two slices based on these slices alone, which can cause abrupt changes in the surface, or solve the problem as a global optimization problem, which is computationally expensive.

We offer a method to incorporate the influence of more than two slices at each point in the reconstructed surface. We investigate the flow of the surface from one slice to the next by matching vertices and extracting differential geometric quantities from that matching. Interpolating these quantities with surface patches then allows a nonlinear reconstruction which produces a free-form, non-intersecting surface. No assumptions are made about the input, such as on the number of contours in each slice, their geometric similarity, their nesting hierarchy, etc., and the proposed algorithm handles automatically all branching and hierarchical structures. Unlike polyhedral-reconstruction methods, the resulting surface is smooth, and it does not require further subdivision measures.
Chapter 1

Overview

1.1 Background

Reconstruction of a surface, either polyhedral or free-form, from a set of consecutive parallel cross-sections, has many applications and, therefore, attracted a substantial amount of work since the 1970’s. The input to such a problem is usually a set of slices containing contours that represent some type of data: A general volumetric object scanned at its level sets, geographic height lines, MRI slices, CT scans, and more. The problem can be formally defined thus: Given a set of \( n \) consecutive slices \( \{L_0, L_1, \ldots, L_{n-1}\} \), each composed of closed simple contours (that can be nested but otherwise not intersecting), and that are located in the \((x, y, z_i)\) plane for a constant \(z_i\), \(0 \leq i \leq n - 1\), construct the boundary surface \(S\) of an object \(O\), such that \(S(x, y, z = z_i) = L_i\). This problem is ill-posed and underconstrained; therefore, infinite solutions exist (unless we impose some restrictions on the output surface), and one must set up a heuristic in order to constrain the problem to a unique solution. This reconstruction poses some characteristic issues to be dealt with:

- **Correspondence**: How to match vertices of different slices in order to create a consistent non-self-intersecting mesh. This issue also involves branching (“one-to-many” correspondence), dealing with geometric dissimilarities, and, in polyhedral reconstruction methods, tiling (how to set up a 2-manifold by simple polygons on a matching).
• **Optimality**: Creating the best surface, in terms that are subjective, but well-defined for each solution.

• **Robustness**: Can a method be fully automatic, or should it make prior assumptions on the input slices. In addition, should an algorithm rely on the user for tuning of parameters.

### 1.2 Preliminaries

#### 1.2.1 Basic differential geometry of curves and surfaces

**Differential geometry of curves**

A parametric space curve $C(t)$ is a map $C : \mathbb{R} \mapsto \mathbb{R}^3$ from an interval $I \subseteq \mathbb{R}$ to a vector field $C(t) = \{x(t), y(t), z(t)\}$ in $\mathbb{R}^3$, where $x, y, z$ are scalar functions. The derivative of a curve is defined as $C'(t) = \{x'(t), y'(t), z'(t)\}$. A parametric curve is called **regular** if for all $t \in I$ we have $C'(t) \neq 0$.

There are a few properties that are unique to any space curve, and are invariant to regular parametrizations:

- The **arclength** of a curve is computed as $s(t) = \int_{t_0}^{t} \|C'(t)\| \, dt$, and a curve is called arclength parameterized if it is represented as $C(s) = C(s(t))$. For every regular parametrization $t$, the fundamental theory of calculus yields $\frac{ds}{dt} = \frac{d}{dt} \int_{t_0}^{t} \|C'(t)\| \, dt = \|C'(t)\|$, and, therefore, $C'(s) = \frac{C'(t)}{\|C'(t)\|}$. Arclength can be thought of the length of the ‘string’ which the curve is made from by twists and bends. The Euclidean norm of the derivative of a curve is the “speed” in which a particle is moving along the curve’s path when advancing uniformly on the parameter $t$ (as time). In that manner, arclength curves are unit speed curves, since $\|C'(s)\| = 1$ for all $s$.

- The **unit tangent vector** to a space curve at every point is computed as $\hat{T}(t) = \frac{C'(t)}{\|C'(t)\|}$ (i.e., $\hat{T}(s) = C'(s)$). This is the unit speed direction of advancement on the curve.
1.2. PRELIMINARIES

- The curvature of the curve, which is computed as \( \kappa(t) = \|C''(s)\| = \|\frac{C''(t) \times C'''(t)}{\|C''(t)\|^2}\| \), represents \( 1/r \), where \( r \) is the radius of the osculating circle of \( C \) at \( t \). Thus, places of high curvature in the curve are where the osculating circle is relatively small, and vice versa (a line in space has an infinite osculating circle, and, therefore, zero curvature).

Bézier curves

Bézier curves, which were introduced almost simultaneously by DeCastlejau and Bézier,\(^1\) are polynomial curves of degree \( n \) (sometime denoted as of order \( n + 1 \)) on the interval \( t : [0, 1] \), which are computed using a control polygon—a connected polyline \( \{P_0, P_1, \ldots, P_n\} \)—and the Bernstein blending functions of degree \( n \): \( \theta^a_n(t) = \binom{n}{i} t^i (1 - t)^{n-i} \), such that \( C(t) = \sum_{i=0}^{n} P_i \theta^a_n(t) \). Bézier curves have the following characteristics:

- Since \( 0 \leq \theta^a_n(t) \leq 1 \), \( C(t) \) is contained in the convex hull of the control polygon.

- The Bernstein blending functions of degree \( n \) are a basis of all polynomials of degree \( n \). That is, every polynomial curve of degree \( n \) can be expressed as a Bézier curve of the same degree (or higher).

- The derivative of a Bézier curve is \( C'(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) \theta^a_{n-1}(t) \). Higher-order derivatives can be computed similarly.

- \( C(0) = P_0, C(1) = P_{n-1}, C'(0) = n(P_1 - P_0), C'(1) = n(P_n - P_{n-1}) \). In the case of cubic (degree 3) curves, and in a similar fashion to the Hermite interpolation curves, a Bézier curve is defined uniquely by the locations and derivatives at its endpoints.

- Bézier curves can be elevated to a higher-degree polynomial using a simple procedure.

Continuity constraints

Two curves \( C_1(t), C_2(t), t \in [0, 1] \) are said to be \( C^k \)-continuous together if, for all \( 0 \leq i \leq k \), \( C^{(i)}(1) = C^{(i)}(0) \) (or vice versa). This continuity can be relaxed to the Geometric

\(^1\)Bézier formulated the blending function representation which is shown here, while DeCastlejau has shown a subdivision algorithm. Both are essentially the same.
continuity $G^k$—two curves are $G^k$-continuous if they can be locally parameterized at the joint point to be $C^k$.

Thus, two $G^0$-continuous curves attach at the endpoints. Two $G^1$-continuous curves share the same unit tangent at the same endpoints (whereas $C^1$ curves also share the same derivative), and two $G^2$-continuous curves also share the same curvature.

Further information about Bézier curves and curves in general can be sought in [13].

Differential geometry of surfaces

A parametric surface is a map $C: \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}^3$ from a two-dimensional domain $I_u \times I_v \subset \mathbb{R} \times \mathbb{R}$ to a vector field $S(u, v) = \{x(u, v), y(u, v), z(u, v)\}$. A partial derivative of a surface is, consequently, $S_{u|v}(u, v) = \{x_{u|v}(u, v), y_{u|v}(u, v), z_{u|v}(u, v)\}$. Should $S_u \times S_v \neq 0$ hold everywhere in the domain, a parameterization $(u, v)$ of a surface would be called regular. There is no equivalent for arclength on surfaces.

The derivative vectors $\{S_u, S_v\}$ span the tangent plane to the surface at a point $(u, v)$. In regular surfaces, this quantity is well-defined. Using these, the unit normal to the surface is defined as $\hat{N}(u, v) = \frac{S_u \times S_v}{\|S_u \times S_v\|}$.

The tangent vector of a curve embedded on the surface that passes through a point $S(u, v)$ is contained within the tangent plane at that point. For a given direction $(x_1, x_2)$ in the tangent plane, the curve’s tangent vector is in the directional derivative of the surface: $f_t = x_1 f_u + x_2 f_v$. The Riemann metric of a surface, which is a matrix of the form

$$RM(u, v) = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} = \begin{pmatrix} S_u^2 & S_u S_v \\ S_v S_u & S_v^2 \end{pmatrix},$$

is a measure of the surface used to determine first order data on the surface at a point, such as the arclength of the embedded curve $C(t) = S(u(t), v(t))$, which is computed as $ds^2 = S_{11} u_t^2 + 2S_{12} u_t v_t + S_{22} v_t^2 = I(u_t, v_t)$. This property is called the First Fundamental Form, and is a geometric property of the surface, invariant under reparameterization.\(^2\) Thus, every curve passing through the surface at a point $(u, v)$ and in the same direction has the same arclength.

