TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

D I P L O M A T H E S I S

Reconstruction of rigid body motions generating 3D geometric texture

Institute of
Discrete Mathematics and Geometry
Vienna University of Technology

supervised by o.Univ-Prof. Dr. Helmut Pottmann

Author:
Murat Arikan
Mengergasse 40 / TOP 24, 1210 Wien

Date Signature

Abstract

There are many sets of transformations, which generate geometric patterns of repeated structures in 3D mesh-based models. Discovery of such patterns generated by commutative one- and two parameter subgroups of the group of similarity transformations has already been handled by other authors. We propose a method, which considers a more general case of geometric patterns generated by one parameter rigid body motions. Our method for the reconstruction of rigid body motions generating geometric patterns is not fully automatic and requires some user interaction to select one of the repeated structures in the model at the moment. Discovery of the others is made possible by using known registration and interpolation algorithms. As an application of our method we use the reconstructed rigid body motion to generate 3D geometric texture.

Kurzfassung

Es gibt viele Mengen von Transformationen, die in 3D netzbasierten Modellen geometrische Muster von wiederholenden Strukturen generieren. Die Entdeckung von solchen Mustern, die von kommutativen ein- und zweiparametrigen Untergruppen der Gruppe der Ähnlichkeitsabbildungen generiert werden, wurde bereits von anderen Authoren behandelt. Wir schlagen eine Methode vor, die einen allgemeineren Fall von geometrischen Mustern, die von einparametrigen Starrkörpertransformationen generiert werden, berücksichtigt. Unsere Methode zur Rekonstruktion von geometrische Muster generierenden Starrkörperbewegungen ist zurzeit nicht vollautomatisch und be- nötigt etwas Benutzerinteraktion zum Selektieren von einer der wiederholenden Strukturen im Modell. Die Entdeckung von anderen wird durch die Verwendung von bekannten Registrierungs- und Interpolationsalgorithmen möglich. Als eine Anwendung unserer Methode verwenden wir die rekonstruierte Starrkörperbewegung zum Generieren von 3D geometrischen Texturen.

Acknowledgements

First and foremost I would like to express my gratitude to my advisor Professor Helmut Pottmann, who gave me the chance to work at the institute. It has been a great pleasure to write this thesis under his guidance and with his helpful comments. I thank all the members of the Institute of Geometric Modeling and Industrial Geometry, especially Simon Flöry for allowing me to use some of his code, Heinz Schmiedhofer for modeling most of the test meshes, Alexander Schiftner and Niloy Mitra for patiently answering my questions.

Special thanks goes to Can Turgay for modeling some of the test meshes and helping me for the creation of the figures in this thesis.

Many thanks to my family for enabling me to study in Austria. Further thanks goes to Christa and Hans Figar for supporting me during my studies and being my second family. My sincerest thanks belong to my wife Karin, who is and has always been there when I needed her.

Contents

1 Introduction

Given a 3D geometric model, one wants to detect repeated geometric structures in the model, which have been extensively studied in [21]. Pauly et al. [21] introduce an algorithm for discovering geometric patterns generated by commutative one- and two-parameter subgroups of the group of similarity transformations and concludes that their work is limited with the commutative case. In this thesis, we contribute to their work by considering a more general case of geometric patterns generated by one parameter rigid body motions. This thesis is structured as follows:

- Chapter 2 introduces some basic concepts from differential geometry, which will be needed throughout our algorithm and the algorithm proposed in [21].
- Chapter 3 gives a short introduction to graph theory, focussing on what will be needed in our method.
- Chapter 4 gives a short introduction to quaternions and describes how rotations can be represented by unit quaternions. There are several notations that can be used to represent rotations. This chapter also addresses the conversion between different notations of rotations. Quaternion representation of rotations is used in the method of Horn [15] for the registration of two point sets by given correspondences.
- Chapter 5 describes some registration algorithms with known and unknown correspondences, focussing on which are implemented in the accompanying software.
- Chapter 6 describes the algorithm for the discovery of geometric patterns generated by commutative one- and two-parameter subgroups of the group of similarity transformations as proposed in [21].
- Chapter 7 presents our method for the reconstruction of rigid body motions generating patterns of repeated structures in 3D geometric models.
- Chapter 8 shows how to use the reconstructed rigid body motion to generate 3D geometric texture and discusses some limitations of our method that were found to be of interest for further investigation.

2 Differential Geometry

We will give some basic definitons from differential geometry. Our goal is to introduce the notion of principal curvatures $\mathbf{k}_1, \mathbf{k}_2$ and the principal directions $\boldsymbol{e}_1, \boldsymbol{e}_2$. Further detail on differential geometry can be found in [18].

2.1 Regular Curves

Definition 2.1. *A parametric differentiable curve is a differentiable function $\alpha : I \to \mathbb{R}^3$ of the open interval $I = (a, b) \subseteq \mathbb{R}$ in $\mathbb{R}^3$.*

The differentiability in the above definition means that the function α maps each $t \in I$ onto a point $\alpha(t) = (x(t), y(t), z(t)) \in \mathbb{R}^3$ such that the functions $x(t), y(t), z(t)$ of the real variable t are differentiable.

Definition 2.2. *A parametric differentiable curve $\alpha : I \to \mathbb{R}^3$ is regular, if $\alpha'(t) \neq 0$ for all $t \in I$.*

In the following we consider only regular curves.

2.1.1 Frenet Formulas

Let $\alpha : I = (a, b) \to \mathbb{R}^3$ denote a curve[1] parametrized by its arc length s.

Definition 2.3. *The magnitude of $\alpha''(s)$*

$$\mathbf{k}(s) = |\alpha''(s)|$$

is called the curvature of α at the parameter value s.

Because of the arc length parametrization, the tangent vector $\alpha'(s) = t(s)$ has unit length at each point:

$$|\alpha'(s)| \equiv 1.$$

If we differentiate the identitiy

$$\alpha'(s) \cdot \alpha'(s) \equiv 1$$

with respect to s, we get another identity

$$\alpha''(s) \cdot \alpha'(s) \equiv 0.$$

[1] In the following the phrase curve will always refer to a regular parametric differentiable curve.

Therefore $\alpha''(s) \equiv \mathbf{0}$ (i.e. the curve is a straight line) or $\alpha''(s)$ is normal to the tangent vector. At every point of the curve for which $\mathbf{k}(s) \neq 0$ there is a well-defined unit vector

$$n(s) = \alpha''(s)/\mathbf{k}(s)$$

called the principal normal.

We consider only curves, which don't have any singular point of order one (i.e. a point with $\mathbf{k}(s) = 0$).

The plane spanned by the unit tangent vector $t(s)$ and the unit principal normal vector $n(s)$ is called the osculating plane at $\alpha(s)$.

The unit vector

$$b(s) = t(s) \times n(s)$$

is perpendicular to the osculating plane at $\alpha(s)$ and is called the binormal vector. Differentiating the identitiy

$$b(s) \cdot b(s) \equiv 1$$

with respect to s gives us

$$b'(s) \cdot b(s) \equiv 0,$$

which implies $b'(s) = \mathbf{0}$ (i.e. α is a plane curve) or $b'(s)$ is normal to $b(s)$. $b'(s)$ is also normal to $t(s)$, since the following holds:

$$b'(s) = t'(s) \times n(s) + t(s) \times n'(s) = t(s) \times n'(s).$$

Thus $b'(s)$ must be parallel to $n(s)$ and it can be written as:

$$b'(s) = \tau(s)n(s)$$

with a function $\tau(s)$.

Definition 2.4. *The number $\tau(s)$ defined by*

$$b'(s) = \tau(s)n(s)$$

is called the torsion of the curve α at the parameter value s.

In order to define the Frenet formulas, the vector $n'(s)$ has to be computed. Since $n(s) = b(s) \times t(s)$, we get

$$\begin{aligned} n'(s) = b'(s) \times t(s) + b(s) \times t'(s) &= \\ \tau(s)(n(s) \times t(s)) + \mathbf{k}(s)(b(s) \times n(s)) &= \\ -\tau(s)b(s) - \mathbf{k}(s)t(s). & \end{aligned}$$

The Frenet formulas are:

$$\begin{aligned} t' &= \mathbf{k}n \\ n' &= -\mathbf{k}t - \tau b \\ b' &= \tau n \end{aligned}$$

Remark 2.5. *In the special case of a plane curve $\alpha : I \to \mathbb{R}^2$ the curvature $\mathbf{k}(s)$ can also be assigned a negative value. For this purpose the normal vector $n(s), s \in I$ is defined in such a way that the basis $\{t(s), n(s)\}$ has the same orientation as the canonical basis of $\mathbb{R}^2$. Then the curvature $\mathbf{k}(s)$ is defined as follows:*

$$\frac{dt}{ds} = \mathbf{k}n$$

and can be both, positiv and negativ.

2.2 Regular Surfaces

In this section a regular surface shall be denoted by S. Each point $p \in S$ of a regular surface has a neighbourhood $\bar{V}$ in $\mathbb{R}^3$ such that $V = \bar{V} \cap S \subset \mathbb{R}^3$ has a parametric representation as follows:

$$x : U \subset \mathbb{R}^2 \to V \subset \mathbb{R}^3 : (u, v) \in U \to x(u, v) \in S. \tag{2.1}$$

For a definition of a regular surface we refer the reader to [18]. It can be shown that geometric objects and properties like tangent space, normal vector, curvature, etc., which are defined in terms of the parametrization (2.1) are independent of a specific parametrization of the regular surface S, but dependent only of the regular surface S itself.

2.2.1 Differentiable Functions on Regular Surfaces

Definition 2.6. *Let $f : V \subset S \to \mathbb{R}$ be a function defined on an open subset V of the regular surface S. f is called differentiable at $p \in V$, if for a parametrization $x : U \subset \mathbb{R}^2 \to S$ with $p \in x(U) \subset V$ the concatenated function $f \circ x : U \subset \mathbb{R}^2 \to \mathbb{R}$ is differentiable at $x^{-1}(p)$. f is called differentiable, if it is differentiable at all $p \in V$.*

This definition of differentiability can easily be extended to functions mapping from a regular surface S_1 onto a regular surface S_2. The function $\psi : V_1 \subset S_1 \to S_2$ from an open subset V_1 of the regular surface S_1 onto the regular surface S_2 is called differentiable at $p \in V_1$, if for the parametrizations

$$x_1 : U_1 \subset \mathbb{R}^2 \to S_1, \quad x_2 : U_2 \subset \mathbb{R}^2 \to S_2,$$

with $p \in x_1(U_1)$ and $\psi(x_1(U_1)) \subset x_2(U_2)$ the function

$$x_2^{-1} \circ \psi \circ x_1 : U_1 \to U_2$$

is differentiable at $q = x_1^{-1}(p)$.

2.2.2 Tangent Space

The partial derivatives of the parameterisation x shall be denoted by $\partial x/\partial u = x_u$ and $\partial x/\partial v = x_v$, their evaluation at a surface point $p = x(q), q \in U$ by $x_u(q), x_v(q)$. The vectors $x_u(q), x_v(q)$ span the tangent space $T_p(S)$ of the regular surface S at a given surface point p. The choice of the parametrization x determines a basis $\{x_u(q), x_v(q)\}$ of $T_p(S)$, which is called an associated basis to x. The regularity of the surface S ensures the existence of the tangent space $T_p(S)$ at every point $p \in S$.

Each tangent vector $w \in T_p(S)$ is the tangent vector $\alpha'(0)$ of a curve $\alpha = x \circ \beta$, where $\beta : (-\epsilon, \epsilon) \to \bar{U}$ is defined by $\beta(t) = (u(t), v(t))$ with $\beta(0) = q = x^{-1}(p)$. It holds that

$$\begin{aligned} \alpha'(0) &= \frac{d}{dt}(x \circ \beta)(0) = \frac{d}{dt}x(u(t), v(t))(0) \\ &= x_u(q)u'(0) + x_v(q)v'(0) = w. \end{aligned}$$

Thus the tangent vector w has the coordinates $(u'(0), v'(0))$ in the associated basis $\{x_u(q), x_v(q)\}$.

2.2.3 First Fundamental Form

By restricting the Euclidean scalar product to each tangent space $T_p(S), p \in S$, a scalar product can be defined on each of the tangent spaces, which will be denoted by $\langle ., . \rangle_p$. The quadratic form given by

$$I_p(w) = \langle w, w \rangle_p = |w|^2 \geq 0$$

is the first fundamental form of the regular surface S at $p \in S$.

Since the tangent vector $w \in T_p(S)$ is the tangent vector $\alpha'(0)$ of the curve $\alpha(t) = x(u(t), v(t)), t \in (-\epsilon, \epsilon)$ with $\alpha(0) = p$, the following holds:

$$\begin{aligned} I_p(\alpha'(0)) &= \langle \alpha'(0), \alpha'(0) \rangle_p \\ &= \langle x_u(q)u'(0) + x_v(q)v'(0), x_u(q)u'(0) + x_v(q)v'(0) \rangle_p \\ &= E(u'(0))^2 + 2Fu'(0)v'(0) + G(v'(0))^2, \end{aligned}$$

where

$$\begin{aligned} E &= \langle x_u(q), x_u(q) \rangle_p \\ F &= \langle x_u(q), x_v(q) \rangle_p \\ G &= \langle x_v(q), x_v(q) \rangle_p \end{aligned}$$

are the coefficients of the first fundamental form in the associated basis $\{x_u(q), x_v(q)\}$ of the tangent space $T_p(S)$.

2.2.4 Unit Normal Vector Field and Orientation

Given a parametrization $x : U \subset \mathbb{R}^2 \to V \subset S$ of the open subset V of the regular surface S, the unit normal vector at the surface point $p = x(q), q \in U$ is defined by

$$N_p = \frac{x_u(q) \times x_v(q)}{|x_u(q) \times x_v(q)|}.$$

The orientation of the associated basis $\{x_u(q), x_v(q)\}$ determines the sign of the vector N_p. The differentiable mapping $N : x(U) \to \mathbb{R}^3$ assigns each $p \in x(U)$ a unit normal vector N_p. If the regular surface S cannot be covered with a single parametrization, it might be that the mapping N (also called the unit normal vector field) cannot be extended differentiably to all points of the surface S. A standard example for this case is the Möbius strip.

A regular surface S is called orientable, if a globally differentiable unit normal vector field can be defined on S. The choice of such a vector field is called the orientation of the surface S. There are two possibilities to choose from. An orientable surface with a chosen orientation is called oriented.

Definition 2.7. *The Gaussian map N of the oriented regular surface S is defined by*

$$N : S \to S^2 : p \to N_p,$$

where S^2 denotes the unit sphere.

2.3 Second Fundamental Form and Normal Curvature

The differential dN_p of the Gaussian map at $p \in S$ is a linear function that maps from $T_p(S)$ to $T_{N_p}(S^2)$. Since the planes $T_p(S)$ and $T_{N_p}(S^2)$ are parallel, dN_p can be considered as a linear mapping of $T_p(S)$ onto itself.

For a given curve $\alpha(t)$ on the regular surface S with $\alpha(0) = p$, the linear mapping $dN_p : T_p(S) \to T_p(S)$ is defined by

$$N'(0) = dN_p(\alpha'(0)),$$

where the curve $N \circ \alpha(t) = N(t)$ on the unit sphere S^2 is the restriction of the unit normal vector field to the curve $\alpha(t)$.

Proposition 2.8. *The differential $dN_p : T_p(S) \to T_p(S)$ of the Gaussian map is a self-adjoint linear mapping (see [?]).*

A theorem in linear algebra tells us that, dN_p being self-adjoint, has an orthonormal basis of eigenvectors $\boldsymbol{e}_1, \boldsymbol{e}_2 \in T_p(S)$ with corresponding eigenvalues $-\mathbf{k}_1, -\mathbf{k}_2$. Without loss of generality we assume $\mathbf{k}_1 \leq \mathbf{k}_2$.

Definition 2.9. *(Second Fundamental Form) The quadratic form*

$$\Pi_p : T_p(S) \to \mathbb{R} : \Pi_p(v) = -\langle dN_p(v), v \rangle$$

is called the second fundamental form.

Definition 2.10. *(Normal Curvature) Let α be a regular curve on S parametrized by its arc length and passing through $p(=\alpha(0))$. The curvature of α at $p \in S$ is denoted by k_p and its unit normal vector by n_p. The normal curvature of α at p is defined by $k_n^p = k_p\langle n_p, N_p\rangle$.*

Remark 2.11. *The sign of the normal curvature k_n^p depends on the orientation of S.*

Proposition 2.12. *Let α be a regular curve as above. Then*

$$k_n^p = \Pi_p(\alpha'(0)).$$

Proof. We differentiate the equation

$$\langle N(s), \alpha'(s)\rangle = 0$$

with respect to the arc length s and get

$$\langle N(s), \alpha''(s)\rangle = -\langle N'(s), \alpha'(s)\rangle.$$

Thus it follows that

$$\begin{aligned}\Pi_p(\alpha'(0)) &= -\langle dN_p(\alpha'(0)), \alpha'(0)\rangle \\ &= -\langle N'(0), \alpha'(0)\rangle \\ &= \langle N(0), \alpha''(0)\rangle \\ &= \langle N_p, k_p n_p\rangle = k_n^p.\end{aligned}$$

□

It follows that the normal curvature depends only on the unit tangent vector, which in turn implies the following proposition of Meusnier.

Proposition 2.13. *(Meusnier) All curves on S passing through a given point $p \in S$ and having the same unit tangent vector at p also have the same normal curvature at p.*

This allows us to speak of the normal curvature at p along a unit tangent vector $v \in T_p(S)$.

Theorem 2.14. *(Euler's Theorem) The normal curvature at p along the unit tangent vector $v \in T_p(S)$ has the form*

$$k_n^p = \mathbf{k}_1 \cos^2\theta + \mathbf{k}_2 \sin^2\theta.$$

It follows that $\mathbf{k}_1$ and $\mathbf{k}_2$ are the extremal values of the normal curvature at $p \in S$.

Proof. Since $\boldsymbol{e}_1, \boldsymbol{e}_2$ form an orthonormal basis of the tangent space $T_p(S)$, we may write

$$v = \langle v, \boldsymbol{e}_1 \rangle \boldsymbol{e}_1 + \langle v, \boldsymbol{e}_2 \rangle \boldsymbol{e}_2 = \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta.$$

The normal curvature along v is given by

$$\begin{aligned} k_n^p &= \Pi_p(v) = -\langle dN_p(v), v \rangle \\ &= -\langle dN_p(\boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta), \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta \rangle \\ &= \langle \boldsymbol{e}_1 \mathbf{k}_1 \cos\theta + \boldsymbol{e}_2 \mathbf{k}_2 \sin\theta, \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta \rangle \\ &= \mathbf{k}_1 \cos^2\theta + \mathbf{k}_2 \sin^2\theta. \end{aligned}$$

□

Definition 2.15. *(Principal Curvatures)* $\mathbf{k}_1$ *and* $\mathbf{k}_2$ *are called the principal curvatures and the corresponding directions given by the vectors* $\boldsymbol{e}_1$ *and* $\boldsymbol{e}_2$ *are called principal directions.*

Definition 2.16. *(Gaussian Curvature) The determinant of the differential* $dN_p : T_p(S) \to T_p(S)$ *of the Gaussian map at* $p \in S$ *is the Gaussian curvature:* $K = det(dN_p) = (-\mathbf{k}_1)(-\mathbf{k}_2) = \mathbf{k}_1\mathbf{k}_2$.

Definition 2.17. *(Mean Curvature) The mean curvature at* $p \in S$ *is the average of the principal curvatures:*

$$H = \frac{\mathbf{k}_1 + \mathbf{k}_2}{2}.$$

Definition 2.18. *(Umbilical Point) A point* $p \in S$ *with* $\mathbf{k}_1 = \mathbf{k}_2$ *is called umbilical point.*

3 Graph Theory

3.1 Basics from Graph Theory

This section gives a brief summary of basic definitions from graph theory. The focus will be on simple graphs. A simple graph is an undirected graph containing no loops or multiple edges.

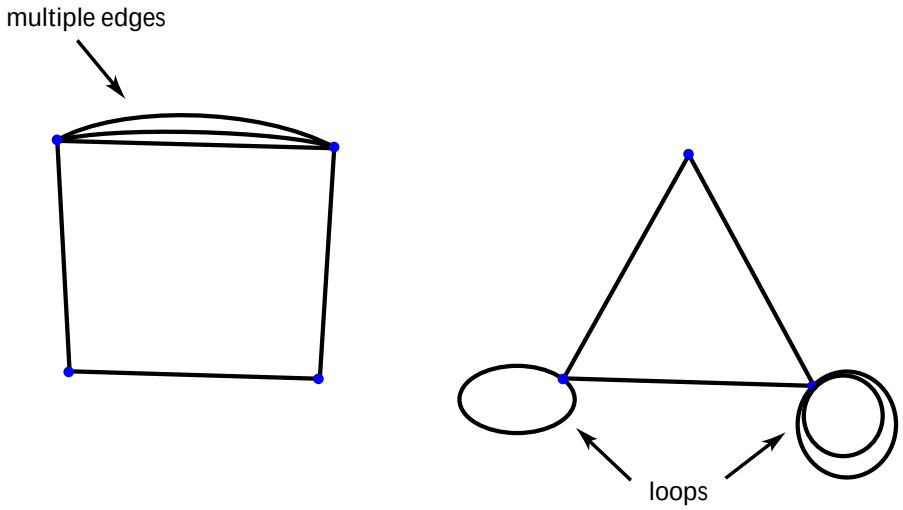

Figure 3.1: Multiple edges and loops

A graph is a pair $G = (V, E)$, $V = \{v_1, v_2, \cdots, v_n\}$ denoting the set of vertices and $E \subseteq V^2$ the set of edges (v_i, v_j). In the following, we shortly use the term graph for a simple graph. A vertex $v \in V$ is incident with an edge $e \in E$, if there exists a vertex $v_i \in V$ with $e = (v, v_i) \in E$. The degree, also called valence of a vertex v is the number of edges incident to the vertex v. Two vertices v_i, v_j of G are adjacent, if $e = (v_i, v_j) \in E$. If all vertices of the graph G are pairwise adjacent, then G is called complete (see Figure 3.2 for an example of a complete graph). A path between two vertices x_0 and x_k is a non-empty[1] graph $G_P = (V_P, E_P)$ of the form

$$V_P = \{x_0, x_1, \cdots, x_k\}, \quad E_P = \{(x_0, x_1), (x_1, x_2), \cdots, (x_{k-1}, x_k)\},$$

[1] An empty graph is a pair $G = (V, E)$ with empty sets V and E.

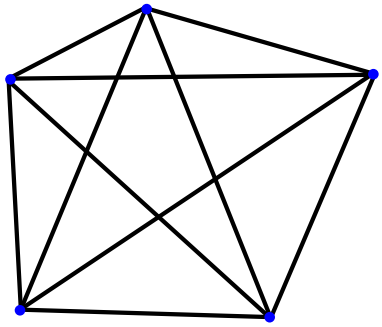

Figure 3.2: Complete graph

where the vertices as well as the edges are pairwise different. The vertices x_0 and x_k are called end vertices of the path. The vertices $x_1, x_2, \cdots, x_{k-1}$ are the inner vertices of G_P. The number of edges of a path is its length. If $x_0 = x_k$, the path is called a cycle. The graph $G = (V, E)$ is connected, if for any two vertices v_i, v_j with $v_i \neq v_j$ there exists a path connecting them. A graph G is called a forest, if G does not contain any cycles. A connected forest is called a tree. Sometimes it's convenient to consider one vertex of a tree as the root of this tree. A tree with a fixed root r is called a rooted tree. The leaves[2] of a tree are the vertices of degree one. The depth of a vertex v in a rooted tree with root r is the length of the path from r to v. Two graphs $G = (V, E)$ and $G' = (V', E')$ are given. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a subgraph of G, written as $G' \subseteq G$. The spanning tree $T = (V_T, E_T)$ of a connected graph $G = (V, E)$ is a subgraph $T \subseteq G$ with $V_T = V$ and $E_T \subseteq E$. A connected graph G can have different spanning trees. We can assign a weight to each spanning tree by computing the sum of the weights[3] of the edges in that tree. The minimum spanning tree of a connected graph G has the minimum weight among all spanning trees of G.

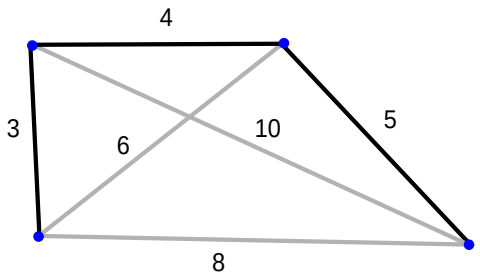

Figure 3.3: An example of the euclidean minimum spanning tree of a complete graph with four vertices

[2]Even if the root of a tree has degree one, it's never called a leaf.
[3]If we speak of a weighted edge $e = (v_i, v_j)$, we mean the euclidean distance of its vertices.

3.2 Algorithms on Simple Graphs

This section introduces some basic algorithms on simple graphs, which are used in our method.

Algorithm 3.1. *(Kruskal's Algorithm[4]) An algorithm that finds the minimum spanning tree of a connected weighted graph $G = (V, E)$.*

- *create a forest of size $|V|$, where each vertex $v \in V$ is a separate tree*
- *define an empty set $S = \emptyset$*
- *Sort the edges $e \in E$ according to their weights: $w_{e_1} \leq w_{e_2} \leq \cdots \leq w_{e_{|E|}}$*
- *for $i = 1, \cdots, |E|$*
  - *if $S \cup \{e_i\}$ does not contain any cycles, add that edge to the forest: $S = S \cup \{e_i\}$*
  - *otherwise discard that edge*

The proof of the correctness of the algorithm consists of two parts. In the first part it's proved that the algorithm produces a spanning tree. It is trivial to see that the produced subgraph H cannot contain any cycles. H must also be connected, since the first encountered edge that connects two components of H would have been added by the algorithm. Secondly, it's proved that the produced spanning tree has the minimum weight among all spanning trees. For the proof of the minimality see [29].

Another algorithm that finds the minimum spanning tree of a connected weighted graph $G = (V, E)$ is the Prim's algorithm.

Algorithm 3.2. *(Prim's Algorithm[5])*

- *select an arbitrary vertex $v \in V$*
- *initialize $V_{new} = \{v\}$ and $E_{new} = \emptyset$*
- *while $|V_{new}| \neq |V|$*
  - *determine the edge $e = (u, v) \in E$ with minimum weight such that $u \in V_{new}, v \in V \setminus V_{new}$*
  - *add e to E_{new}: $E_{new} = E_{new} \cup \{e\}$*
  - *add v to V_{new}: $V_{new} = V_{new} \cup \{v\}$*

[4] comp. [29]
[5] comp. [30]

For the proof of the correctness of the algorithm we refer the reader to [30].

Finally we introduce an algorithm to compute the longest path[6] in a tree. Given a weighted tree, the longest path in the tree can be computed using a method of Edsger W. Dijkstra, which is as follows.

Algorithm 3.3. *An algorithm to compute the longest path in a given tree T.*

- *choose an arbitrary vertex v of the tree T as its root*
- *determine the deepest vertex u in the rooted tree T with the root v*
- *determine the deepest vertex w in the rooted tree T with the root u*

The claim is that the path from u to w is the longest path in the given tree T (see [4] for a formal proof of this claim.)

[6]The length of a path is not the number of edges of the path here, but the sum of the weights of edges of the path.

4 Quaternions

There are many ways for representing a rotation, including Euler angles, axis and angle, orthogonal matrices and unit quaternions. This chapter gives a brief summary of quaternions and then focusses on rotations represented by unit quaternions.

More detailed information about quaternions can be found in [15] and [11]. In [15] Horn uses unit quaternions to find a closed-form solution to the registration problem with known correspondences. In [11] quaternions are used to generate surfaces of fractal objects.

4.1 Basics of Quaternions

There are different ways, how a quaternion can be thought of. One can see a quaternion as a four dimensional vector or as composite of a scalar and a three dimensional vector or as a complex number with three imaginary parts. Complex number notation of a quaternion with a real part q_0 and three imaginary parts q_x, q_y and q_z has the following form:

$$\dot{q} = q_0 + iq_x + jq_y + kq_z.$$

Parameters i, j, k are defined with the properties

$$i^2 = j^2 = k^2 = -1, \quad ij = -ji = k.$$

Using these properties, one can easily compute:

$$jk = -kj = i, \quad ki = -ik = j.$$

Multiplication of quaternions is defined by the products of the components and is written as

$$\begin{aligned}\dot{r}\dot{q} = (&r_0q_0 - r_xq_x - r_yq_y - r_zq_z)\\ &+i(r_0q_x + r_xq_0 + r_yq_z - r_zq_y)\\ &+j(r_0q_y - r_xq_z + r_yq_0 + r_zq_x)\\ &+k(r_0q_z + r_xq_y - r_yq_x + r_zq_0).\end{aligned}$$

As it can easily be computed, quaternion multiplication is not commutative. Quaternion multiplication can be expressed in a more convenient way as a matrix-vector product as follows:

$$\dot{r}\dot{q} = \begin{pmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & -r_z & r_y \\ r_y & r_z & r_0 & -r_x \\ r_z & -r_y & r_x & r_0 \end{pmatrix} \dot{q} = R\dot{q}$$

or

$$\dot{q}\dot{r} = \begin{pmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{pmatrix} \dot{q} = \bar{R}\dot{q}.$$

If $\dot{r}$ is a unit quaternion, the matrices R and $\bar{R}$ associated with this quaternion are orthogonal.

The dot product of two quaternions is defined by analogy with vector operations[1], using the following sum of products of corresponding components:

$$\dot{p} \cdot \dot{q} = p_0 q_0 + p_x q_x + p_y q_y + p_z q_z.$$

The magnitude squared of a quaternion is defined as the dot product of the quaternion with itself. Unit quaternions have magnitude 1. Taking the conjugate of a quaternion has the following form:

$$\dot{q}^* = q_0 - iq_x - jq_y - kq_z.$$

Since the matrices associated with $\dot{q}^*$ are the transposes of the matrices associated with $\dot{q}$, it can easily be computed that the product of $\dot{q}$ and $\dot{q}^*$ is real and equals to $\dot{q} \cdot \dot{q}$. The inverse of a nonzero quaternion is computed using the expression

$$\dot{q}^{-1} = \left(\frac{1}{\dot{q} \cdot \dot{q}}\right) \dot{q}^*.$$

We conclude this section by deriving a result,

$$(\dot{p}\dot{q}) \cdot \dot{r} = (\bar{Q}\dot{p}) \cdot \dot{r} = (\bar{Q}\dot{p})^T \dot{r} = \dot{p}^T (\bar{Q}^T \dot{r}) = \dot{p} \cdot (\dot{r}\dot{q}^*), \tag{4.1}$$

which will be used in the next chapter.