\(^2\)Although in general the Riemann metric is not invariant to reparameterization, the first fundamental form always is.
1.2. PRELIMINARIES

Surface continuity constraints

Two surfaces $S_1(u, v), S_2(u, v)$ are $C^k$-continuous along a common boundary, without loss of generality, $S_1(1, v) = S_2(0, v)$, if the respective partial derivatives agree everywhere. As in the case of curves, this can be relaxed to $G^k$, where both surfaces can be reparameterized locally to be $C^k$. Consequently, two surfaces are $G^1$-continuous across a boundary if they share a common tangent plane on it, i.e., the normal to the joint surface is a continuous function. This property is usually called tangent plane continuity.

Bézier tensor-product surfaces

A Bézier tensor product surface (also called a Bézier patch) of degree $n \times m$ over the interval $U \times V = [0, 1] \times [0, 1]$ is defined by a matrix of control points, and the Bernstein basis functions of degree $n$ and $m$ in the following manner:

$$P = \{ P_{ij} \in \mathbb{R}^3 | 0 \leq i \leq n, 0 \leq j \leq m \}, \quad S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} \theta^n_i(u) \theta^m_j(v).$$

Here are some important properties of these patches:

- The boundaries of a patch are Bézier curves. Namely, $S(u, 0) = \sum_{i=0}^{n} P_{i0} \theta^n_i(u)$, and the other three boundaries follow in the same fashion.

- The partial derivatives at the corner $(0, 0)$ (and, similarly, at the other three corners) are $S_u(0, 0) = n(P_{10} - P_{00})$ and $S_v(0, 0) = n(P_{01} - P_{00})$. That is, these derivatives depend solely on the boundary curves $S(u, 0), S(0, v)$. However, the cross-boundary derivatives ($S_u(0, v), S_v(u, 0)$ for example), that are required for $G^1$ continuity, depend on the inner control points. This property poses some problematic issues when joining surfaces along a common boundary or point, and is discussed in Chapter 4.
1.2.2 Straight skeletons

Definition and computation

The straight skeleton, introduced in [1], can be viewed as an approximation of the Medial Axis of a polygon. The skeleton is the result of offsetting all the edges inwards at equal speed, tracing the movement of vertices along the angular bisectors of adjacent edges, until the polygon vanishes. Two types of events occur in the course of this process (see Figure 1.1):

1. **Edge event**: An edge shrinks to a point, and two bisectors meet to create a node of the skeleton.

2. **Split event**: An edge splits when the bisector of an opposite vertex (denoted as a Reflex Vertex) hits it. A skeleton node is created here as well. The polygon then splits into two polygons that continue to shrink to zero. If the reflex vertex and the edge are from different polygonal chains originally (such as a polygon and its hole), this might lead to uniting of two offsetting fronts.

The union of the traces of all bisectors in the process is the straight-skeleton of the polygon \( S(P) \), and is a unique structure that serves as a partition of the polygon. The area that an edge \( e \) sweeps along the process is called the face of \( e \).

Various algorithms exist in the literature to compute the straight skeleton of a polygon, all of which are based on the simulation of the shrinking process, such as the one given in [1]. Unfortunately, since local changes in the polygon have a significant global effect on its straight skeleton, the algorithms to compute it are limited in utilizing traditional methods, such as divide-and-conquer or incremental methods. In this work, we employ the algorithm of [14] which runs in \( O(n(r + \log n)) \) (where \( n \) is the number of vertices in the polygon and \( r \) is the number of its reflex vertices), because of its intuitive approach. The fastest (running-time-wise) algorithm known in the literature is due to Eppstein and Erickson [12], which running time is \( O(n^{1+\varepsilon} + n^{8/11+\varepsilon} + r^{9/11+\varepsilon}) \) for any \( \varepsilon > 0 \).

The straight skeleton \( S(P) \) has the following properties, investigated in [1]:
1.3. OVERVIEW OF OUR METHOD

Figure 1.1: An example of a polygon ABCDE and its straight skeleton (The propagation is in dashed lines). The edges EA and CD vanish in edge events, and B splits the edge DE

- The straight skeleton of a simple polygon is a tree. For an \( n \)-gon \( P \), there are \( n \) faces, \( n - 2 \) vertices and \( 2n - 3 \) edges (not including the original vertices and edges of \( P \)) in \( S(P) \). In case a polygon has \( h \) holes, \( h \) edges can be removed from the skeleton to produce a tree. When a polygon is not in general position (e.g., when more than one edge event occurs at the same point, like in a regular polygon), some skeletal edges may be degenerate (having a zero length).

- Exactly three skeletal edges (considering degenerates, which create skeletal nodes with zero-length edges between them) coincide at each skeletal inner node.

- Every face of \( S(P) \) is monotone with respect to its defining edge of \( P \).

1.3 Overview of Our Method

1.3.1 Problem definition

Given a set of consecutive planar cross-sections of an unknown object, consisting of multiple non-intersecting simple contours, each defined by a cyclic list of vertices, reconstruct
the boundary of the object, which is a 3D surface of the following characteristics:

- The surface interpolates the vertices of the cross-sections. Therefore, a cross-section of the surface in the appropriate plane is a planar smooth interpolation of the vertices.

- The interpolated surface consists of surface patches, constrained with tangent plane continuity between neighboring patches, rendering the surface to be (at least) $G^1$-continuous everywhere.

- The surface is non-self-intersecting.

- The surface is a closed 2-manifold, and so it has a proper orientation.

In addition, the algorithm has to be robust; it should automatically handle input slices comprised of any number of contours, and all cases of branching and hierarchical structures (such as holes). In addition, the result should be visually pleasing as much as possible.

### 1.3.2 Our contribution

We use a robust matching technique between vertices of the original contours and intersection vertices of their overlays, based on the computation of straight-skeletons of the cells of the symmetric difference of two consecutive slices at a time (as classified by [4]). Thus, we inherit the full generality of [4], without intermediate contour interpolation, which introduces linearity, and extra complexity to the sought surface.

This matching yields a graph representing the flow of the surface, from which we extract first-order data and to which we fit a cubic curve network, and then $G^1$-continuous surface Gregory patches, comprising the desired surface. We prove that the straight-skeleton-based matching works for any instance of the problem, and thus the algorithm works for any set-up of contours. The acquiring of first-order data requires the incorporation of more than two slices at each vertex, and, thus, this algorithm produces a smooth surface even when there are abrupt changes between slices. In addition, albeit using information acquired from more than two slices, the solution is local, and does not require solving global systems of constraints.
1.3. OVERVIEW OF OUR METHOD

1.3.3 Previous work

Most of the work done so far on this subject offered solutions for piecewise-linear reconstruction between two adjacent slices. Only a few early works [3, 6, 7] offer a solution to the problem in full generality. Recent works [4, 24, 15] suggest using the straight skeleton, in order to solve the problem of reconstructing a surface with multiple contours of arbitrary geometry. The works [4, 28] base their solutions on this skeleton to create a general algorithm for reconstruction, imposing no constraints on the input. They utilize the straight-skeleton computation algorithm offered by [14]. These methods, although dealing well with robustness, create linear solutions based on stacking the interpolation results of only two slices at a time, and do not consider the general flow of the sought surfaces. In such solutions, the linearity cannot always be overcome by subdivision.

A few works deal with the problem of a general reconstruction, incorporating more than two slices at a time. In [5], a piecewise-linear solution is offered, considering the slopes of triangles from neighboring layers. The focus of most recent works is on PDE’s or level-set solving methods for creating a smooth surface [8, 10, 17, 23, 27], providing means to match contours, or to morph between adjacent curves. In order to create a smooth surface from a given reconstruction, some works utilize mesh smoothing techniques [27, 28] which introduce extra complexity to the surface in order to smooth it after the reconstruction. Other works offer to build iso-surfaces, a volumetric representation directly [10, 23], which is good for visualization, but not as a geometric entity, or an implicit surface [2], which is naturally smooth, but requires the solving of general systems of equations. Some works resort to general free-form interpolation techniques, a great deal of which is summarized in the survey [21]. Some of these methods (e.g., [8, 17]) offer to output free-form surfaces, yet they use intermediate contours and address a terrain reconstruction, which is a special case of the slice-interpolation problem. We are not aware of any general method that creates an explicit continuous surface directly from the input, that considers more than two slices, and that can be used both directly for visualization and conveniently for tessellation, such as the one presented in this work.
1.3.4 Outline of the algorithm

The algorithm proceeds with the following steps (exemplified in Figure 1.2):

1. Reading the sequence of input slices.

2. Computing the overlay of each pair of consecutive slices. The cells of the symmetric difference are identified and their straight-skeletons are computed (using the algorithm of [14]).