4.2 Unit Quaternion Representation of Rotations

In this section, we first find a way to represent a rotation using a unit quaternion $\dot{q}$ and then show the orthogonal rotation matrix corresponding to $\dot{q}$.

Purely imaginary quaternions represent vectors. We need some way to map purely imaginary quaternions to purely imaginary quaternions. Since the product of such a quaternion with a unit quaternion can result in a quaternion with nonzero real part, we define the composite product

$$\dot{\bar{r}} = \dot{q}\dot{r}\dot{q}^* = (Q\dot{r})\dot{q}^* = \bar{Q}^T(Q\dot{r}) = (\bar{Q}^T Q)\dot{r},$$

which is purely imaginary. Q and $\bar{Q}$ are the matrices associated with the unit quaternion $\dot{q}$.

[1]The canonical inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ will also be denoted by the product $\boldsymbol{x} \cdot \boldsymbol{y}$.

Note that

$$\bar{Q}^T Q = \begin{pmatrix} \dot{q}\cdot\dot{q} & 0 & 0 & 0 \\ 0 & (q_0^2+q_x^2-q_y^2-q_z^2) & 2(q_xq_y-q_0q_z) & 2(q_xq_z+q_0q_y) \\ 0 & 2(q_yq_x+q_0q_z) & (q_0^2-q_x^2+q_y^2-q_z^2) & 2(q_yq_z-q_0q_z) \\ 0 & 2(q_zq_x-q_0q_y) & 2(q_zq_y+q_0q_x) & (q_0^2-q_x^2-q_y^2+q_z^2) \end{pmatrix}. \quad (4.2)$$

As you might know, rotations preserve dot- and cross products. Only rotations and reflections preserve dot products, but reflections reverse the sense of a cross product. The following equations reveal the fact that dot products are preserved by the composite product defined above.

$$\begin{aligned} (\dot{q}\dot{r}\dot{q}^*)\cdot(\dot{q}\dot{p}\dot{q}^*) &= \\ ((\bar{Q}^TQ)\dot{r})\cdot((\bar{Q}^TQ)\dot{p}) &= \\ ((\bar{Q}^TQ)\dot{r})^T((\bar{Q}^TQ)\dot{p}) &= \\ \dot{r}^TQ^T\bar{Q}\bar{Q}^TQ\dot{p} &= \dot{r}\cdot\dot{p} \end{aligned}$$

The last equation is true due to the fact that the matrices Q and $\bar{Q}$ corresponding to the unit quaternion $\dot{q}$ are orthogonal. In order to show that cross products are also preserved by the composite product, we think of quaternions as composite of a scalar and a three dimensional vector as follows:

$$\dot{q} = q + \boldsymbol{q}.$$

Now we use this notation for each quaternion and give the product $\dot{p} = \dot{r}\dot{s}$ in a more compact form

$$p = rs - \boldsymbol{r}\cdot\boldsymbol{s}, \qquad \boldsymbol{p} = r\boldsymbol{s} + s\boldsymbol{r} + \boldsymbol{r}\times\boldsymbol{s}.$$

The equations simplify, if the quaternions $\dot{r}$ and $\dot{s}$ have zero scalar part. In this case the product $\dot{p} = \dot{r}\dot{s} = p + \boldsymbol{p}$ is written as

$$p = -\boldsymbol{r}\cdot\boldsymbol{s}, \qquad \boldsymbol{p} = \boldsymbol{r}\times\boldsymbol{s}.$$

We apply the composite product with a unit quaternion $\dot{q}$ to purely imaginary quaternions $\dot{r}, \dot{s}$ and to the quaternion $\dot{p}$ and get

$$\dot{\bar{r}} = \dot{q}\dot{r}\dot{q}^*, \quad \dot{\bar{s}} = \dot{q}\dot{s}\dot{q}^*, \quad \dot{\bar{p}} = \dot{q}\dot{p}\dot{q}^*.$$

Now consider

$$\begin{aligned} \dot{\bar{r}}\dot{\bar{s}} = (\dot{q}\dot{r}\dot{q}^*)(\dot{q}\dot{s}\dot{q}^*) &= \\ (\dot{q}\dot{r})\underbrace{(\dot{q}^*\dot{q})}_{1}(\dot{s}\dot{q}^*) = \dot{q}(\dot{r}\dot{s})\dot{q}^* &= \\ (\bar{Q}^TQ)\dot{r}\dot{s} = (\bar{Q}^TQ)\dot{p} &= \\ \dot{q}\dot{p}\dot{q}^* = \dot{\bar{p}} &= \\ \bar{p} + \bar{\boldsymbol{p}}. & \end{aligned}$$

From the matrix (4.2), it's clear that $\boldsymbol{r} \times \boldsymbol{s}$ is mapped to $\bar{\boldsymbol{r}} \times \bar{\boldsymbol{s}}$ by the composite product with the unit quaternion $\dot{q}$. Thus cross products are preserved and the composite product with a unit quaternion can be used to represent rotations. Consequently, the lower-right-hand 3×3 submatrix of (4.2) is the orthogonal rotation matrix that corresponds to the unit quaternion $\dot{q}$.

4.3 Unit Quaternion from Orthogonal Matrix

In the preceding chapter, we have shown that the orthogonal rotation matrix R corresponding to the unit quaternion $\dot{q}$ is the lower-right-hand 3×3 submatrix of (4.2) and looks as follows

$$R = \begin{pmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_z) \\ 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{pmatrix}.$$

Sometimes it could be necessary to convert from the orthogonal rotation matrix to the unit quaternion. Consider following equations

$$\begin{aligned} 1 + r_{11} + r_{22} + r_{33} &= 4q_0^2 \\ 1 + r_{11} - r_{22} - r_{33} &= 4q_x^2 \\ 1 - r_{11} + r_{22} - r_{33} &= 4q_y^2 \\ 1 - r_{11} - r_{22} + r_{33} &= 4q_z^2, \end{aligned}$$

which are obtained from the diagonal elements of the matrix R. We evaluate left-hand side of these four equations and extract the largest component of the unit quaternion $\dot{q}$ to guarantee numerical accuracy by evaluating remaining components in the next step. Since $\dot{q}$ and $-\dot{q}$ represent the same rotation, we may choose either sign by taking the square root. Then we consider the following equations, which are obtained from the off-diagonal elements of the matrix R.

$$\begin{aligned} r_{32} - r_{23} &= 4q_0 q_x \\ r_{13} - r_{31} &= 4q_0 q_y \\ r_{21} - r_{12} &= 4q_0 q_z \\ r_{21} + r_{12} &= 4q_x q_y \\ r_{32} + r_{23} &= 4q_y q_z \\ r_{13} + r_{31} &= 4q_z q_x \end{aligned}$$

The remaining three components of the unit quaternion are evaluated by using three of these equations.

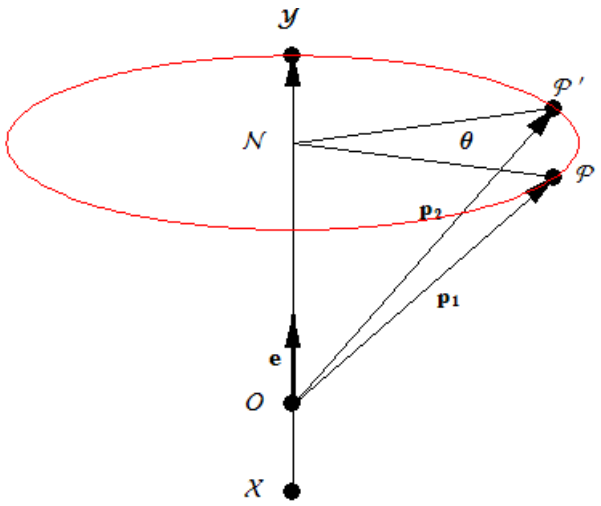

Figure 4.1: Rodrigues's rotation formula follows from the diagramm.

4.4 Unit Quaternion from Axis and Angle

$\boldsymbol{p_1}$ is any vector to be rotated by an angle θ about an axis (through the origin) given by the unit vector $\boldsymbol{e} = (e_x, e_y, e_z)^T$. On closer inspection of the Figure 4.1[2], we obtain the formula of Rodrigues

$$\boldsymbol{p_2} = \cos\theta \boldsymbol{p_1} + \sin\theta \boldsymbol{e} \times \boldsymbol{p_1} + (1 - \cos\theta)(\boldsymbol{e} \cdot \boldsymbol{p_1})\boldsymbol{e}.$$

Now we would like to show that the composite product $\dot{p_2} = \dot{q}\dot{p_1}\dot{q}^*$ corresponds to the same rotation described above, where the quaternions are

$$\dot{p_1} = 0 + \boldsymbol{p_1}, \quad \dot{p_2} = 0 + \boldsymbol{p_2}, \quad \dot{q} = q + \boldsymbol{q} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})\boldsymbol{e}.$$

Since $\dot{p_1}$ is purely imaginary and purely imaginary quaternions are mapped to purely imaginary quaternions by the composite product, $\dot{p_2}$ has zero scalar part and vector part

$$\boldsymbol{p_2} = (q^2 - \boldsymbol{q} \cdot \boldsymbol{q})\boldsymbol{p_1} + 2q\boldsymbol{q} \times \boldsymbol{p_1} + 2(\boldsymbol{q} \cdot \boldsymbol{p_1})\boldsymbol{q}.$$

Now we use the following identities from trigonometry and obtain the formula of Rodrigues.

$$2\sin(\frac{\theta}{2})\cos(\frac{\theta}{2}) = \sin\theta$$
$$\cos^2(\frac{\theta}{2}) - \sin^2(\frac{\theta}{2}) = \cos\theta$$

[2]source: http://www.sas.org/E-Bulletin/2002-10-25/mot/body.html

We conclude this chapter with a brief discussion of advantages of using unit quaternions to represent rotations. Composite rotations are formed by multiplication of quaternions and it takes fewer arithmetic operations to multiply two quaternions than to multiply two orthogonal rotation matrices. Since we calculate with finite precision, product of many orthogonal matrices may no longer be orthogonal, as product of many unit quaternions may no longer be unit. However, a quaternion is easy to normalize, but it is difficult to find the nearest orthogonal matrix to one that is quite not orthogonal.

5 Registration of two 3D Systems

5.1 Registration with Known Correspondences

We consider two point sets $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n)$ and $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n)$, where points with the same index correspond to each other. Our goal is to compute a similarity transformation

$$\alpha : \boldsymbol{x}' = \boldsymbol{a} + sR\boldsymbol{x}$$

that brings the transformed point set $X' = \alpha(X)$ as close as possible to the point set Y, where R is the rotation matrix, s is the scale factor and $\boldsymbol{a}$ is the translation vector. The objective function to be minimized is

$$F(\boldsymbol{a}, s, R) = \sum_{i=1}^{n} \|\boldsymbol{x}_i' - \boldsymbol{y}_i\|^2 = \sum_{i=1}^{n} (\boldsymbol{a} + sR\boldsymbol{x}_i - \boldsymbol{y}_i)^2. \tag{5.1}$$

First of all, we show that the optimal similarity transformation maps the centroids of point sets X and Y onto each other [15].

Lemma 5.1. *An optimal similarity transformation α, which minimizes (5.1) maps the centroid*

$$\boldsymbol{s}_x = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$$

of the point set X to the centroid of the point set Y,

$$\boldsymbol{s}_y = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i.$$

Proof. A necessary condition on a local minimizer $(\boldsymbol{a}^*, s^*, R^*)$ of (5.1) is vanishing of the gradient. Thus we have

$$\frac{\partial}{\partial \boldsymbol{a}} F(\boldsymbol{a}^*, s^*, R^*) = 2 \sum_{i=1}^{n} (\boldsymbol{a}^* + s^* R^* \boldsymbol{x}_i - \boldsymbol{y}_i) = 0. \tag{5.2}$$

We assume that α maps $\boldsymbol{s}_x$ to $\boldsymbol{s}_y$. Now we put $\boldsymbol{a}^* = \boldsymbol{s}_y - s^* R^* \boldsymbol{s}_x$ into equation (5.2) and get

$$\begin{aligned} \sum_{i=1}^{n} (\boldsymbol{s}_y - s^* R^* \boldsymbol{s}_x + s^* R^* \boldsymbol{x}_i - \boldsymbol{y}_i) &= \\ \sum_{i=1}^{n} (\boldsymbol{s}_y - \boldsymbol{y}_i) + s^* R^* \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{s}_x) &= 0. \end{aligned}$$

The last equation is satisfied, since

$$\sum_{i=1}^{n}(\boldsymbol{s}_y - \boldsymbol{y}_i) = n\boldsymbol{s}_y - \sum_{i=1}^{n}\boldsymbol{y}_i = 0, \quad \sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{s}_x) = 0.$$

□

The translational part of the optimal similarity transformation, which minimizes (5.1) is just the difference of the centroid of the point set Y and the rotated and scaled centroid of the point set X.

Now we translate both point sets X and Y such that the centroids are aligned at the origin. The new coordinates are denoted by

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{s}_x, \quad \tilde{\boldsymbol{y}}_i = \boldsymbol{y}_i - \boldsymbol{s}_y.$$

The objective function to be minimized is just

$$\bar{F}(s, R) = \sum_{i=1}^{n}(sR\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i)^2. \tag{5.3}$$

If $\alpha : \boldsymbol{x}' = sR\boldsymbol{x}$ is an affine map, $\bar{F}$ is quadratic in the unknown elements and its minimization amounts to the solution of a linear system of equations. But in our case the situation becomes harder, since the orthogonality condition $R \cdot R^T = I$ is a nonlinear constraint on R. However, this problem can be solved explicitly. For a comparison of four closed-form solutions to this problem, see [16]. We will present here the method of Horn [15].

5.1.1 Explicit Solution using Method of Horn

We expand the objective function (5.3), use the identity

$$\sum_{i=1}^{n}\|R\tilde{\boldsymbol{x}}_i\|^2 = \sum_{i=1}^{n}\|\tilde{\boldsymbol{x}}_i\|^2$$

and get

$$\underbrace{\sum_{i=1}^{n}\|\tilde{\boldsymbol{y}}_i\|^2}_{S_y} - 2s\underbrace{\sum_{i=1}^{n}\tilde{\boldsymbol{y}}_i \cdot (R\tilde{\boldsymbol{x}}_i)}_{D} + s^2\underbrace{\sum_{i=1}^{n}\|\tilde{\boldsymbol{x}}_i\|^2}_{S_x}. \tag{5.4}$$

After completing the square in s, (5.4) can be written as

$$\left(s\sqrt{S_x} - \frac{D}{\sqrt{S_x}}\right)^2 + \frac{(S_y S_x - D^2)}{S_x},$$

which is minimized with respect to the scale factor s, if the first term is zero, that is

$$s = \frac{\sum_{i=1}^{n}\tilde{\boldsymbol{y}}_i \cdot (R\tilde{\boldsymbol{x}}_i)}{\sum_{i=1}^{n}\|\tilde{\boldsymbol{x}}_i\|^2}.$$

The optimal rotation R that minimizes (5.4), maximizes

$$\sum_{i=1}^{n} \tilde{\boldsymbol{y}}_i \cdot (R\tilde{\boldsymbol{x}}_i),$$

which can be written in quaternion notation as

$$\sum_{i=1}^{n} (\dot{q}\dot{\tilde{x}}_i\dot{q}^*) \cdot \dot{\tilde{y}}_i, \tag{5.5}$$

where the quaternions $\dot{\tilde{x}}_i$ and $\dot{\tilde{y}}_i$ are

$$\dot{\tilde{x}}_i = 0 + \tilde{\boldsymbol{x}}_i, \quad \dot{\tilde{y}}_i = 0 + \tilde{\boldsymbol{y}}_i.$$

Suppose that $\tilde{\boldsymbol{x}}_i = (\tilde{x}_{l,i}, \tilde{y}_{l,i}, \tilde{z}_{l,i})$ while $\tilde{\boldsymbol{y}}_i = (\tilde{x}_{r,i}, \tilde{y}_{r,i}, \tilde{z}_{r,i})$. Our goal is finding the optimal unit quaternion $\dot{q}$ corresponding to the optimal rotation R, which maximizes (5.5). Using the result (4.1), we can rewrite (5.5) as

$$\sum_{i=1}^{n} (\dot{q}\dot{\tilde{x}}_i) \cdot (\dot{\tilde{y}}_i\dot{q})$$

that is,

$$\sum_{i=1}^{n} (\bar{R}_{\dot{\tilde{x}}_i}\dot{q}) \cdot (R_{\dot{\tilde{y}}_i}\dot{q})$$

with the matrices

$$\bar{R}_{\dot{\tilde{x}}_i} = \begin{pmatrix} 0 & -\tilde{x}_{l,i} & -\tilde{y}_{l,i} & -\tilde{z}_{l,i} \\ \tilde{x}_{l,i} & 0 & \tilde{z}_{l,i} & -\tilde{y}_{l,i} \\ \tilde{y}_{l,i} & -\tilde{z}_{l,i} & 0 & \tilde{x}_{l,i} \\ \tilde{z}_{l,i} & \tilde{y}_{l,i} & -\tilde{x}_{l,i} & 0 \end{pmatrix}$$

and

$$R_{\dot{\tilde{y}}_i} = \begin{pmatrix} 0 & -\tilde{x}_{r,i} & -\tilde{y}_{r,i} & -\tilde{z}_{r,i} \\ \tilde{x}_{r,i} & 0 & -\tilde{z}_{r,i} & \tilde{y}_{r,i} \\ \tilde{y}_{r,i} & \tilde{z}_{r,i} & 0 & -\tilde{x}_{r,i} \\ \tilde{z}_{r,i} & -\tilde{y}_{r,i} & \tilde{x}_{r,i} & 0 \end{pmatrix}.$$

The sum (5.5) that we have to maximize can now be written as

$$\sum_{i=1}^{n} (\dot{q}^T \bar{R}^T_{\dot{\tilde{x}}_i})(R_{\dot{\tilde{y}}_i}\dot{q})$$

or

$$\dot{q}^T \left(\sum_{i=1}^{n} \bar{R}^T_{\dot{\tilde{x}}_i} R_{\dot{\tilde{y}}_i} \right) \dot{q}$$

that is,

$$\dot{q}^T \left(\sum_{i=1}^{n} N_i \right) \dot{q}$$

or

$$\dot{q}^T N \dot{q}.$$

Each N_i is symmetric, thus N is a 4×4 symmetric matrix.

Lemma 5.2. *The normalized eigenvector corresponding to the most positive eigenvalue of the real symmetric matrix N is the unit quaternion $\dot{q}$ that maximizes*

$$\dot{q}^T N \dot{q}.$$

Proof. The real symmetric matrix N has four real eigenvalues $\lambda_1, \lambda_2, \lambda_3$ and λ_4. The corresponding set of orthonormal eigenvectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ and $\boldsymbol{e}_4$ span the four dimensional space. Thus an arbitrary unit quaternion $\dot{q}$ can be written as a linear combination of these eigenvectors

$$\dot{q} = \alpha_1 \boldsymbol{e}_1 + \alpha_2 \boldsymbol{e}_2 + \alpha_3 \boldsymbol{e}_3 + \alpha_4 \boldsymbol{e}_4.$$

Note that

$$N\dot{q} = \alpha_1 \lambda_1 \boldsymbol{e}_1 + \alpha_2 \lambda_2 \boldsymbol{e}_2 + \alpha_3 \lambda_3 \boldsymbol{e}_3 + \alpha_4 \lambda_4 \boldsymbol{e}_4,$$

since $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ and $\boldsymbol{e}_4$ are the eigenvectors of the symmetric matrix N. The term to maximize can be written in the form

$$\dot{q}^T N \dot{q} = {\alpha_1}^2 \lambda_1 + {\alpha_2}^2 \lambda_2 + {\alpha_3}^2 \lambda_3 + {\alpha_4}^2 \lambda_4,$$

since the normalized eigenvectors are orthogonal to each other. Without loss of generality, we assume that the eigenvalues are arranged in descending order, so that

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4.$$

Consider

$$\begin{aligned} \dot{q}^T N \dot{q} \leq {\alpha_1}^2 \lambda_1 + {\alpha_2}^2 \lambda_1 + {\alpha_3}^2 \lambda_1 + {\alpha_4}^2 \lambda_1 \quad = \\ \lambda_1 ({\alpha_1}^2 + {\alpha_2}^2 + {\alpha_3}^2 + {\alpha_4}^2) = \lambda_1 (\dot{q} \cdot \dot{q}) = \lambda_1. \end{aligned}$$

This maximum of the quadratic form is attained, if we choose $\alpha_1 = 1$ and $\alpha_2 = \alpha_3 = \alpha_4 = 0$, that is

$$\dot{q} = \boldsymbol{e}_1.$$

□

The solution given above simplifies, if each point set is coplanar. This special case is dealt more directly. For information about this special case and more detail about the method of Horn, we refer to [15].

There is also an iterative solution to the registration problem discussed above using instantaneous kinematics. This solution computes a time independent velocity vector field, which attaches to each point $\boldsymbol{x}$ a velocity vector

$$\boldsymbol{v}(\boldsymbol{x}) = \bar{\boldsymbol{c}} + \gamma\boldsymbol{x} + \boldsymbol{c} \times \boldsymbol{x}$$

such that the quadratic function

$$F(\boldsymbol{c}, \bar{\boldsymbol{c}}, \gamma) = \sum_i (\boldsymbol{x}_i + \boldsymbol{v}(\boldsymbol{x}_i) - \boldsymbol{y}_i)^2 = \sum_i (\boldsymbol{x}_i + \bar{\boldsymbol{c}} + \gamma\boldsymbol{x}_i + \boldsymbol{c} \times \boldsymbol{x}_i - \boldsymbol{y}_i)^2$$

is minimized. This approach of registration using instantaneous kinematics will be discussed in more detail in the next section. We will introduce the method proposed in [25], since we use it in our implementation.

The time independent velocity vector field

$$\boldsymbol{v}(\boldsymbol{x}) = \bar{\boldsymbol{c}} + \gamma\boldsymbol{x} + \boldsymbol{c} \times \boldsymbol{x}$$

given by the triple $(\boldsymbol{c}, \bar{\boldsymbol{c}}, \gamma) \in \mathbb{R}^7$ determines a unique uniform equiform motion that maps points $\boldsymbol{x} \in \mathbb{R}^3$ according to $\boldsymbol{y}(t) = \alpha(t)A(t)\boldsymbol{x} + \boldsymbol{a}(t)$, with a rotation matrix $A(t)$, a translation vector $\boldsymbol{a}(t)$ and a scaling factor $\alpha(t)$ (cf. [12]).

Instantaneous kinematics has been furthermore used for simultaneously registration of more than two systems (see [23]).

5.2 Registration with Unknown Correspondences

In the preceding section we discussed about the registration problem of two point sets $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n)$ and $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_n)$ by given correspondences. In applications, however, it could occur that the number of points in both point sets are not equal and point-to-point correspondences are not given. A well known application of registration without correspondences is the alignment of a 3D point set to a CAD model (see Figure 5.1[1]). This application makes it clear that the geometric data is not necessarily represented by a point set. In this section, we introduce two algorithms how to move a source model S to be in best alignment with a target model T, where following representations are allowed for the 3D models:

1. Point set
2. Triangulated surface

In [3] more allowable representations are considered.

[1] source: [25]

Figure 5.1: Registration of a point cloud to a CAD model represented as a triangulated surface

5.2.1 Registration using ICP

Iterative closest point (ICP) algorithm of P.Besl and N.D. McKay [3] is the most widely used algorithm for the solution of the problem discussed above.

Algorithm 5.3. *Iterative closest point (ICP) algorithm is an iterative algorithm and involves the following steps*

1. *If the source model S is not given as a point set, but as a triangulated surface, it must be decomposed into a point set $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots)$. It is trivial, since the vertices of the triangles can be used as the point set.*

2. *It is important that the ICP starts with an acceptable initial position of the source model S.*

3. *In the first step of each iteration, for every source point $\boldsymbol{x}_i \in X$ the closest point on the target model T is computed. Denote the resulting set of closest points on the target model T as $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots)$. Each point $\boldsymbol{x}_i$ corresponds to the point $\boldsymbol{y}_i$ with the same index.*

4. *Compute the similarity transformation α such that the following objective function is minimized*

$$\sum_i (\alpha(\boldsymbol{x}_i) - \boldsymbol{y}_i)^2, \tag{5.6}$$

   as described in the last section.

5. *Update the positions of the source points via $X_{new} = \alpha(X_{old})$.*

6. *Repeat the steps 3 to 5, until the change in the mean-square error falls below a preset threshold $\tau > 0$ or a maximum number of iterations is reached.*

Since the value of the objective function (5.6) reduces in each iteration step, the algorithm converges monotonically to a local minimum. In the following, steps 2 and 3 will be discussed in more detail.

Initial Position for ICP

A rough initial alignment of the models can be obtained using principal component analysis (PCA). Compute the covariance matrices of the point sets X and Y, which are sampled from the surfaces of the models. The covariance matrix is given by

$$J_X := \sum_i \boldsymbol{x}_i \cdot \boldsymbol{x}_i{}^T,$$

where $\boldsymbol{x}_i$ are considered as column vectors. Then, compute the normalized eigenvectors $\boldsymbol{e}_{x_1}, \boldsymbol{e}_{x_2}, \boldsymbol{e}_{x_3}$ and eigenvalues $\lambda_{x_1}, \lambda_{x_2}$ and λ_{x_3} of the matrix J_X such that the eigenvalues are arranged in descending order:

$$\lambda_{x_1} \geq \lambda_{x_2} \geq \lambda_{x_3}.$$

Now do the same for the matrix J_Y. For an initial alignment of the models, we align centroids of the point sets X and Y at the origin and then rotate the source model such that the eigenvectors $\boldsymbol{e}_{x_1}, \boldsymbol{e}_{x_2}, \boldsymbol{e}_{x_3}$ of J_X are aligned with the eigenvectors $\boldsymbol{e}_{y_1}, \boldsymbol{e}_{y_2}, \boldsymbol{e}_{y_3}$ of J_Y. Note that there are four possibilities for the rotation: take the one which gives the least sum of squared distances of $\boldsymbol{x}_i \in X$ to the closest points $\boldsymbol{y}_i \in Y$.

Computation of Closest Points

In the first step of the ICP algorithm, we decomposed the source model S into a point set $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots)$. Our goal is for each $\boldsymbol{x}_i$ finding the closest point on the target model T given as a point set $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots)$ or as a triangulated surface. It is the most computationally expensive step in the ICP algorithm and has to be implemented efficiently. We give an overview of the methods used for finding closest points.

1. The closest point in the point set Y or the closest vertex of the triangulated surface is found efficiently using a kd-tree [2].

2. For a better approximation of the closest point of a triangulated surface to a given source point p, computation of an additional step is required. Given v as the closest vertex to the source point p, the closest point will lie within or on the boundary of one of the triangles $\triangle_1, \cdots, \triangle_n$ to which the vertex v belongs. In order to find the closest point, we project the source point p into the planes defined by each of the triangles. If the projected point lies within the triangle $\triangle_j$, p_j denotes this projected point. If the projected point does not lie within the corresponding

triangle $\triangle_j$, p_j denotes the point on the boundary of the triangle $\triangle_j$, which is closest to the projected point. The closest point is found among the points p_j, which is closest to p.

This method does not guarantee to find the true closest point, since the true closest point may lie in a totally different triangle (see Figure 5.2). The presented method is introduced in the thesis [28].

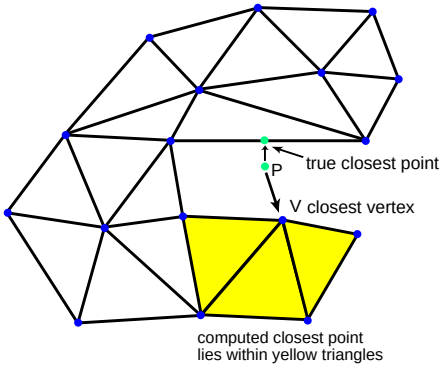

Figure 5.2: Closest point computation fails, since true closest point lies in a different triangle.

3. The closest point on the triangulated surface is the intersection point of the straight line going through the source point in the direction of the source point's normal with the triangulated surface. This method, denoted as normal shooting in [6], yields bad results for complex and noisy meshes (see Figure 5.3).

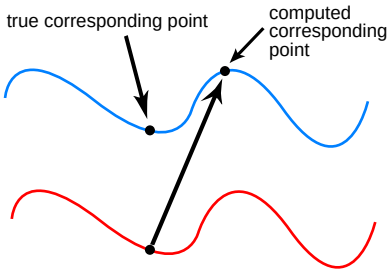

Figure 5.3: Normal shooting yields bad results for complex and noisy meshes.

4. Maier et al. [17] build a data structure that classifies many of the triangles as irrelevant for closest point search. The source point is projected onto every remaining

triangle and the projected point with the minimum distance is found as the closest point.

Now we present a variant of the ICP algorithm, where appropriate weights are assigned to corresponding point pairs to downweight outliers. The weights are choosen such that pairs with greater point-to-point distances are assigned lower weights. We run the ICP algorithm for a few steps with constant weights and then enter a weight iteration, where the objective function in the fourth step of the ICP algorithm looks as follows

$$\sum_i w_i(\alpha(\boldsymbol{x}_i) - \boldsymbol{y}_i)^2. \tag{5.7}$$

The minimization of the objective function (5.7) is slightly different from the minimization of (5.6) and is also performed using the method of Horn [15]. The centroids become weighted centroids

$$\boldsymbol{s}_x = \frac{\sum_i w_i \boldsymbol{x}_i}{\sum_i w_i}, \quad \boldsymbol{s}_y = \frac{\sum_i w_i \boldsymbol{y}_i}{\sum_i w_i}.$$

The translation vector $\boldsymbol{a}$ is computed using these centroid, as before. The only change in the method for finding the unit quaternion corresponding to the optimal rotation is that the following sum is weighted

$$\sum_i w_i N_i.$$

We show some weighting schemes, which have also been used in our implementation.

1. Compute for each corresponding point pair $(\boldsymbol{x}, \boldsymbol{y})$ the euclidean distance d, then use the following weighting function

   $$w(d) = \frac{1}{1 + \alpha d^\beta}.$$

   The constants α and β are choosen such that the pair with the lowest distance is assigned a weight close to 1 and the pair with the greatest distance is assigned a weight close to 0.

2. We use the following linear function to assign a weight w to the point pair $(\boldsymbol{x}, \boldsymbol{y})$

   $$w = 1 - \frac{d - d_{min}}{d_{max} - d_{min}},$$

   where d is the euclidean distance of this pair, d_{min} and d_{max} are the lowest and greatest distances among all point pairs (comp. [10]).

There are many other variants of the ICP algorithm. We refer the reader to [26] for more detail about these variants.