3. Based on the structure of the straight skeleton of each cell, the vertices comprising it are matched. This step creates a matching graph (so-called flow graph), in which all connections (arcs) are between a slice to itself or to its adjacent slices.

4. An orientation is imposed on the surface, considering the flow graph as an underlying mesh to the complete surface (in which faces are not necessarily planar).

5. First-order quantities (tangent planes, used later for the $G^1$ constraints) of the sought surface are computed from the flow graph at the vertices.

6. A network of cubic Bézier curves is constructed from the arcs of the flow graph.

7. Gregory surface patches are constructed from the faces of the flow graph, interpolating the curve network. This step completes the algorithm and produces the final surface.

1.4 Structure of the Dissertation

In Chapter 2 the matching algorithm and the construction of the flow graph are detailed, their properties are investigated, and the orientation process that produces the topology of the surface is considered. In Chapter 3 the acquisition of first-order data and the creation of the curve network are shown. Finally, Chapter 4 explains the creation of the surface patches that fully interpolate the curves into the final surface.
1.4. **STRUCTURE OF THE DISSERTATION**

(a) Overlay of two slices

(b) Active cells and straight skeletons ($U \setminus L$ cells in green, $L \setminus U$ cells in blue)

(c) The matching of vertices (shown by dashed-dotted arcs)

(d) The curve network

(e) The reconstructed surface

Figure 1.2: An example of surface reconstruction from two slices
Chapter 2

Straight-Skeleton-Based Matching

In this chapter, the method to create the flow graph of the object is explained. This method is based on the matching of vertices from consecutive slices, following the construction of the straight-skeleton of the active cells of their overlay. The method is presented, and then some arguments are made to ensure that the matching does not contradict the algorithm’s demands (namely, at this stage, no self-intersections).

We reduce the problem of constructing a flow graph FL, which is a structure describing a matching between all the vertices in the scene, to creating local two-dimensional matching graphs on the overlay of a pair of consecutive slices, and then these two-dimensional graphs are lifted to the three-dimensional space, so that FL comprises their union. We now show how to match vertices in an overlay.

2.1 Matching Vertices in Active Cells

Given a set of \( n \) consecutive slices \( L_0, L_1, \ldots, L_{n-1} \), we consider a pair of slices \( \{L_i, L_{i+1}\} \), for which the overlay is computed. The active cells of the overlay are the set

\[
SD(i) = (L_i \setminus L_{i+1}) \cup (L_{i+1} \setminus L_i) = \{A_{i,1}, \ldots, A_{i,m}\},
\]

where \( m \) is the number of cells in the symmetric difference of \( L_i \) and \( L_{i+1} \) (see Figure 1.2). In relation to any overlay, and without loss of generality, we denote by \( L_i \) (resp., \( L_{i+1} \)) the
“lower” (resp., “upper”) slice. With that denomination, we distinguish between $U \setminus L$ cells (cells that are in the “material” zone of the upper slice alone) and similarly $L \setminus U$ cells.\footnote{This distinction is important when dealing with intersection vertices of the overlay (as is seen in this section), and surface orientation (as is seen in Section 2.4).}

The algorithm proceeds with computing the straight skeletons of all the non-empty active cells (empty active cells are the result of overlapping portions of edges in the overlay), by using the algorithm proposed in [14].\footnote{We used this algorithm because of its straightforward approach. However, because of the unique shape of the skeleton faces (see [28]), simple adjustments can make this algorithm work on a precomputed straight skeleton.} We assume that the active cells are in general position (i.e., there are no degeneracies in the skeleton, which can be regarded as multiple regular events by creating different skeletal nodes with zero-length edges connecting between them, and so no special treatment is needed for them). The matching is then performed on each active cell of the overlay, following the steps of the shrinking process creating the skeleton. The motivation for such a matching is that the straight skeleton serves as an indication of a vertex’s proper mates (in the sense of closest neighbors when offsetting inwards) in an active cell, and by following the events of its creation (split and edge events characterized in [14]) the algorithm matches them naturally. The matching algorithm is as follows:

**Input:** An active cell $A_{i,j}$ in $SD(i)$, for which the skeleton $S(A_{i,j})$ is computed.

**Output:** A local matching represented by a graph $M_{ij}$ defined on the vertices of $A_{i,j}$.

**The Algorithm:** A matching is created with every event of the skeleton creation, identified in [14], following these rules:

1. An edge event makes a portion of an active cell edge $e$ vanish. An edge of $M_{ij}$ is created between the two original vertices comprising the portion of $e$ at the point of vanishing. Should an original edge vanish without splitting, the matching edge would be created between the two original endpoints of $e$. Should several edges vanish at the same point, we match all the vertices of these edges in a counter-clockwise order. Figure 2.1 exemplifies this case.

2. A split event occurs between a reflex vertex (so denoted in [14]) and a portion of an edge $e$. An edge of $M_{ij}$ is created between the reflex vertex and the closest neighbor
2.2. PROPERTIES OF THE MATCHING

(by Euclidean distance) out of the two concurrent endpoints of that portion of $e$. Figure 2.1 exemplifies this event as well.

![Figure 2.1: An example of a partial matching following some events in the active cell. The skeleton is shown by solid lines, the progression of cell edges by dashed lines, and the matching by dashed-dotted lines. The matching GH is due to an edge event at $S_3$, while IB is actually two matching edges: a split event at $S_2$ followed by an edge event at $S_5$. KI is due to a split event of $A$ and $I$ at $S_1$, and KA is due to an edge event at $S_4$.](image)

2.2 Properties of the Matching

**Lemma 2.2.1.** $M_{ij}$ is a connected planar graph.

**Proof.** This claim is a direct consequence of the fact that the matching algorithm matches only pairs of vertices that meet during the offset process, or vertices and the concurrent endpoints of edges they split. Since the shrinking algorithm always creates non-intersecting
intermediate polygons [14], the traces of the movement of vertices along the process do not cross, considering that reflex vertices that split an edge can only be matched to other vertices that meet the same edge in the order of splitting (including original endpoints). Then, since a matching edge in $M_{ij}$ is the union of two traces, all edges are noncrossing, and, therefore, $M_{ij}$ is a planar graph.

Assume for contradiction that $M_{ij}$ is not connected, i.e., there is at least one group of polygonal holes which vertices are only matched to other vertices among that group. Since every skeletal node, which is created by an event of the process, creates a matching, the straight skeleton created by the propagation of vertices of this group has to be disconnected from the outer boundary of the active cell, or from the other inner groups. This is impossible, since the straight skeleton of any polygon is a single connected graph. Therefore, $M_{ij}$ is connected.

**Lemma 2.2.2.** The straight segment that connects two matched vertices $p, q$ does not intersect the boundary of their active cell.

**Proof.** Assume for contradiction that the segment $pt + q(1 - t), t \in [0, 1]$ intersects with (at least) two edges $e_1, e_2$ of the boundary of the active cell, coinciding at a concave vertex $r$. The construction of $M_{ij}$ indicates that two vertices are matched if and only if they share the same intermediate polygon in the shrinking process until the event that matches them. Assume $N_{pq}$ is the skeletal node and event that created $m_{pq}$, i.e., without loss of generality, the meeting point of bisector $b_q$ and the edge $e_p$. By definition, this point is equidistant from the two edges $e_p$ and $e_q$ which initially coincide at $p$ and $q$, respectively. In any such configuration, the distances $d_{e_1}, d_{e_2}$ are shorter than at least one of the distances $d_{e_p}, d_{e_q}$, and the propagation of $r$ would eventually occlude $e_p$ from $e_q$. Since the faces of the straight skeleton are monotone, that means that the match $m_{pq}$ is not possible (See Figure 2.2 for an example).

**Theorem 2.2.3.** For any $i, j$, $M_{ij}$ is non-intersecting (thus, $M_{ij}$ is a planar geometric graph).

---

3If all vertices are convex, then the containment of the segment connecting $p$ and $q$ is assured by definition.

4We assume, without loss of generality, that $N_{pq}$ is the first meeting point of these edges, since otherwise they would be neighboring edges in the same intermediate polygon until it vanishes or splits between them, and such matching would not exist. Thus, $e_p, e_q$ are never within the triangle $\Delta_{pqN_{pq}}$. 