Now we describe how to compute the second order Taylor approximant of the squared distance function of the target model at a source point $\boldsymbol{x}$. Our goal is to show that the

distances to the closest points, which are used in the ICP algorithm, are not good in the vicinity of the target model. We consider the oriented surface represented by the target model. For any surface point, we assume that the unit normal $\boldsymbol{n}$ along with the principal curvature directions $\boldsymbol{e}_1, \boldsymbol{e}_2$ are given. These three unit vectors form a local right-handed Cartesian system, which is called the principal frame. At umbilical points, where the principal directions are not defined uniquely, we take any of two orthogonal tangent vectors $\boldsymbol{e}_1, \boldsymbol{e}_2$. The principal radius of curvature in the direction of $\boldsymbol{e}_i$ is denoted by ρ_i. Furthermore $\boldsymbol{y}$ denotes the closest point on the surface from the source point $\boldsymbol{x}$. The coordinates of $\boldsymbol{x}$ in the principal frame at $\boldsymbol{y}$ is denoted by x_1, x_2, x_3. The signed distance from $\boldsymbol{x}$ to its closest point $\boldsymbol{y}$ is represented by d. The local quadratic Taylor approximant of the squared distance function of a surface at a point $\boldsymbol{x}$ is expressed in the principal frame at the closest point via

$$F_d(x_1, x_2, x_3) = \frac{d}{d-\rho_1} x_1^2 + \frac{d}{d-\rho_2} x_2^2 + x_3^2. \tag{5.8}$$

For more detail about the squared distance function of surfaces and the derived result, we refer the reader to [22].

We look at two special cases of the function (5.8)

- Consider the case
$$F_0(x_1, x_2, x_3) = x_3^2,$$
where $d = 0$. This means, that the squared distance function to the tangent plane at the closest point, is a good approximant in the vicinity of the surface.

- For $d = \infty$, we obtain
$$F_\infty(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2.$$
This means, that the squared distance function to the closest point, which is used in the ICP algorithm, is a good approximant, if we are in greater distance to the surface.

In the following, we present a registration algorithm, where sum of squared distances to the tangent planes is minimized. We use instantaneous kinematics for the solution.

5.2.2 Registration using Instantaneous Kinematics

Our goal is to apply a rigid body transformation m to the point set $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N)$ such that the following sum of squared distances d^2 to the tangent planes is minimized

$$F = \sum_{i=1}^{N} d^2(m(\boldsymbol{x}_i), \Phi), \tag{5.9}$$

where Φ is the triangulated surface representing the target model. In the first step of the algorithm, we compute for each source point $\boldsymbol{x}_i$ the closest point $\boldsymbol{y}_i$ of the surface Φ

and determine the unit normal vector $\boldsymbol{n}_i$ there. For a triangulated surface, $\boldsymbol{x}_i$ will not lie exactly on the surface normal $\boldsymbol{n}_i$ of its closest point $\boldsymbol{y}_i$ and we have $\boldsymbol{x}_i = \boldsymbol{y}_i + d_i\boldsymbol{n}_i + \boldsymbol{t}_i$, where d_i is the signed distance of $\boldsymbol{x}_i$ to the tangent plane in $\boldsymbol{y}_i$ and $\boldsymbol{t}_i$ is a vector orthogonal to the unit normal vector $\boldsymbol{n}_i$. In the following, we ignore the tangential component $\boldsymbol{t}_i$, since we are interested in the minimization of the squared distances to the tangent planes. The following step of the algorithm concerns the minimization of the objective function (5.9) using instantaneous kinematics, as proposed in [25].

We aim to compute a velocity vector field determined by the tuple $(\boldsymbol{c}, \bar{\boldsymbol{c}})$, which attaches to each point $\boldsymbol{x}_i$ a velocity vector

$$\boldsymbol{v}(\boldsymbol{x}_i) = \bar{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}_i$$

such that the sum of squared distances of the points $\boldsymbol{x}_i + \boldsymbol{v}(\boldsymbol{x}_i)$ to the tangent planes at $\boldsymbol{y}_i$ is minimized. The distance of $\boldsymbol{x}_i + \boldsymbol{v}(\boldsymbol{x}_i)$ to the tangent plane in the point $\boldsymbol{y}_i$ is given by

$$d_i + \boldsymbol{n}_i \cdot \boldsymbol{v}(\boldsymbol{x}_i). \tag{5.10}$$

The objective function to be minimized can be written as

$$F(\boldsymbol{c}, \bar{\boldsymbol{c}}) = \sum_{i=1}^{N} (d_i + \boldsymbol{n}_i \cdot (\bar{\boldsymbol{c}} + \boldsymbol{c} \times \boldsymbol{x}_i))^2.$$

$F(\boldsymbol{c}, \bar{\boldsymbol{c}})$ is quadratic in the unknowns $\boldsymbol{c}$ and $\bar{\boldsymbol{c}}$ and its minimization leads to the solution of a system of linear equations. To derive the system of linear equations, we rewrite (5.10) as

$$d_i + \boldsymbol{n}_i \cdot \bar{\boldsymbol{c}} + (\boldsymbol{x}_i \times \boldsymbol{n}_i) \cdot \boldsymbol{c} = d_i + (\boldsymbol{x}_i \times \boldsymbol{n}_i, \boldsymbol{n}_i) \begin{pmatrix} \boldsymbol{c} \\ \bar{\boldsymbol{c}} \end{pmatrix} = d_i + A_i C.$$

Using that we rewrite the objective function $F(\boldsymbol{c}, \bar{\boldsymbol{c}})$ as

$$\begin{aligned} F(\boldsymbol{c}, \bar{\boldsymbol{c}}) &= \sum_{i=1}^{N} (d_i + A_i C)^2 \\ &= \sum_{i=1}^{N} d_i^2 + 2 \sum_{i=1}^{N} d_i A_i C + \sum_{i=1}^{N} C^T A_i^T A_i C \\ &= D + 2B^T C + C^T A C^T, \end{aligned}$$

where D is a scalar, B is a six dimensional column vector and A is a symmetric six-by-six matrix. It is well known that a local minimizer C^* solves the linear system

$$AC + B = 0.$$

If A is positive definit, the system is regular and its solution is the unique global minimizer of $F(\boldsymbol{c}, \bar{\boldsymbol{c}})$. Moving each point $\boldsymbol{x}_i$ to the position $\boldsymbol{x}_i + \boldsymbol{v}(\boldsymbol{x}_i)$ would not yield a rigid body transformation, but an affine one. Therefore we use the helical motion determined

by the pair $(\boldsymbol{c}, \bar{\boldsymbol{c}})$ to update the position of the point set X. From the pair $(\boldsymbol{c}, \bar{\boldsymbol{c}})$, the Plücker coordinates of the axis G, the pitch p and the angular velocity ω of the helical motion are computed as

$$\boldsymbol{g} = \frac{\boldsymbol{c}}{||\boldsymbol{c}||}, \quad \bar{\boldsymbol{g}} = \frac{\bar{\boldsymbol{c}} - p\boldsymbol{c}}{||\boldsymbol{c}||}, \quad p = \frac{\boldsymbol{c} \cdot \bar{\boldsymbol{c}}}{\boldsymbol{c}^2}, \quad \omega = ||\boldsymbol{c}||. \tag{5.11}$$

The Plücker coordinates $(\boldsymbol{g}, \bar{\boldsymbol{g}})$ of a line G consist of the direction vector $\boldsymbol{g}$ and the moment vector $\bar{\boldsymbol{g}} = \boldsymbol{p} \times \boldsymbol{g}$, where $\boldsymbol{p}$ is an arbitrary point on the line G. Detail information about Plücker coordinates of a line, helical motions and the equations (5.11) can be found in [24]. To update the positions of the source points via $X_{new} = m(X_{old})$, we apply a rotation about the axis G by an angle $\alpha = \arctan(\omega)$ and a translation parallel to G by the distance $p \cdot \alpha$. We iterate, always using the updated point set X_{new}, until the change in the mean-square error falls below a preset threshold or a maximum number of iterations is reached. The presented algorithm can be extended with a minor change to consider similarity transformations, since the velocity vector is still linear and has one more parameter γ.

For a solution of the registration problem, where the general second order Taylor approximant (5.8) of the squared distance function of a surface is used, see [19].

6 Discovering Structural Regularity

This chapter concerns the detection of regular structures in 3D mesh based models as proposed in [21]. They define a *regular structure* of size n as a tuple $(\mathcal{P}, \mathcal{G})$, where $\mathcal{P} = \{P_0, \cdots, P_{n-1}\}$ is a collection of n patches $P_k \subset \mathcal{S}$ of a given surface $\mathcal{S}$ and $\mathcal{G}$ is a k-parameter transformation group acting on $\mathcal{P}$ with generating similarity transformation(s) $T_1, T_2, \cdots, T_k$.

The geometry of a regular structure $(\mathcal{P}, \mathcal{G})$, where $\mathcal{G}$ is a k-parameter group, can be represented by a single representative patch P_0, group generator(s) $T_1, T_2, \cdots, T_k$ and integer dimension(s) $n_1, n_2, \cdots, n_k$ with $n_1 \cdots n_k = n$. They call $(P_0, \{T_i\}, \{n_i\})$ the *generative model* of the regular structure.

As an example, consider that $\mathcal{G}$ is a two-parameter group with group generators T_1 and T_2. Then each element $P_{i,j} \in \mathcal{P} = \{P_{0,0}, \cdots, P_{n_1-1,0}, P_{0,1}, \cdots, P_{n_1-1,1}, \cdots, P_{n_1-1,n_2-1}\}$ can be represented by $P_{i,j} = T_1^i T_2^j P_0$, where $P_0 \in \mathcal{P}$ is the representative element and $|\mathcal{P}| = n_1 \cdot n_2 = n$. The generative model of this regular structure is $(P_0, \{T_1, T_2\}, \{n_1, n_2\})$. This example is illustrated in the figure shown below.

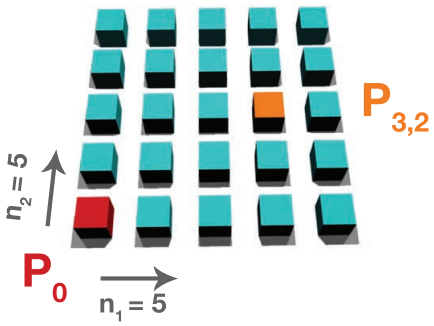

Figure 6.1: The group generators T_1 and T_2 are two independent translations. Let P_0 be the representative element of the regular structure. It can be transformed into any other element $P_{i,j}$ by the transformation $T_1^i T_2^j$, e.g. $P_{3,2} = T_1^3 T_2^2 P_0$.

The objective of the algorithm is to find a generative model $(P_0, \{T_i\}, \{n_i\})$ such that as much as possible of the given input surface $\mathcal{S}$ can be represented by the union of repetitive patches $P_0, \cdots, P_{n-1}$, while the number of repetitions n is maximized.

Pauly et al. [21] consider the detection of regular structures $(\mathcal{P}, \mathcal{G})$, where $\mathcal{G}$ is a commutative one- or two-parameter subgroup of the group of similarity transformations.

They describe their approach for two-parameter subgroups and consider one-parameter subgroups as a special case.

They show that there are only the following three types of commutative two-parameter groups, which can be seen in Figure 6.2

- Case *Trans* × *Trans*: Two independent translations.
- Case *Rot* × *Trans*: Rotation and translation parallel to the rotation axis.
- Case *Rot* × *Scale*: Rotation and scaling with center of scaling on the rotation axis.

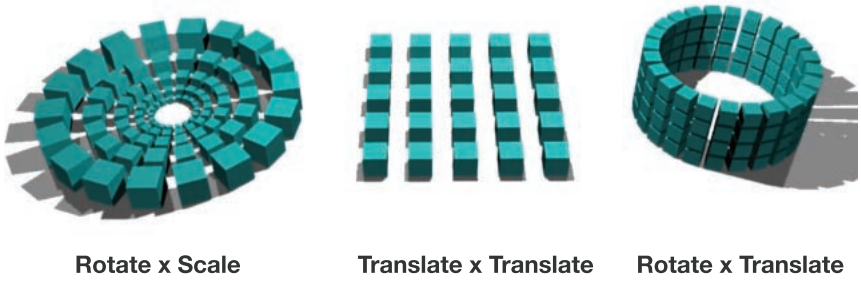

Figure 6.2: Commutative two-parameter subgroups of the group of similarity transformations.

One-parameter groups are shown on the next page in Figure 6.3

Before describing the main three steps of the algorithm in detail, we give an overview of these steps.

The first step of the algorithm decomposes the input surface $\mathcal{S}$ into small local surface patches, estimates similarity transformations between these patches and defines a suitable mapping from the space of similarity transformations to an auxiliary 2D space.

In the second step of the algorithm, they search this 2D space for characteristic lattice patterns, which is equivalent to estimate the parameters of the generative model of a regular structure.

In the final step, they aggregate spatially adjacent local surface patches with compatible group structure to build large-scale repetitive elements and optimize generating transformations estimated from small-scale surface patches using simultaneous registration in the spatial domain.

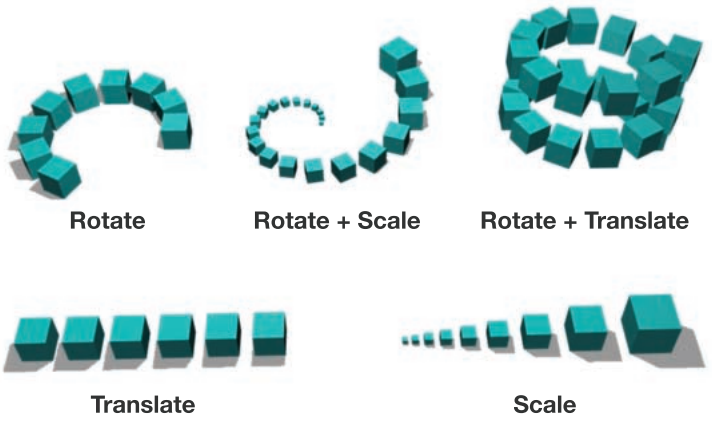

Figure 6.3: Commutative one-parameter subgroups of the group of similarity transformations.

6.1 Transformation Analysis

6.1.1 Similarity Sets

They compute a uniform random sampling of the input surface $\mathcal{S}$ by following the sampling approach proposed in [20]. Each sample point represents a local surface patch of the surface $\mathcal{S}$. Their goal is to find evidence for a regular structure by analyzing the similarity transformations between these patches. They avoid the quadratic complexity of considering all pairwise matches by grouping the local surface patches into similarity sets Ω_l using a local shape descriptor, which is invariant under similarity transformations. Only sample points with similar descriptor values are considered as a candidate for a regular structure. They estimate for each sample point mean and Gaussian curvatures H and K based on the method proposed in [5] and use H^2/K as the shape descriptor value, which is invariant under similarity transformations. If scaling is not considered, they use the tupel (H, K) as the local shape descriptor, since this tupel is invariant under rotation and translation. Sample points with similar descriptor values are grouped together.

6.1.2 Local Alignment

The similarity sets are processed in descending order of number of sample points. Let Ω denote the current processed similarity set. They estimate for each sample point pair $(\boldsymbol{p}_i, \boldsymbol{p}_j) \in \Omega^2$ the similarity transformation T_{ij} that maps the corresponding local surface patches onto each other. The translational and rotational part of T_{ij} is computed

by aligning the local principal frames. Umbilical points are not considered, since the principal directions are not defined uniquely. The uniform scaling factor is calculated from the ratio of the mean curvatures H_i/H_j. They refine the estimated initial similarity transformation using the non-rigid registration algorithm [25]. The sample pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$ is discarded, if the alignment error of the registration algorithm is above a given threshold. They define $\mathcal{T}$ as the set of the remaining similarity transformations.