2.2. PROPERTIES OF THE MATCHING

Figure 2.2: An example of lemma 2.2.2. The active-cell edges are in solid lines, the bisectors (and the line of sight between $p$ and $q$) are in dashed-dotted lines, and the distances to the edges are in dashed lines. Here, $b_r$ forms an event with $b_q$, because $d_{e_2} < d_{e_p}$, $d_{e_q}$, and so prevents any event of $p$, $r$ that would have created the matching.

Proof. If $A_{i,j}$ is a simple polygon (with no holes), then every matching edge partitions the polygon into two parts. $M_{ij}$ is planar according to Lemma 2.2.1, which also implies that the graph is planar with all matching edges inside the polygon. Thus, when one matching edge, drawn as a straight-line, partitions the polygon (and does not intersect it, as seen by Lemma 2.2.2), there can be no other matching edges from a vertex on one side of the polygon to another in such a partition, since it contradicts the planarity. Therefore, all other matching edges are completely contained in either partition, and can be drawn as straight segments without any intersections of neither matching edges, nor the polygon boundary, and $M_{ij}$ of a simple active cell is a planar geometric graph. (Here, all the faces are simple by definition)

If $A_{i,j}$ contains holes, the proof follows a similar pattern. The case of a matching between two vertices of the outer boundary, is identical to the one shown above. We assume for contradiction that two matching edges $m_{pq}$, $m_{rs}$ cross when they are drawn as straight segments. Without loss of generality, we assume that $m_{pq}$ connects two holes $h_1$, $h_2$. We extend the line $pq$ on both sides to infinity. It is clear that $r$ and $s$ are separated by this line (see Figure 2.2). By Lemmas 2.2.1 and 2.2.2, it is clear that the trace of $m_{rs}$, constructed by the propagation, should intersect the line $pq$ outside the segment $pq$, resulting in $m_{rs}$ occluding, without loss of generality, the hole $h_1$ and part of the outer boundary directly.
to its right, as ordered by the line $pq$, from each other. However, the propagation of the vertices of $h_1$ and those of the outer boundary in the shrinking process then intersect $m_{rs}$, contradicting the construction in Lemma 2.2.1. We conclude that there is no possible trace of matching between $r$ and $s$, and so $m_{rs}$ cannot exist. Therefore, even if $A_{i,j}$ has holes, no two matching edges drawn as straight segments intersect. Moreover, because $M_{ij}$ is connected, all faces of the straight-segment representation of $M_{ij}$ are simple, since they cannot contain disconnected holes.

Naturally, since active cells are disjoint, and all $M_{ij}$ are contained in their respective active cell, the general 2D matching $M_{G_i} = \bigcup_{1 \leq j \leq m} M_{ij}$ of the overlay $SD(i)$ has no crossings (and no intersections as a geometric graph) as well. We will refer to this general graph of the overlay from now on.

**Theorem 2.2.4.** For any $i$, all the faces of $M_{G_i}$ are star-shaped.

**Proof.** In an inward propagation of the edges of such a face $F$, there are no split events, otherwise the hitting vertex would cause a split event in the original polygon as well, and
$F$ would not be a face of $\text{MG}_i$\textsuperscript{5}. Hence, $F$ shrinks to a single point, which is by definition in the halfplanes defined by all its edges (directed inwards). Therefore, this point, and a small neighborhood around it, are part of the kernel of $F$, which is thus non-empty.

\section*{2.3 Lifting Up in Space}

We now show how to create the complete flow graph $\text{FL}(V, E)$ in 3D.

1. The vertices of $\text{FL}$ are created from the original vertices of the contours of all slices, and two new vertices are added per each edge intersection in the overlay, in the respective upper and lower slices. In addition, all original edges of the contours are added (parted by intersection nodes).

2. An edge of $\text{FL}$ is created per each edge of all $\text{MG}_i$. Its connectivity is as follows:

   - If an endpoint of an edge is an original vertex in a slice $L_i$, which is not also an intersection vertex, it remains so in $\text{FL}$.
   
   - When an edge of $\text{MG}_i$ is incident to an intersection vertex in the overlay, corresponding to two vertices (upper and lower), a decision is made as to which of these two it connects. If the edge of $M_i$ lies in a $U \setminus L$ (resp., a $L \setminus U$) cell of the overlay, we connect it to the lower (resp., upper) $\text{FL}$ vertex. Intersection vertices represent places where the surface changes orientation from the viewing direction (the $Z$ axis)—see Figure 2.4. Therefore, this approach allows a gradual change of the surface around these vertices, and prevents foldovers. An intersection point can serve as a meeting point for the lower slice and the contained upper slice, or vice versa, in which case this matching is cogent as well (see Figures 2.4(d) and 2.4(e)).

   - A new edge is created between two vertices of $\text{FL}$ which correspond to the same intersection vertex in the overlay. Only these types of vertices exist in an empty active cell, making up a trivial match between overlapping segments of the symmetric difference.

\textsuperscript{5}In most cases, the faces of $\text{MG}_i$ are convex, but not always, since not all reflex vertices cause split events.
• After applying all of the previous steps, edges that are parallel (having the same endpoints) to other edges in FL (including edges which are parallel to original contour edges) are cast out.

The resultant flow graph contains three types of edges: Original contour edges, edges between two adjacent slices (which we denote as across-edges), and edges from a slice to itself (denoted as self-edges). The self-edges have two subcategories: Down-self-edges (resp., up-self-edges) are self edges created from a matching in which the endpoints belong to the upper (resp., lower) slice in that overlay. This distinction is important when treating the edges as boundary curves, for then two self-edges between the same endpoints would diverge in this subcategory, and create two different curves. FL can be oriented to be a non-intersecting mesh, simply because of the no-intersection property of MG, 0 ≤ i ≤ n − 1, the matching graphs that comprise it.

2.4 Surface Orientation

In this section, we establish an orientation on the flow graph to form the faces of the surface, that will pose as the surface patches in the interpolation. The orientation is fairly simple, resulting from the layered construction of FL from every MG. We orient the sets of edges (and faces) between each pair of consecutive slices independently. The final stitching is simple since the disjoint sets of faces join only along original contours. For such an orientation to take place, we treat each edge of FL as two twin directed half-edges, of opposite directions and with the same endpoints. Then, we proceed as follows:

1. All faces that correspond to faces of MG are oriented CCW (resp., CW) if they reside in an L \ U (resp., a U \ L) cell in the corresponding graph MG.

2. Vertical faces (orthogonal to the XY plane), which translate to segments in the corresponding graph M, are oriented according to the affiliation of the segment in the two-dimensional overlay (U \ L or L \ U).

Note that when orienting an overlay, only one half-edge of each original contour edge is oriented, and the other half is oriented in an adjacent overlay (see Figure 2.5). Therefore,
2.4. **SURFACE ORIENTATION**

The orientation of the different layers results in a complete orientation of the flow graph. The oriented graph is now referred to as the *structural mesh*. This cannot, however, be treated as a regular mesh, since its faces are not necessarily planar, and furthermore, they can be overlapping (because of the inability to distinguish yet between two self-edges of different subcategories and with the same endpoints). A DCEL data structure [11] is used in order to represent the orientation.
Figure 2.4: Matching decisions in intersection vertices. Figure 2.4(a) shows the top view of a pure intersection, that is also shown in isometric views by Figures 2.4(c) to 2.4(b) which demonstrate the twist in the surface. Figures 2.4(d) and 2.4(e) exemplify intersection vertices that are meeting points.
Figure 2.5: Orientation of an overlay of two squares. The faces that are visible are oriented CCW. In addition, the original contours are oriented with only one (opposite) half-edge each.
Chapter 3

Creating a Curve Network

Practically, all surface interpolation techniques begin with the construction of a curve network conforming to a set of geometric constraints. As we seek to create a $G^1$-continuous surface, we begin with constructing a suitable $G^1$-continuous network, based on the acquired structural mesh created in the previous chapters. We proceed in two steps:

1. Extracting first-order data (i.e., the tangent plane) at each vertex.
2. Creating a curve network with the tangent plane constraint.