6.1.3 Transformation Mapping

As we said before, they introduce a suitable mapping that exposes a uniform lattice structure for the set $\mathcal{T}$ of similarity transformations. They first determine if a rotational regular structure could be present by considering the rotation angle of the transformations. If only an insignificant part of the transformations has a non-zero rotation angle, the model contains regular structures of type *Trans* $\times$ *Trans*. In this case, they search for 2D planes through the origin in the space of 3D translation vectors. They use RANSAC (Random Sample Consensus) method by Fischler and Bolles [8] to estimate the parameters of the planes. After projecting the translation vectors onto the retrieved planes, they apply the mapping $T \rightarrow (t_1, t_2)$, where t_1 and t_2 denote the 2D coordinates of the projected translation vectors.

If a significant part of the transformations has non-zero rotation angle, they group the transformations into sets with similar direction of the rotation axis. Then for each such set they determine if a regular structure of type *Rot* $\times$ *Scale* could be present by checking if there is a significant variation in the scaling factors. If such a variation is found, the model contains regular structures of type *Rot* $\times$ *Scale* and they define the mapping $T \rightarrow (\theta, \log s)$.

If a great variation in scaling factors is not found, the model contains regular structures of type *Rot* $\times$ *Trans* and they apply the mapping $T \rightarrow (\theta, t)$, where $t = \boldsymbol{t} \cdot \boldsymbol{a}$ is computed as the projection of the translation vector $\boldsymbol{t}$ onto the unit direction vector $\boldsymbol{a}$ of the rotation axis.

Each of the defined mappings $T \rightarrow (t_1, t_2), T \rightarrow (\theta, \log s)$ and $T \rightarrow (\theta, t)$ has the common property that composition of similarity transformations corresponds to the sum of vectors in the auxiliary 2D space. Consider the composition of the transformations T and T', where the second transformation T' is mapped to (t'_1, t'_2) or $(\theta', \log s')$ or (θ', t'), respectively. Then the composite product $T'T$ is mapped to

- Case *Trans* $\times$ *Trans*: $T'T \rightarrow (t'_1 + t_1, t'_2 + t_2)$
- Case *Rot* $\times$ *Scale*: $T'T \rightarrow (\theta' + \theta, \log(s's)) = (\theta' + \theta, \log s' + \log s)$
- Case *Rot* $\times$ *Trans*: $T'T \rightarrow (\theta' + \theta, t' + t)$

The identity transformation is always mapped to the origin $(0, 0)$. The inverse transformation T^{-1} is mapped to $(-t_1, -t_2)$ or $(-\theta, -\log s)$ or $(-\theta, -t)$, respectively. The sum of angles are computed modulo 2π. It follows that the commutative two-parameter group of transformations $\{T^i T'^j\}$ is mapped to a regular lattice in the auxiliary 2D

space, where the lattice passes through the origin, since the identity transformation is always part of a transformation group.

6.2 Model Estimation

The goal of this step is to estimate the parameters of the generative model of a regular structure. In other words, one searches for regular lattices of clusters in a 2D distribution of points. The following issues make this task difficult:

- The similarity set Ω contains local surface patches, which are not part of the regular structure. The similarity transformations between these patches add clutter to the 2D transformation space such that lattice patterns are hidden.
- Some clusters that should be present are less pronounced, since noise in the model and local variations of sample positions can lead to inaccuracies in the estimation of the pairwise similarity transformations. Also missing geometry and thus missing transformations result in less pronounced clusters.

Their grid fitting approach, which is robust to outliers and missing geometry, operates on the set of cluster centers $C = \{\boldsymbol{c}_k\}$, which are obtained using the mean-shift clustering algorithm [7]. They describe their approach for a regular grid of size $n_1 \times n_2$. One-parameter groups are considered as a special case with $n_1 = 1$. In the following, we introduce their optimization method for grid fitting, where two vectors $\mathbf{g}_1, \mathbf{g}_2 \in \mathbb{R}^2$ are sufficient to represent the grid locations $X = \{x_{ij}\}$, since the grid must pass through the origin.

6.2.1 Energy Minimization

The objective function to be minimized is a combination of different energy terms. The first term minimizes the sum of squared distances of the grid locations to the closest cluster centers and looks as follows:

$$E_{X \to C} = \sum_i \sum_j \alpha_{ij}^2 \|x_{ij} - \boldsymbol{c}(i,j)\|^2,$$

where $\boldsymbol{c}(i,j)$ is the cluster center closest to the grid location x_{ij}. The second term in the objective function is similar to the first term and minimizes the sum of squared distances of cluster centers to the closest grid locations using the energy

$$E_{C \to X} = \sum_{i=1}^{|C|} \beta_i^2 \|\boldsymbol{c}_i - x(i)\|^2,$$

where $x(i) \in X$ is the grid location closest to the cluster center $\boldsymbol{c}_i$. The variables α_{ij} and β_i are weights that measure how reliably a grid position $x_{ij} \in X$ can be mapped to its closest cluster center $\boldsymbol{c}(i,j)$ and vice versa. Values of the weights α_{ij}, β_i close to

one indicate a reliable correspondence of the grid locations and cluster centers. On the other hand, values close to zero indicate outliers or missing geometry. There are two additional terms in the objective function, which aim to maximize the number of valid corresponding grid locations and cluster centers:

$$E_\alpha = \sum_i \sum_j (1 - \alpha_{ij}^2)^2, \quad E_\beta = \sum_i (1 - \beta_i^2)^2.$$

Finally, the fitting terms $E_{X\to C}, E_{C\to X}$ and correspondence terms E_α, E_β are combined in the following objective function

$$E = \gamma(E_{X\to C} + E_{C\to X}) + (1-\gamma)(E_\alpha + E_\beta),$$

which is minimized with respect to the grid generators $\mathbf{g}_1, \mathbf{g}_2 \in \mathbb{R}^2$ and the weights α_{ij}, β_i. The parameter γ balances the fitting and correspondence terms. An iterative Gauss-Newton solver is used to minimize the objective function E. They initialize the value of the weights to one, since they assume no prior knowledge of the geometry. In order to initialize the grid generators $\mathbf{g}_1, \mathbf{g}_2 \in \mathbb{R}^2$, they compute the two most dominant lines through the origin using the RANSAC method and determine the initial generator values as clusters with $\|\mathbf{g}_i\|$ minimal on these lines. The grid size is estimated from the furthest clusters on these lines.

6.3 Aggregation

The output of the last step is a set of regular structures at the scale of local surface patches. The goal of this final step is to aggregate spatially adjacent local surface patches with compatible group structure to build large-scale repetitive elements and improve the accuracy of estimated generating transformations using simultaneous registration, while growing the regions being matched. Simultaneous registration in the spatial domain is essential, since the similarity transformations estimated from small surface patches can be inaccurate.

6.3.1 Simultaneous Registration

They first describe the case of a one-parameter regular structure. Using homogeneous coordinates, a similarity transformation T can be represented using the matrix $\mathbf{H}$

$$\mathbf{H} = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix},$$

where s is the uniform scale factor, $\mathbf{R}$ is a rotation matrix and $\mathbf{t}$ is a translation vector. During registration they apply small changes to the generating transformation T using linearization. An image point $\boldsymbol{y} = T(\boldsymbol{x})$ is modified using the velocity vector $\boldsymbol{v}(\boldsymbol{y})$

$$T_+(\boldsymbol{x}) \approx T(\boldsymbol{x}) + \epsilon(\boldsymbol{v}(\boldsymbol{y})) = T(\boldsymbol{x}) + \epsilon(\mathbf{d} \times T(\boldsymbol{x}) + \delta T(\boldsymbol{x}) + \bar{\mathbf{d}}),$$

where $\mathbf{d} = (d_1, d_2, d_3)$, $\bar{\mathbf{d}} = (\bar{d}_1, \bar{d}_2, \bar{d}_3)$ and ϵ is a small number. This is equivalent to applying small changes to the matrix $\mathbf{H}$ of the original generating transformation T

$$\mathbf{H}_+ \approx \mathbf{H} + \epsilon\mathbf{D}\mathbf{H},$$

where the matrix $\mathbf{D}$ has the following form

$$\mathbf{D} = \begin{pmatrix} \delta & -d_3 & d_2 & \bar{d}_1 \\ d_3 & \delta & -d_1 & \bar{d}_2 \\ -d_2 & d_1 & \delta & \bar{d}_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

To show this equivalence, we rewrite $\mathbf{DH}$ as

$$\begin{pmatrix} \mathbf{D}_{33}s\mathbf{R} & \mathbf{D}_{33}\mathbf{t} \\ \mathbf{0} & 0 \end{pmatrix} + \begin{pmatrix} \mathbf{0} & \bar{\mathbf{d}} \\ \mathbf{0} & 0 \end{pmatrix}$$

where $\mathbf{D}_{33}$ is the upper-left-hand submatrix of $\mathbf{D}$. Let $\boldsymbol{x}_h = (x_1, x_2, x_3, 1)^T$ denote the homogeneous coordinates of $\boldsymbol{x}$. Consider the product $\mathbf{H}_+\boldsymbol{x}_h$

$$\begin{aligned} (\mathbf{H} + \epsilon\mathbf{D}\mathbf{H})\boldsymbol{x}_h &= \mathbf{H}\boldsymbol{x}_h + \epsilon\left(\begin{pmatrix} \mathbf{D}_{33}s\mathbf{R}\boldsymbol{x} + \mathbf{D}_{33}\mathbf{t} \\ 0 \end{pmatrix} + \begin{pmatrix} \bar{\mathbf{d}} \\ 0 \end{pmatrix}\right) \\ &= \mathbf{H}\boldsymbol{x}_h + \epsilon\left(\begin{pmatrix} \mathbf{D}_{33}\boldsymbol{y} \\ 0 \end{pmatrix} + \begin{pmatrix} \bar{\mathbf{d}} \\ 0 \end{pmatrix}\right) \\ &= \mathbf{H}\boldsymbol{x}_h + \epsilon\begin{pmatrix} \mathbf{d} \times \boldsymbol{y} + \delta\boldsymbol{y} + \bar{\mathbf{d}} \\ 0 \end{pmatrix} \end{aligned}$$

with $\boldsymbol{y} = \mathbf{T}(\boldsymbol{x})$. The equivalence is shown, since $\mathbf{H}_+\boldsymbol{x}_h$ is the homogeneous coordinates of the point $T_+(\boldsymbol{x})$. They linearize iterated transformations by omitting terms of order higher than 2 in the expression $\mathbf{H}_+^k \approx (\mathbf{H} + \epsilon\mathbf{D}\mathbf{H})^k$ and obtain

$$\mathbf{H}_+^k \approx \mathbf{H}^k + \epsilon(\mathbf{D}\mathbf{H}^k + \mathbf{H}\mathbf{D}\mathbf{H}^{k-1} + \cdots \mathbf{H}^{k-1}\mathbf{D}\mathbf{H}) + \mathcal{O}(\epsilon^2).$$

Assume that the previous step of the algorithm has related patch P_i to the patch P_j by the transformation T^k. Each such pair (P_i, P_j) contributes the following term Q_{ij} to the objective function to be minimized

$$Q_{ij} = \sum_l \left([(\mathbf{T}_+^k(\boldsymbol{x}_l) - \boldsymbol{y}_l) \cdot \boldsymbol{n}_l]^2 + \mu[\mathbf{T}_+^k(\boldsymbol{x}_l) - \boldsymbol{y}_l]^2 \right),$$

where the patch P_i is represented by the sample points $\boldsymbol{x}_l$ and $\boldsymbol{y}_l$ is that point in P_j that is closest to $\mathbf{T}(\boldsymbol{x}_l)$. Moreover, $\boldsymbol{n}_l$ denotes the unit normal vector of P_j at $\boldsymbol{y}_l$. The function Q_{ij} is a combination of point-to-point and point-to-plane distances. The final quadratic function to be minimized is the sum over all patch pairs (P_i, P_j) that the previous step has related by the transformation T^k, that is

$$F(\epsilon, \mathbf{D}) = \sum_{i,j} Q_{ij}.$$

After solving the resulting linear system of equations, the modified generating transformation $\mathbf{T}_+$ is replaced by a true similarity transformation using the method of Horn [15]. The simultaneous registration algorithm iterates between the minimization of the objection function $F(\epsilon, \mathbf{D})$ and the projection to the seven dimensional space of similarity transformations. We have introduced one-parameter structures. For the case of a two-parameter regular structure, we refer to [21].

7 Detection of Patches Generated by Rigid Body Motions

In this chapter, we describe our implementation to detect geometric patterns of patches generated by one-parameter rigid body motions. A one-parameter rigid body motion $m(u)$ maps points $\boldsymbol{x} \in \mathbb{R}^3$ according to

$$\boldsymbol{y}(u) = R(u)\boldsymbol{x} + \boldsymbol{a}(u) = m(u)(\boldsymbol{x}),$$

with a rotation matrix $R(u)$ and a translation vector $\boldsymbol{a}(u)$. From the images shown below, the reader may get a better idea on what kind of patterns we consider with our method.

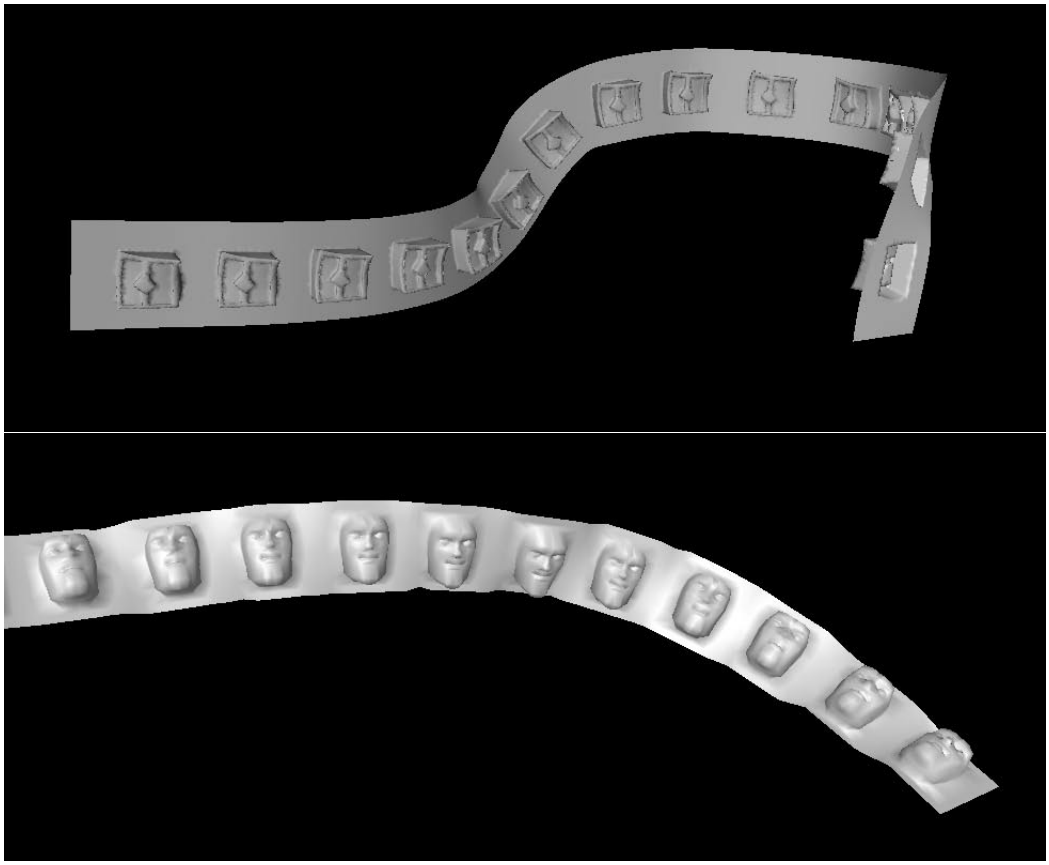

Figure 7.1: Examples of geometric patterns generated by one-parameter rigid body motions.