3.1 Extracting First-Order Data

A common practice in extraction of first-order data from a mesh treats it as a planar mesh and averages the normals of all faces coincident to a vertex in order to approximate its normal vector. However, the flow graph is not a regular mesh. Its faces are not necessarily planar, and are sometimes overlapping. Moreover, it induces a hierarchy of level sets and flow edges between them. We utilize this property to extract first-order data that will serve for creating a curve network which will eliminate this overlapping and fit the natural flow of the surface. We establish two linear-independent tangent directions: the flow of the surface between adjacent slices (the “vertical” direction), and the original contours describing the flow in the input slice (the “horizontal” direction).
3.1.1 Horizontal direction

We interpolate all original contours using periodic cubic splines. For this matter, we consider the original vertices as fixed and the intersection vertices as floating. First, the spline is created for the fixed points: Assuming that the set of original points on a contour is \( \{y_0, y_1, \ldots, y_{n-1}\} \), we seek the tangent directions \( D_i, 0 \leq i \leq n - 1 \), with the periodic spline system:

\[
\begin{pmatrix}
4 & 1 & \ldots & 1 \\
1 & 4 & 1 & \ldots \\
\vdots & \vdots & \ddots & \vdots \\
1 & 4 & 1 \\
\end{pmatrix}
\begin{pmatrix}
D_0 \\
D_1 \\
\vdots \\
D_{n-1} \\
\end{pmatrix}
= 
\begin{pmatrix}
3(y_1 - y_{n-1}) \\
3(y_2 - y_0) \\
\vdots \\
3(y_{n-1} - y_{n-3}) \\
3(y_0 - y_{n-2}) \\
\end{pmatrix}.
\]

The derivatives and the positions of the fixed points determine the piecewise-cubic periodic splines of each contour uniquely. Next, we position the floating vertices on the spline, proportionally to their location on the original edge. Namely, if the original edge \( AB \) can be described as \( A t + B(1 - t), \ t \in [0, 1] \), the cubic spline computed for that edge is \( C_{AB}(t) \), and the intersection vertex lies in position \( t_0 \) on the edge (its floating position is thus \( A t_0 + B(1 - t_0) \)), then its final position is set to \( C_{AB}(t_0) \). This is not always a good practice, since cubic splines are not arclength parameterized, yet complex contours with many vertices produce shorter spline segments to which this positioning applies well.

Then, we compute the horizontal direction \( \vec{H} \) for each vertex (fixed or former floating) from the the derivation of \( C_{AB}(t) \) (for which the control polygon is denoted \( \{P_i | 0 \leq i \leq 3\} \)):

\[
C_{AB}(A) = 3(P_1 - P_0), C_{AB}(B) = 3(P_3 - P_2).
\]

Since cubic splines ensure (at least) \( G^1 \) continuity, both splines coinciding at each vertex agree with the above quantity.
3.1. EXTRACTING FIRST-ORDER DATA

3.1.2 Vertical direction

We find, by a least-square method, the vertical direction that, along with the horizontal direction, creates the tangent plane that fits best the directions acquired from across-edges and self-edges. For an edge between vertices \(A\) and \(B\), where \(A\) is the vertex in mentioning, the derived direction is \(\vec{B} - \vec{A}\). If the edge is a self-edge on slice \(L_i\), we add an element to the \(Z\) component which is \(z_{\text{diff}}^u = z_{i+1} - z_i\) (resp., \(z_{\text{diff}}^d = z_{i-1} - z_i\)) for an up-self-edge (resp., down-self-edge). This addition differentiates between the two types of self-edges connecting identical vertices, and so it defeats the fold-overs in the surface introduced by self-edges in their straight-line form. Self-edges mark places where the object vanishes, so that its boundary is not connected to the previous or next slice, and therefore, this step ensures the creation of an arc that allows its neighboring patch to vanish smoothly. Figure 3.1 shows a case of two self-edges coinciding at the same vertices, which produces two different curves. Figure 3.2 consists of a synthetic example which shows the difference this addition makes in the resulting surface.

Given the tangent plane \(TP(u, v) = \vec{H} u + \vec{V} v\) for the sought variable \(\vec{V}\), and the set of directions \(\vec{M}_i\), \(0 \leq i \leq k\), where \(k\) is the number of directions acquired from the edges of FL, we compute the deviation error of every direction \(\vec{M}_i\) from that plane as \(\langle \vec{M}_i, \vec{N} \rangle\), where \(\vec{N}\) is the unit normal to the plane. The problem of minimizing the error is finding

\[
\arg\min_{\vec{V}, \|\vec{V}\|=1} \sum_{i=0}^{k-1} \langle \vec{H} \times \vec{V}, \vec{M}_i \rangle = \arg\min_{\vec{V}, \|\vec{V}\|=1} \sum_{i=0}^{k-1} \langle \vec{V}, \vec{M}_i \times \vec{H} \rangle,
\]

under the constraint \(\|\vec{V}\| = 1\). Denote \(\vec{G}_i = \vec{M}_i \times \vec{H} = (G_{i,x}, G_{i,y}, G_{i,z})\), and rewrite Equation 3.1 as

\[
\arg\min_{\vec{V}, \|\vec{V}\|=1} \begin{pmatrix}
\sum_{i=0}^{n-1} G_{i,x}^2 & \sum_{i=0}^{n-1} G_{i,x}G_{i,y} & \sum_{i=0}^{n-1} G_{i,x}G_{i,z} \\
\sum_{i=0}^{n-1} G_{i,y}G_{i,x} & \sum_{i=0}^{n-1} G_{i,y}^2 & \sum_{i=0}^{n-1} G_{i,y}G_{i,z} \\
\sum_{i=0}^{n-1} G_{i,z}G_{i,x} & \sum_{i=0}^{n-1} G_{i,z}G_{i,y} & \sum_{i=0}^{n-1} G_{i,z}^2
\end{pmatrix}
\begin{pmatrix}
V_x \\
V_y \\
V_z
\end{pmatrix}.
\]

This problem is solved by finding the smallest eigenvalue of the matrix in Equation 3.2 for which the associated eigenvector is the sought vector \(\vec{V}\). The least-squares method thus
Figure 3.1: The dissimilarity between $L_2$ and its adjacent slices is reflected in the matchings $m_{AB,1}, m_{AB,2}$ which matched vertices $A$ and $B$ in the overlay of slices $L_2$ and $L_3$, and the overlay of slices $L_1$ and $L_2$, respectively. the $z_{\text{diff}}$ component differentiates between these two matches, and produces a smoothly vanishing object on both sides.

produces a tangent plane which minimizes the (squared) deviations of the edges from it.\textsuperscript{1} Figure 3.3 exemplifies this computation.

### 3.2 Creating the curve network

Having tangent planes set at each vertex, the underlying curve network for the surface is created. The network is comprised of cubic Bézier curves, which interpolate the vertices and conform to the tangent plane constraints. The original contours have already been set with the cubic splines, and for the other curves we use the projection method detailed in [26] to establish the tangent vector of a curve $C(t)$ from endpoint $\vec{A}$ (at which the tangent

\textsuperscript{1}Naturally, when there is only one across- or self-edge, the solution is unique, as it happens to be in many cases.
3.2. CREATING THE CURVE NETWORK

vector is sought) with normal $\hat{N}_A$, to $\vec{B}$ with normal $\hat{N}_B$: The chord $\vec{A}\vec{B}$ (if $\vec{A}\vec{B}$ is a self-edge, then with the addition of $z_{\text{diff}}$) is projected onto the appropriate tangent plane, and the tangent is scaled to the chord’s length, as seen in Figure 3.4

$$\vec{T}_A = \left\| \vec{A}\vec{B} \right\| \cdot \frac{\vec{A}\vec{B} - \langle \vec{A}\vec{B}, \hat{N}_A \rangle \hat{N}_A}{\left\| \vec{A}\vec{B} - \langle \vec{A}\vec{B}, \hat{N}_A \rangle \hat{N}_A \right\|}.$$  (3.3)

$\vec{T}_B$ is computed in the same manner. Two endpoints and two tangents suffice to establish a cubic bézier curve with $\{P_0, P_1, P_2, P_3\}$ as a control polygon, since: $P_0 = \vec{A}$, $P_1 = \vec{A} + \frac{1}{3} \vec{T}_A$, $P_2 = \vec{B} + \frac{1}{3} \vec{T}_B$, $P_3 = \vec{B}$. 

CHAPTER 3. CREATING A CURVE NETWORK

(a) Overlay of three slices (slices 1 and 3 are identical in black, and slice 2 in red)

(b) The reconstructed surface. The self edges of the extra contour part in slice 2 caused the surface to vanish smoothly on both sides of the part in slice 2 which is considerably different from the other slices

(c) Another view of the surface

Figure 3.2: An example of the effect of adding a component in the $Z$ direction for up- and down-self edges
Figure 3.3: An example of the calculation of the normal $\hat{N}$ at the vertex $P$. Four directions are acknowledged from matching arcs, and the $z_{\text{diff}}$ component is added to $M_4$, as it is a self edge. The direction $\hat{H}$ is already given from the spline interpolation of the middle contour.
Figure 3.4: Projection of the vector $\vec{AB}$ into the tangent plane at vertex $A$, with normal $\hat{N}_A$. The projected vector $\hat{T}_A$ is then scaled to the length of $AB$, and the result is $\vec{T}_A$. 
Chapter 4

Building Surface Patches

Having the structural mesh and the appropriate curve network, we interpolate them by building surface patches on the faces of the mesh, using the curves as their boundaries. In order for the patches to be $G^1$-continuous, along their common boundaries, we have to define their control points accordingly. We treat three types of faces: quadrilaterals, triangular, and $k$-sided.

4.1 Interpolating Curve Networks

Considerable amount of work has been done on $G^1$-interpolation methods for curve networks. A well-versed survey of these [21] deals with the different popular methods. In Chapter 1, tangent-plane continuity was defined. In general, there are two approaches to ensure continuity of a surface composed of patches:

1. Solving a global system of equations representing the continuity constraints.

2. Creating a common tangent plane field along every boundary joining two patches, and coercing them to follow this constraint.

Clearly, the second approach, albeit not optimal, is more efficient with respect to computation time. However, it poses a typical problem. There exists a limitation in using regular Bézier patches for such an interpolation. Without loss of generality, we assume a quadrilateral patch $S(u, v)$ of order $n$ to which we apply the interpolation over the $S(0, v)$ and the
CHAPTER 4. BUILDING SURFACE PATCHES

\( S(u, 0) \) boundaries. The cross derivatives \( S_u(0, v) \) and \( S_v(u, 0) \) that stem from the tangent plane computation on each boundary determine the next row of control points next to the boundary (i.e., \( P_{ii} \) and \( P_{jj} \), \( 0 \leq i, j \leq n \)). However, this means that the variable \( P_{11} \) is over-constrained, and in general, this system is not guaranteed to have a solution. This is the twist compatibility or vertex consistency problem, so-called because it is equivalent to guaranteeing that \( S_{uv}(0, 0) = S_{vu}(0, 0) \) in this example.

One solution to this problem is using Gregory patches. Gregory patches, introduced in [16], and put to use for surface interpolation in [9], decouple the problem from the two conflicting boundaries.

4.2 Interpolating Quadrilateral and Triangular Gregory Patches

In this section, we detail the specifics of the patch-patch interpolation using the basic Gregory patches.

4.2.1 Quadrilateral Gregory patches

The basic quadrilateral Gregory patch, following the formulation of [9], is a variation of the classic cubic Bézier patch, in which each of the four inner control points is doubled into two control points, and then the couple is linearly-blended into a single point, in order to
4.2. INTERPOLATING QUADRILATERAL AND TRIANGULAR GREGORY PATCHES

fit the Bézier formula:

\[ G(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij}(u, v) \theta_i^3(u) \theta_j^3(v), \]

\[ P_{ij}(u, v) = P_{ij}, \quad i, j = 0, 3, \]

\[ P_{11}(u, v) = \frac{uP_{11u} + vP_{11v}}, \]

\[ P_{21}(u, v) = \frac{(1 - u)P_{21u} + vP_{21v}}{(1 - u) + v}, \]

\[ P_{12}(u, v) = \frac{uP_{12u} + (1 - v)P_{12v}}{u + (1 - v)}, \]

\[ P_{22}(u, v) = \frac{(1 - u)P_{22u} + (1 - v)P_{22v}}{(1 - u) + (1 - v)}. \]  \hspace{1cm} (4.1)

Figure 4.1(a) shows the control structure of a quadrilateral Gregory patch. A cubic Gregory patch allows the definition of the inner control points in terms of each boundary independently, making the twist constraint a blend of two independent twists. As seen in Equation 4.1, the tradeoff is the rational form, which introduces a parametric singularity at the endpoints (e.g., \((0, 0)\)) of the patch, which may render Gregory patches undesirable for some manufacturing uses. However, these patches are geometrically-continuous everywhere, and, thus, they quite fit visualization purposes.
4.2.2 Triangular Gregory patches

The triangular version [19] uses barycentric coordinates, and is an equivalent Gregory variation of the quadratic Bézier triangle (see Figure 4.1(b)):

\[ T(u, v) = \sum_{i+j+k=4 \in \mathbb{N}} P_{ijk}(u, v, w)u^i v^j w^k, \quad u, v, w \in [0, 1], \ u + v + w = 1, \]

\[ P_{ijk}(u, v, w) = P_{ijk}, \ i = 0, j = 0, k = 0, \]

\[ P_{211}(u, v, w) = \frac{v(1-w)P_{211v} + w(1-v)P_{211w}}{v(1-w) + w(1-v)}, \]

\[ P_{121}(u, v, w) = \frac{u(1-w)P_{121u} + w(1-v)P_{121w}}{u(1-w) + w(1-u)}, \]

\[ P_{112}(u, v, w) = \frac{u(1-v)P_{112u} + v(1-u)P_{112v}}{u(1-v) + v(1-u)}. \]

(4.2)
4.2. INTERPOLATING QUADRILATERAL AND TRIANGULAR GREGORY PATCHES

4.2.3 Interpolation along a common boundary

Following the method of [9], we assume a common boundary between two quadrilateral patches $S_1(u, v), S_2(u, v)$, which is, without loss of generality, $S_1(1, v) = S_2(0, v) = B(v)$. We wish to interpolate $S_2(u, v)$, that is, to determine the inner points $Q_{11v}, Q_{12v}$ (see Figure 4.2). We establish a common tangent plane composed of linearly-independent directions:

1. The boundary curve derivative $B_v(v)$.

2. A cross-boundary vector field $C(v)$, defined by the blending of two artificially created vectors $C_0$ and $C_1$ situated at the two common endpoints $Q_{00}$ and $Q_{03}$, respectively: $C(v) = C_0v + C_1(1 - v)$. These vectors could be chosen in any fashion that produces the same vector from both sides of the boundary with opposite signs. For example, we chose them to be $C_0 = N_0 \times (Q_{01} - Q_{00})$ and $C_1 = N_1 \times (Q_{03} - Q_{02})$, which makes the interpolation of both patches independent of each other, since it uses information about the boundary curve alone.

$S_2$ must agree with the tangent plane across the boundary, and so there exist two functions $k(t)$ and $b(t)$ such that

$$\frac{\partial S_2(0, v)}{\partial u} = k(v)C(v) + b(v)\frac{\partial B_v}{\partial v}. \quad (4.3)$$

At the endpoints, the values of $k(v), b(v)$ can be determined by

$$\frac{\partial S_2(0, 0)}{\partial u} = Q_{10} - Q_{00} = k_0C_0 + b_0(Q_{01} - Q_{00}),$$

$$\frac{\partial S_2(0, 1)}{\partial u} = Q_{13} - Q_{03} = k_1C_1 + b_1(Q_{03} - Q_{02}). \quad (4.4)$$

In order for $k(v)$ and $b(v)$ to be uniquely determined, they must be linear functions:

$$k(v) = k_0v + k_1(1 - v),$$

$$b(v) = b_0v + b_1(1 - v). \quad (4.5)$$

1The triangular interpolation is the same, should we use cubic boundaries and then raise their degree to be quadratic.
Having the cross-derivative $\frac{\partial^2 S_2(0,v)}{\partial u \partial v}$ at every point, we can compute the inner row of control points:

$$Q_{11v} - Q_{01} = \frac{1}{3} \left[ (k_0 + k_1)C_0 + k_0C_1 + 2b_0(Q_{02} - Q_{01}) + b_1(Q_{01} - Q_{00}) \right],$$

$$Q_{12v} - Q_{02} = \frac{1}{3} \left[ k_1C_0 + (k_0 + k_1)C_1 + b_0(Q_{03} - Q_{02}) + 2b_1(Q_{02} - Q_{01}) \right].$$  (4.6)

Interpolating in that manner around all of the boundaries of $S_2$ determines all of the inner control points, for either a quadrilateral (such as in this example), or a triangular patch, with three boundary curves and six inner control points.