Before going into detail, we give an overview of our algorithm. As we mentioned in the abstract, our algorithm requires some user interaction to select one of the repetitive patches[1]. Let S denote the selected patch. Discovery of the set of other repetitive patches $\mathcal{P} = \{P_0, P_1, \cdots, P_{n-1}\}$ is equivalent to estimate a set of rigid body transformations $\mathcal{T} = \{T_0, T_1, \cdots, T_{n-1}\}$ such that for every $i \in \{0, 1, \cdots, n-1\}$ the transformed patch $T_i(S)$ is in best alignment with the patch P_i. Due to noise in the model and the sampling based approach of our method, it's rarely the case that we obtain the complete set of patches $\mathcal{P}$ and the corresponding set of transformations $\mathcal{T}$ by applying registrations. In order to detect the missing repetitive patches, we compute a smooth rigid body motion $m(u)$ that interpolates the N positions[2] $S_i := S(u_i)$ of the patch $S \subset \mathbb{R}^3$ at parameter instances u_i (comp. [14]). The challenging part of our method is to compute appropriate parameter values u_i. After computing the rigid body motion $m(u)$ we uniformly sample the parameter interval of $m(u)$ and apply for each sampled parameter value u_j registration of the patch S to the input mesh with the initial transformation $m(u_j)$. Thus we detect another $M \geq 0$ patches, which couldn't be obtained before. We recompute the rigid body motion that interpolates the $N + M \leq n + 1$ positions of the patch S and use the reconstructed motion to generate 3D geometric texture.

Our implementation to detect geometric patterns of patches generated by one-parameter rigid body motions involves the following steps:

1. Selection of an arbitrary repetitive patch
2. Sampling of the objects and pruning of the umbilical points
3. Pairing of the sample points
4. Pruning of the sample point pairs
5. Detection of the repetitive patches

Before describing these steps in detail, we begin by presenting the data structure that we used to represent the 3D triangular input meshes.

7.1 Data Structure used in Algorithm

There are several data structures that can be used to represent 3D triangular meshes on a computer. A well-known data structure stores a table of vertices along with their coordinates and for each face pointers to its vertices. This representation of 3D meshes requires little memory, but most operations such as collecting all adjacent faces for a vertex or finding the adjacent faces to an edge are inefficient. In order to perform these operations, one needs to search the whole face table.

[1] In our notation there are altogether $n + 1$ repetitive patches.
[2] N input positions of the interpolation algorithm are the detected patches and the patch S itself.

Vertex	Coordinates		
v_1	x_1	y_1	z_1
v_2	x_2	y_2	z_2
v_3	x_3	y_3	z_3
v_4	x_4	y_4	z_4

Face	Vertex		
f_1	v_1	v_2	v_3
f_2	v_1	v_3	v_4

Figure 7.2: Vertex and face tables

The half-edge data stucture efficiently solves these problems by splitting each edge into two half-edges, where each half-edge points to its opposite half-edge. The data structure consists of vertex, face and half-edge records, which are described in the following:

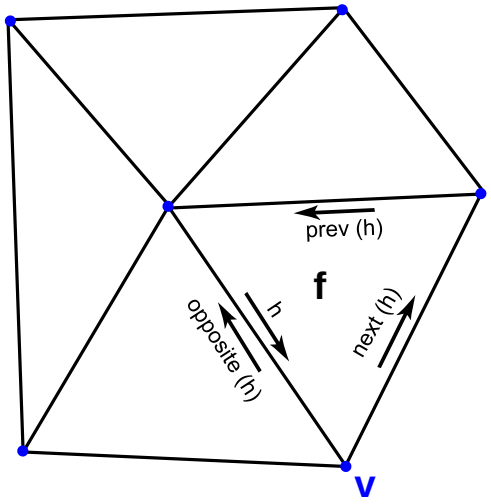

Figure 7.3: Illustration of the half-edge data structure

Vertex v

- Coordinates
- Pointer to one incident half-edge, which points to that vertex

Face f

- Pointer to one of the incident half-edges h, next(h) or prev(h)

Half-edge h

- Pointer to the vertex v
- Pointer to the incident face f
- Pointer to the half-edge next(h) on the boundary of its incident face f
- Pointer to the half-edge prev(h) on the boundary of its incident face f
- Pointer to its opposite half-edge opposite(h)

With this data structure used, operations concerning topological relations between primitives (vertices, edges, faces) are performed efficiently at the expense of a higher memory requirement. Collecting all adjacent faces for a vertex is now equivalent to iterating over the circular sequence of half-edges pointing to that vertex and collecting the incident faces of the half-edges. We use the Computational Geometry Algorithm Library, CGAL[3] based on half-edge data structure, to manage and organize the geometric primitives of the 3D triangular input meshes of our software. Our triangular input meshes can contain only regular and boundary edges. Regular edges are adjacent to exactly two faces, while boundary edges are adjacent to exactly one face. We don't consider meshes containing singular edges, which are adjacent to more than two faces.

7.2 Selection of an Arbitrary Repetitive Patch

Our implementation for the discovery of geometric patterns of repetitive patches generated by rigid body motions is at the moment not fully automatic and requires some user interaction to select one of the repetitive patches. The selection is performed using OpenGL's[4] picking routine. To do the selection, OpenGL [27] creates a projection matrix that restricts drawing to a rectangular region of the viewport. This region of the viewport is determined by the user's mouse press and release events. With this special projection matrix used, OpenGL finds out, which triangles of the 3D triangular input mesh are drawn in this region. In our implementation, we give the user more flexibility by allowing repeated selection and unselection of the already selected triangles. In Figure 7.4, two screenshots of our software are shown, where the selection step of our implementation can be seen.

After selecting one of the repetitive patches of the geometric pattern, we separate the selected patch from the input mesh and obtain two meshes. From now on we denote the selected patch as the source model and the complement as the target model (see Figure 7.5).

[3] www.cgal.org
[4] www.opengl.org

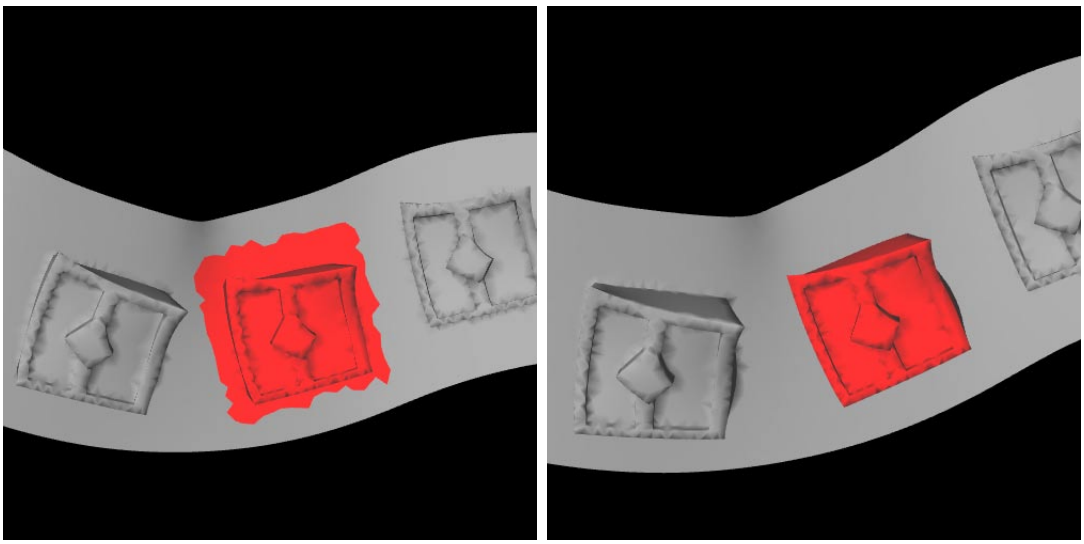

Figure 7.4: On the right image, the repetitive patch has been selected more carefully.

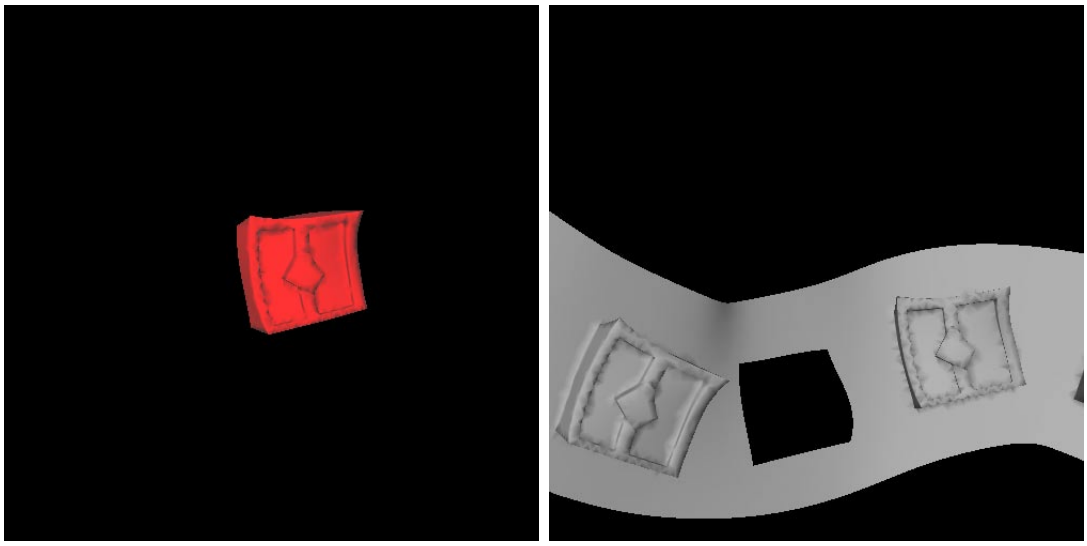

Figure 7.5: Source and target model

7.3 Sampling of the Models

We compute a uniform random sampling of the source and the target model by following the sampling approach proposed in [20]. Since the number of sample points is essential for the accuracy of the algorithm, it's given interactively by the user. By specifying the number of sample points, the user can trade accuracy for computational efficiency.

For each sample point $\boldsymbol{s}_i$ we estimate principal curvatures $\mathbf{k}_{i,1} \leq \mathbf{k}_{i,2}$ and principal directions $\boldsymbol{e}_{i,1}$ and $\boldsymbol{e}_{i,2}$ by applying the algorithm proposed in [1]. The three unit vectors $\boldsymbol{e}_{i,1}, \boldsymbol{e}_{i,2}$ and $\boldsymbol{n}_i = \boldsymbol{e}_{i,1} \times \boldsymbol{e}_{i,2}$ form a local right-handed frame. We can obtain four different right-handed local frames depending on the signs of the unit vectors $\boldsymbol{e}_{i,1}, \boldsymbol{e}_{i,2}$.

We consider two of these frames for which the inequality $\boldsymbol{n}_i \cdot \boldsymbol{n}_{s_i} \geq 0$ holds, with the outward pointing unit surface normal $\boldsymbol{n}_{s_i}$ at the sample point $\boldsymbol{s}_i$.

Similar to Mitra [21], we do not consider umbilical points, since at these points the principal directions are not defined uniquely. We prune sample points $\boldsymbol{s}_i$ with

$$\frac{\mathbf{k}_{i,1}}{\mathbf{k}_{i,2}} > \tau,$$

where the parameter $\tau < 1$ is given interactively by the user.

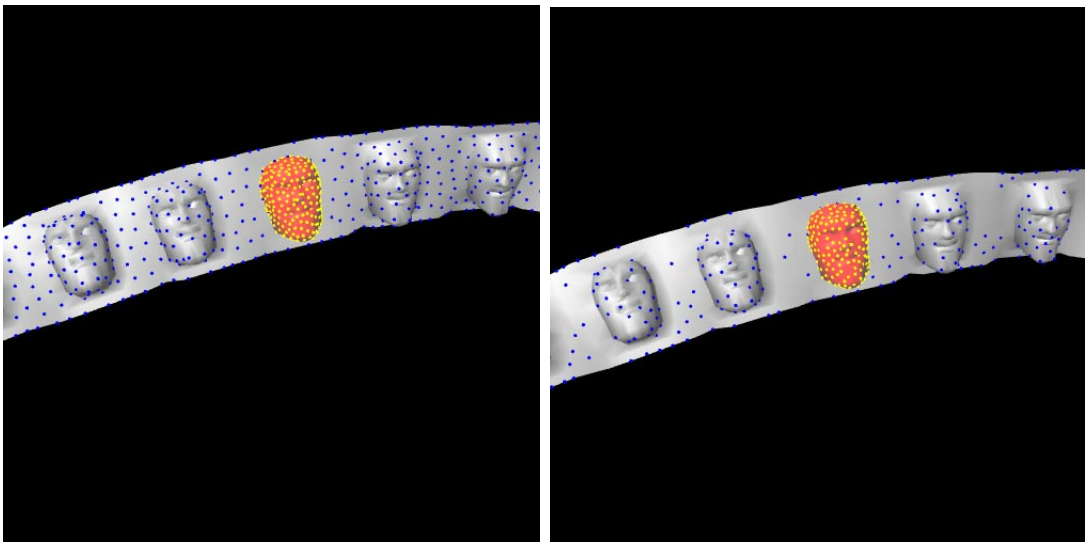

Figure 7.6: Sample points before and after pruning of the umbilical points

7.4 Pairing of the Sample Points

In this step of our method, sample points of the source model are paired with the sample points of the target model. To obtain an efficient pairing algorithm, we map all sample points $\boldsymbol{s}_i$ to the two dimensional space Ω of principal curvatures via $\sigma(\boldsymbol{s}_i) = (\mathbf{k}_{i,1}, \mathbf{k}_{i,2})$ and determine for each sample point of the source model all partners in the target model by performing a range query in Ω, where a user-defined parameter determines the range query radius. Since principal curvatures are invariant under rigid body transformations, sample point pairs that are close in Ω are considered as potential candidates for a triple (S, P, T), where S is the source model, P is another repetitive patch and T is a rigid body transformation such that the transformed patch $T(S)$ is in best alignment with the patch P. The alignment error is defined as the weighted sum of squared distances of the points $x_j \in T(S)$ to the closest points in P. Pairing is performed efficiently using a kd-tree [2], which is a standard spatial proximity data structure.

7.5 Pruning of the Sample Pairs

As we mentioned in the preceding section each determined sample pair is a potential candidate for the existence of a repetitive patch P and a rigid body transformation T such that the transformed patch $T(S)$ is in best alignment with the patch P. In the next section we will compute for each sample pair global registration of the source model to the target model in order to detect repetitive patches and corresponding rigid body transformations. This would be much time consuming, if one obtains many sample pairs in the pairing step of our method. To improve the computational efficiency and prune out incorrect matches of sample pairs, we perform pruning of sample pairs based on alignment error of local surface patches. A user-defined parameter h defines the size of local surface patches of sample points. This is illustrated in the figure shown below.