**Figure 4.2:** Determining inner points across a boundary between two patches

### 4.3 Interpolation with $k$-Sided Patches

Some methods, described in the literature to deal with $k$-sided patches, suggest either to cover the surface with a single patch [20], or offer subdivision schemes [9, 18, 22, 25].
4.3. INTERPOLATION WITH K-SIDED PATCHES

Since the faces of the underlying mesh have a star-shaped embedding (see Theorem 2.2.4), we employ a simple subdivision scheme: We subdivide the patch into convex faces, and then recursively subdivide subpatches by forming a new boundary curve between two vertices of the patch, until all subpatches are either triangular or quadrilateral. The two endpoints chosen in every subdivision are the two non-adjacent closest vertices of that subpatch (see Figure 4.3). In case the two endpoints belong to the same slice, we add the $z_{\text{diff}}$ component as explained in Chapter 3. This subdivision method does not create extra vertices, and so it is appealing. Then, having created quadrilateral and triangular patches, we proceed with the regular interpolation.

![Figure 4.3: Subdividing a 7-sided patch. The subdivision lines are produced in the following order: $v_1 - v_3, v_3 - v_5$ and $v_5 - v_7$, creating three triangular and one quadrilateral patch that are interpolated.](image)
Chapter 5

Results

In this chapter we explore the complexity of the algorithm presented in this dissertation, and the actual running times of the algorithm. We employ the algorithm to several examples.

5.1 Complexity of the Algorithm

Calculating the symmetric difference of two slices is done by a simple line-sweep algorithm which time complexity is $O(n \log n + k)$, where $k$ is the number of intersection points and $n$ is the number of original contour vertices. The value of $k$ can be up to $O(n^2)$ in the worst case, but for most practical cases it is $O(n)$. The bottleneck of the algorithm is the straight-skeleton computation. Although theoretically it can be done in subquadratic time [12], we utilized the algorithm of [14], whose running time is $O(N(r + \log N))$ ($N = \Theta(n + k)$ being the size of the input to the algorithm and $r$ being the number of reflex (concave) vertices, which is $O(N)$). The matching algorithm requires $O(1)$ steps per edge or split event, and, thus, does not add to the total complexity. Subdividing and calculating the network and control points of patches require additional $O(N)$ time. Therefore, the total worst-case time complexity is $O(N^2) = O(n^4)$, yet most cases produce subquadratic behaviour. The space complexity of the algorithm is $O(N)$, since there are $O(N)$ matching edges (every skeletal edge creates $O(1)$ matches), and the total number of faces is $O(N)$ as well.
CHAPTER 5. RESULTS

5.2 Experimental Results

The algorithm was implemented in Visual C++ .NET and run on a 3GHz Athlon 64 processor PC with 1Gb of RAM. The straight-skeleton computation code is courtesy of Petr Felkel. The source consists of about 7,000 lines of code.

5.2.1 Complex examples

We present results obtained by applying our algorithm on several MRI or CT scanned instances: the steps of the reconstruction of a heart in Figure 5.1, and other medical input examples in Figures 5.2 to 5.4. The algorithm was also applied to a topographic map (of the Zikhron-Yaakov area), shown in Figure 5.5. Table 5.1 shows the respective running times of the complex inputs, and Figure 5.6 measures the practical complexity of the algorithm.

<table>
<thead>
<tr>
<th>Input</th>
<th>Heart</th>
<th>Hip bone</th>
<th>Pelvis</th>
<th>Lungs</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of Input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slices</td>
<td>30</td>
<td>34</td>
<td>50</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td>Contours</td>
<td>65</td>
<td>38</td>
<td>108</td>
<td>88</td>
<td>111</td>
</tr>
<tr>
<td>Vertices</td>
<td>1,285</td>
<td>1,706</td>
<td>2,277</td>
<td>3,121</td>
<td>4,546</td>
</tr>
<tr>
<td>Running Times (Seconds)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric difference</td>
<td>0.079</td>
<td>0.062</td>
<td>0.190</td>
<td>0.202</td>
<td>0.237</td>
</tr>
<tr>
<td>Straight-skeleton and matching</td>
<td>0.874</td>
<td>1.118</td>
<td>1.81</td>
<td>4.235</td>
<td>17.857</td>
</tr>
<tr>
<td>Orientation</td>
<td>0.031</td>
<td>0.032</td>
<td>0.047</td>
<td>0.062</td>
<td>0.065</td>
</tr>
<tr>
<td>Curve network</td>
<td>0.156</td>
<td>0.218</td>
<td>0.312</td>
<td>0.453</td>
<td>0.813</td>
</tr>
<tr>
<td>Patches interpolation</td>
<td>0.094</td>
<td>0.108</td>
<td>0.126</td>
<td>0.222</td>
<td>0.230</td>
</tr>
<tr>
<td>Total time</td>
<td>1.234</td>
<td>1.538</td>
<td>2.485</td>
<td>5.174</td>
<td>19.202</td>
</tr>
<tr>
<td>Size of Output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Curves</td>
<td>4,630</td>
<td>5,386</td>
<td>8,805</td>
<td>11,479</td>
<td>14,533</td>
</tr>
<tr>
<td>Patches</td>
<td>2,982</td>
<td>3,485</td>
<td>5,662</td>
<td>7,362</td>
<td>8,971</td>
</tr>
</tbody>
</table>

Table 5.1: Time measurements for several inputs. Excluded are irrelevant visualization methods (such as patch tesselation). Curve network time includes first-order data extraction.

These complex results exhibit a variety of slice set-ups for which the algorithm constructs a smooth surface successfully. Problematic areas which introduce dissimilarities and branching (like the trachea in Figure 5.7 and the major arteries in Figure 5.8) are dealt
with automatically and visually satisfactorily. The examples are of a large scale of input
data for which the measured timing was considerable. The topographic map is unique for
two reasons: There are no intersection vertices, and the amount of vertices per slice is con-
siderably larger than in the medical examples, making the straight-skeleton computation
time substantially dominant, and the overall complexity superquadratic. (The logarithmic
scale in Figure 5.6(b) shows that the measured time complexity is about $O(n^{2.12})$. The
medical examples alone show subquadratic behaviour (about $O(n^{1.6})$, as was measured
by [4])).

5.2.2 Synthetic examples

We tested the algorithm on some synthetic examples in order to address specific issues. A
branching example is demonstrated in Figure 5.9, and a simple example featuring consid-
erable dissimilarities between slices is shown in Figure 5.10. These examples show the
ability of the algorithm to cope with these problematic issues automatically even when the
solution is not trivial. However, there are cases where the algorithm has shortcomings,
which will be discussed in the next chapter. The synthetic examples’ running times were
negligible.
Figure 5.1: A reconstructed heart. 5.1(a) Shows an overlay of two adjacent slices in the input, 5.1(b) shows the curve network, showing splines of original contours (blue), matching curves (green) and subdivision curves (purple). 5.1(c) and 5.1(d) show the final results with shading.
5.2. EXPERIMENTAL RESULTS

(a) Slices

(b) Curve network

(c) Full view

(d) Shaded view

Figure 5.2: A pair of lungs
Figure 5.3: A Pelvis
5.2. EXPERIMENTAL RESULTS

(a) Slices  
(b) Curve network

(c) Full view  
(d) Shaded view

Figure 5.4: A hip bone
Figure 5.5: An example of a reconstructed topographic map
5.2. EXPERIMENTAL RESULTS

(a) Total measured time

\[ y = 1 \times 10^{-10}x^4 - 9 \times 10^{-7}x^3 + 0.0031x^2 - 4.1979x + 3153.3 \]
\[ R^2 = 1 \]

(b) A logarithmic chart of the total measured time

\[ y = 2.1275 \times 3.6275 \]
\[ R^2 = 0.996 \]

Figure 5.6: Timing results for the algorithm
CHAPTER 5. RESULTS

Figure 5.7: The trachea area of the reconstructed lungs

Figure 5.8: The major arteries of the reconstructed heart
5.2. EXPERIMENTAL RESULTS

Figure 5.9: A reconstruction of a synthetic example, featuring a smooth solution to the branching and correspondence issues.
Figure 5.10: Another synthetic example, featuring a smooth solution to a considerable dissimilarity
Chapter 6

Conclusions and Future Work

6.1 Summary

We have shown a method to reconstruct an explicit free-form smooth surface directly from a set of parallel cross-sections, which is robust, creates a visually-pleasing result, and is without any need for intermediate contour interpolation. In addition, since the interpolation is independent of the vertex-matching step, different matching heuristics could benefit from employing this algorithm as well. However, the given $G^1$-continuity does not always satisfy needs of smoothness, especially with geometric dissimilarities which create high curvature areas. By using higher-degree curves and surfaces, one can reconstruct a surface with $G^2$-continuity for a better result, albeit further complications. The structure of the flow graph allows further extraction of geometric quantities (e.g., curvature) in a similar manner, avoiding some of the complications introduced in general mesh surface fitting.

6.2 Shortcomings of the Algorithm

Although this algorithm handles dissimilarities well, it is at its best when the active cells of an overlay are small in area with respect to the original contours, and so it might benefit greatly from preprocessing the input contours with proper transformations. When the area of these cells is substantial, the reconstruction might not seem intuitive enough (see Figure 6.1). In addition, the matching can create long “skinny” patches that are a shortcoming
in the interpolation of a smooth shape, because they can create undesirable areas of high curvature (see Figure 6.2). The solution to this is either to further constrain the interpolation with full or partial $C^2$ continuity terms (which is an optimal solution), or to try to unify nearby triangles in order to create patches which are more proportionate.

Figure 6.1: The result of the reconstruction of two shapes with a small intersection area. Even though the shapes are quite similar, the given overlay does not produce the best result possible.
6.2. **SHORTCOMINGS OF THE ALGORITHM**

Figure 6.2: The matching produces long skinny triangles, and the result shows unwanted “ripples”
Bibliography


אינטפרטציה לא ליניארית בין חתכים

חיבור על מחקר

לייבור על מחקר

לשם דרגת תואר של תואר ראשון לרבנות הפרטים

מגייסר למדעי במדעי המחשב

אמיר וקסמן

ודגש.bridge רציניות — מרכז מחקרים למידה

נתיב יד - מסלול, התשס"ט

נובמבר 2006
תודה

הזה מחקר וنعשתה הזה הדריכה של פרופ' ג'יל ברקה בפקולטה
למדעי המחשב.

אני מצהיר שלטתחום על התמיכות הכפיפה הנדרשת בהשחלה.

תודהlein

Technion - Computer Science Department - M.Sc. Thesis MSC-2006-29 - 2006
תוכן עניינים

תודות iii

1 הקציר באנגלית

3 מבוא א. מחינים 1.1

4 תמר רך 1.2

4.1 אגסיוריה דיפרגנולית בקירוב שלعقدות ומישוטים 1.2.1

8 שלדים ישרים 1.2.2

9 סקירת השיטה שלנו 1.3

10 הגרדרת טרנייה 1.3.1

11 הגרדרת טרנייה 1.3.2

11.1 שבירות קדומות 1.3.3

11.2 תמלאת אולפיים 1.3.4

12 מבנה ההיבורה 1.4

2 תואמת מכוסת-שלד-ישר 2

15 התואמת קדוקדיה בתאים פיליים 2.1

17 התואמת את התואמת 2.2

21 הגרדרת ל‑ vậnח 2.3

22 אオリוג年底前 המשט 2.4

27 יצירה של קדוקדיה 3

27.1 מיפוי גרגורי מסדר ראשוני 3.1

28.1.1 בתווך האופק פ 3.1.1

29 חיבור האנגלי 3.1.2

30 יצירה של תואמת 3.2

35 בניית התלךית פשתנה 4

35.1 אנטרפליציה של רשת קדוקדיה 4.1

36 אנטרפליציה של תלקוח גרגורי מרחבים ומישולות 4.2

36.1 תלקוח גרגורי מרחב 4.2.1

38 תלקוח גרגורי משולש 4.2.2

39 אנטריפלייז של גבלי מישוק 4.2.3

40 אנטריפלייז של תלקוח מרחב צלעות 4.3
53. תוצאות סיבוכיות האלגוריתם 5.1
53. תוצאות נסיוניות 5.2
53. תוצאות מורכבות 5.2.1
53. תוצאות סינתטיות 5.2.2

55. מסקנות ועבודה עתידית 6.1
55. סיכום 6.2
55. סיכום האלגוריתם 6.2

59. ספרות
רשימת טבלאות

.............................................................. תוצאות מדידות הזמנים לקלטים שונים

5.1 הוצאת מודדות הופכים לקלטים שונם
רשימה אורחית

9. דוגמה של פוליגון והשלד השריר של...
13. דוגמה של חומרים מחשבים משני חתכים...
17. דוגמה של חלון של התאמה כחובקי אורח ארועים בחה הפיטיל...
19. דוגמה של לימה...
20. דוגמה של פוליגון עם חורים...
24. בחירויות התאמה בצמות החריכה...
25. אוריינטציה של מנה של שני ריבועים...
30. דוגמה של התאמה ריבוע מגה בחיתוך...
32. דוגמה של והפסת ריבוע בכיוון Z לקשתה בעברית...
33. דוגמה של החישוב הנורמל בנקודה...
38. דוגמה של חישוב המרומ...
40. קיבולת נקודה בכיוון פונומיא ממזויי של גבול בך של חלון...
41. חלון של חלון בצורת 7 צדדים...
46. שלחון של הברל...
47. שלחון של ריאגורית...
A summary of the contents of the page:

1. The thesis focuses on the study and description of an object that is a subject of research for many years, which is a representation of a volume composed of simple polygons. The volume is comprised of only a few cuts.

2. A suitable interpolation solution can be used between the cuts of corners. The object is defined by a set of points that are defined on a 3D object.

3. The volume is defined by the cuts of corners, and the problem is described as follows:

\[ n \]

\[ \{ L_0, L_1, \ldots, L_{n-1} \} \]

where \( i \) is the number of the cut, \( z_i \) is the depth of the cut, \( (x, y, z) \) is the corner of the cut, and \( S(x, y, z) = L_i \) is the cut of the object defined by the problem.

4. Therefore, the problem is not well-defined.

5. Maximal solutions are required to be determined, but there are infinitely many possible solutions for each problem.

6. The problem of more than two cuts in each point of the set and the solution suggested in this work is a method of triangulation, which is achieved by interpolating the values of the same mapping between the cuts and the plane and by using geometric derivatives of the same mapping.

7. The shape of the cut is linear, and the surface can be created using linear parts. The method of interpolation works for all parts with linear surface parts.

8. The cut of the surface passes through all the corners that are cut, and therefore it is well-defined, closed, and satisfies the following conditions:

- The cut of the surface in the plane is a cut of the object.
- The surface passes through all the corners that are cut.
- The surface does not intersect itself.
- Therefore, it has a well-defined orientation.

The summary shows a summary of the summary and the conclusion of the paper.
The algorithm must be robust, and in addition, it must be able to handle complex inputs. It must automatically deal with holes and in all cases of intersections and hierarchical structures, of any number of polygons. The result must be pleasant to the eye as much as possible, also, our algorithm presents a robust matching between the original polygons' vertices (in overlays) and overlay vertices, based on the calculation of each two adjacent cuts, as defined in [4].

Thus, we can use this algorithm for any array of polygons, and therefore the algorithm can create a smooth surface even if there are changes, depending on each cut. These algorithms suggested linear returns on consecutive works, such as linearization, or前锋, or partial linearization, or other relevant works. The algorithm is based on the calculation of the direct integral of the straight line of the cut, of which we can find information of the first order, and, at the same time, it is suitable for partially curved surfaces, which are divided into sections.

At the same time, we can derive a matching between the data and the algorithm itself. However, some works mentioned restrictions on this without suggesting a solution to the problem. The partial solution is used to divide the work. The mesh of the polygon is divided according to the original work, and we get an interface that fits the work. The algorithm itself presents the interface in all cases of intersections and hierarchical structures, of any number of polygons.

The algorithm itself suggests using the algorithm calculation of the straight line cut, as defined in [14].

Works that mention the method of cutting both the workstation and the workstation, the edges of the workstation and the workstation, and the workstation.
In the framework of the research, we developed a system for interpolation implemented in VC.NET. The input of the system is a set of polygons of the same structure as the model's vertices, and the output is a set of parts of the surface that form the complete surface of the object \( O \). The system allows for a complete display of the surface using the graphics hardware through OpenGL.

As examples of the system in action, we present several results obtained in the implementation of 5.1, and in the figure, the structure of organs, lungs and other organs 5.2, a topographic map of the area of Zarche 5.3. We check the algorithm on these results, and also we check the interpolation algorithm and its performance with synthetic data that are used to verify the system.

In section 1, we present a general introduction to this research and a review of previous works. Section 2 describes and describes the process, its properties are studied, and the matching algorithm and graph flow orientation algorithm are described in section 3. The final section 5.5 we summarize the results of the algorithm implementation and measurements made on them. We examine the thesis and discuss the problems and drawbacks of the algorithm in section 6.