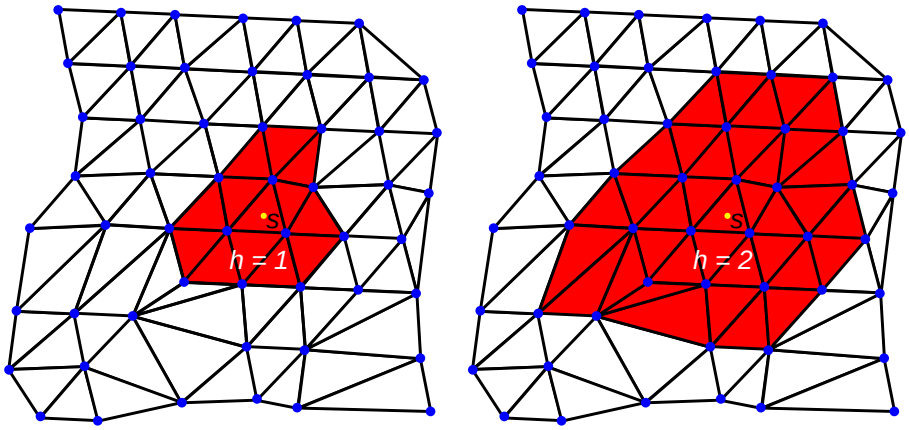

Figure 7.7: The local surface patch of the sample point s (yellow) is easily determined by using circulators in the CGAL. Circulators are used for accessing the circular sequence of half-edges around a vertex or around a face.

Let P_{s_i} denote the local surface patch of the sample point $\boldsymbol{s}_i$. For each sample pair $(\boldsymbol{s}_i, \boldsymbol{s}_j)$ we estimate the transformation T_{ij} that maps P_{s_i} onto P_{s_j}. The rotational component R_{ij} of T_{ij} is obtained by aligning the principal frames of the sample points. We adjoin the column vectors to form the orthogonal matrices R_{s_i} and R_{s_j} as follows:

$$R_{s_i} = |\boldsymbol{e}_{i,1}\boldsymbol{e}_{i,2}\boldsymbol{n}_i|, \quad R_{s_j} = |\boldsymbol{e}_{j,1}\boldsymbol{e}_{j,2}\boldsymbol{n}_j|.$$

It's easy to show (comp. [15]) that the rotation matrix that maps $\boldsymbol{e}_{i,1}$ into $\boldsymbol{e}_{j,1}$, $\boldsymbol{e}_{i,2}$ into $\boldsymbol{e}_{j,2}$ and $\boldsymbol{n}_i$ into $\boldsymbol{n}_j$ is given by

$$R_{ij} = R_{s_j} R_{s_i}^T.$$

The matrix R_{ij} is orthogonal, since the matrices R_{s_i} and R_{s_j} are orthogonal. The translational component is computed by aligning the rotated sample point $\boldsymbol{s}_i$ with the sample point $\boldsymbol{s}_j$:

$$\boldsymbol{t}_{ij} = \boldsymbol{s}_j - R_{ij}\boldsymbol{s}_i.$$

We refine this initial transformation using geometric registration. We employ point-to-point ICP algorithm of P.Besl and N.D. McKay [3]. Since we consider two different local frames for each sample point, there are several possibilities for the estimation of the transformation T_{ij} that maps the local surface patches onto each other. We take the one, which gives the least weighted sum of squared distances of the points $\boldsymbol{x}_l \in T_{ij}(P_{s_i})$ to the closest points $\boldsymbol{y}_i \in P_{s_j}$. We prune the sample pair $(\boldsymbol{s}_i, \boldsymbol{s}_j)$, if the alignment error of the registration is above a user-defined threshold.

7.6 Detection of the Repetitive Patches

As we mentioned before we compute for each remaining sample point pair $(\boldsymbol{s}_i, \boldsymbol{s}_j)$ registration of the source model to the target model. We implemented both point-to-plane ICP using instantaneous kinematics as proposed in [25] and point-to-point ICP algorithm of P.Besl and N.D. McKay [3]. The initial transformation of each registration algorithm is estimated as in the pruning step by aligning the principal frames of the sample points. The closest points on the target model are computed efficiently using a kd-tree [2]. We also implemented the weighting schemes described in chapter 5 to downweight outliers in noisy models. The figure below shows some results of our implementation of the registration algorithm.

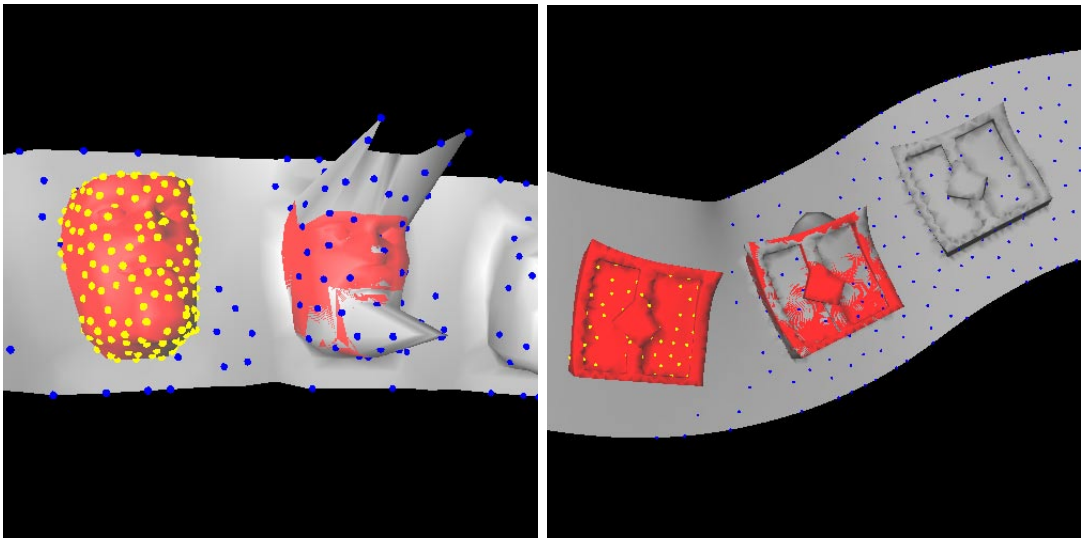

Figure 7.8: Registration of the patches to distorted copies of themselves.

As a result of applying registrations of the source model to the target model we get a set $\tilde{\mathcal{T}}$ of rigid body transformations. We discard a transformation $\tilde{T}_i \in \tilde{\mathcal{T}}$, if $\tilde{T}_i(S)$ does not produce a good alignment with the target model. The alignment error is defined as the weighted sum of squared distances of the points $\boldsymbol{x}_j \in \tilde{T}_i(S)$ to the closest points in the target model. The resulting set $\bar{\mathcal{T}}$ of rigid body transformations is not necessarily unique. In other words, the squared distance of some $\bar{T}_i, \bar{T}_j \in \bar{\mathcal{T}}$ can be very small. We group the transformations according to their squared distances from each other and

take only one transformation from each group. The squared distance between two affine maps α and β that are applied to a moving body S, which is represented by feature points $\mathbf{f}^1, \cdots, \mathbf{f}^K$, is defined as sum of squared distances of feature point positions after application of α and β, respectively (comp. [13]),

$$d^2(\alpha(S), \beta(S)) := \sum_{i=1}^{K} \|\alpha(\mathbf{f}^i) - \beta(\mathbf{f}^i)\|^2. \tag{7.1}$$

We describe the source model S by the four feature points described in the following. Let $\mathcal{T}' = \{T_{i_1}, \cdots, T_{i_{N-1}}\} \subseteq \mathcal{T}$ denote the resulting unique set of rigid body transformations and $\mathcal{P}' = \{P_{i_1}, \cdots, P_{i_{N-1}}\} \subseteq \mathcal{P}$ the corresponding set of repetitive patches such that for every $i_j \in \{i_1, \cdots, i_{N-1}\}$ the weighted sum of squared distances of the points $x_k \in T_{i_j}(S)$ to the closest points in the repetitive patch P_{i_j} is below the user-defined threshold. Due to noise in the model and variations in the sample point positions we can rarely detect all repetitive patches, which are present in the model, by applying registrations.

In the following our goal is to detect the missing $n + 1 - N \geq 0$ repetitive patches, which couldn't be obtained by applying registrations. In order to do that, we want to compute a rigid body motion $m(u)$ which interpolates the N positions $S(u_i)$ of the patch $S \subset \mathbb{R}^3$ such that chosen feature points of the patch S run on smooth paths. We start by choosing an arbitrary orthonormal right-handed frame $f_S = (\mathbf{x}_S, \mathbf{y}_S, \mathbf{z}_S)$ originating at the centroid $\mathbf{c}_S$ of the source model and determine four feature points $\mathbf{f}^1, \cdots, \mathbf{f}^4$ of the source model S as follows:

$$\mathbf{f}^1 = \mathbf{c}_S, \quad \mathbf{f}^2 = \mathbf{c}_S + \mathbf{x}_S, \quad \mathbf{f}^3 = \mathbf{c}_S + \mathbf{y}_S, \quad \mathbf{f}^4 = \mathbf{c}_S + \mathbf{z}_S.$$

We transform the local frame f_S for every $i_j \in \{i_1, \cdots, i_{N-1}\}$ by the transformation $T_{i_j} \in \mathcal{T}'$ and obtain the different positions of the feature points $\mathbf{f}^k, k = 1, \cdots, 4$. The N different positions[5] $\mathbf{f}^k_0, \cdots, \mathbf{f}^k_{N-1}$ of the same feature point $\mathbf{f}^k$ are called as homologous points[6] (see Figure 7.9). To each of the 4 sequences of homologous points we will apply the same interpolating curve design algorithm. Assume that parameter values are given, we want to find B-Spline curves $\mathbf{f}^k(u)$ with

$$\mathbf{f}^k(u_i) = \mathbf{f}^k_i, \qquad i = 0, \cdots, N-1, \quad k = 1, \cdots, 4.$$

The most challenging part of our method is the computation of appropriate parameter values u_i, which is described in the following. We aim to approximate the first sequence $\mathbf{f}^1_0, \cdots, \mathbf{f}^1_{N-1}$ of homologous points by a B-Spline curve that reflects the form of the curve on which the centroids of all available repetitive patches lie. A B-Spline curve is defined by the formula

$$P(t) = \sum_{i=0}^{m} B^n_i(t) d_i,$$

[5] $N - 1$ transformed and the original position of the feature point
[6] comp. [14]

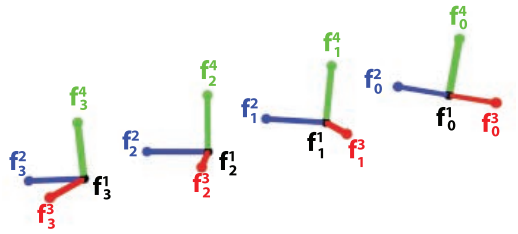

Figure 7.9: Homologous points

where $B_i^n(t)$ describes the basis B-Spline functions of degree n and d_i the control points. We use cubic basis B-Spline functions for the approximation. Additionally, the knot sequence and the number of control points are not subject to the optimization. We use the following algorithm to approximate the sequence of homologous points by a B-Spline curve.

Algorithm 7.1. *An algorithm for the approximation of a point cloud X by a B-Spline curve $P(t)$ (comp. [9]).*

1. *Define a suitable start position of the approximating B-Spline curve.*
2. *Assign each point $\boldsymbol{x}_k \in X$ a parameter value t_k such that $P(t_k)$ is the foot point of $\boldsymbol{x}_k$ on the approximating B-Spline curve.*
3. *Find a displacement $c = (c_i)_{i=0,\cdots,m}$ of the control points d_i*

$$P_c(t) = \sum_{i=0}^{m} B_i^n(t)(d_i + c_i)$$

such that the following quadratic objective function is minimized

$$g(c) = \sum_{k=0}^{n} d^2(P_c(t_k), \boldsymbol{x}_k) + F_r(c),$$

where $d^2(P_c(t_k), \boldsymbol{x}_k)$ is a local approximation of the squared distance function and $F_r(c)$ is a regularization term ensuring a smooth solution.

4. *Update the control points of the approximating B-Spline curve: $d_i \to (d_i + c_i)$. If the updated B-Spline curve is a good approximation of the given point cloud X, stop.*

5. Otherwise, continue with step 2.

In [9] the steps 2 and 3 are discussed in detail. To define a suitable start position of the approximating B-Spline curve, we first compute the euclidean minimum spanning tree (EMST) of the complete graph with the vertices $\mathbf{f}_0^1, \cdots, \mathbf{f}_{N-1}^1$ (see Figure below).

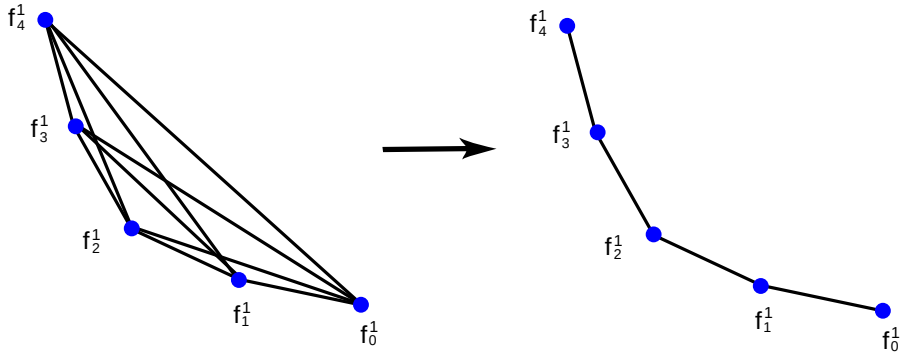

Figure 7.10: In most cases the euclidean minimum spanning tree can be used as the control polygon of the initial B-Spline curve.

Since the computed EMST can contain vertices with degree higher than two, we find the longest path $\mathbf{f}_{i_0}^1, \cdots, \mathbf{f}_{i_l}^1$ with $\mathbf{f}_{i_0}^1 \neq \mathbf{f}_{i_l}^1$ in this tree (see Figure below) and choose the vertices of this path as the control points of the initial B-Spline curve.

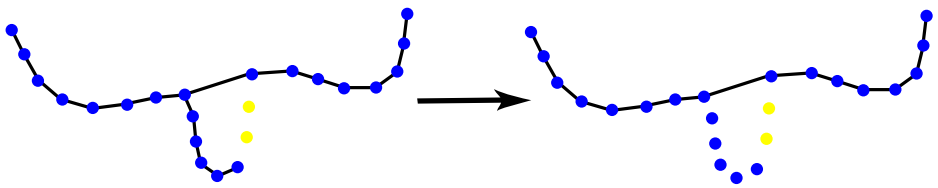

Figure 7.11: In cases where consecutive repetitive patches couldn't be detected, some vertex might have a degree higher than two. The blue points represent the centroids of the detected patches, while the yellow points represent the centroids of two undetected consecutive patches.

The sequence $\mathbf{f}_0^1, \cdots, \mathbf{f}_{N-1}^1$ of homologous points are approximated with the constraint that the end-points $\mathbf{f}_{i_0}^1, \mathbf{f}_{i_l}^1$ of the computed longest path be interpolated. Let $s(t), t \in [t_{start}, t_{end}]$ denote the final approximating B-Spline curve. For every $i \in \{0, \cdots, N-1\}$, we assign the length of the curve segment $s([t_{start}, t_i])$ to the points $\mathbf{f}_i^k, k = 1, \cdots, 4$, where $s(t_i)$ is the closest point of $\mathbf{f}_i^1$ on the approximating B-Spline curve (see figure below).

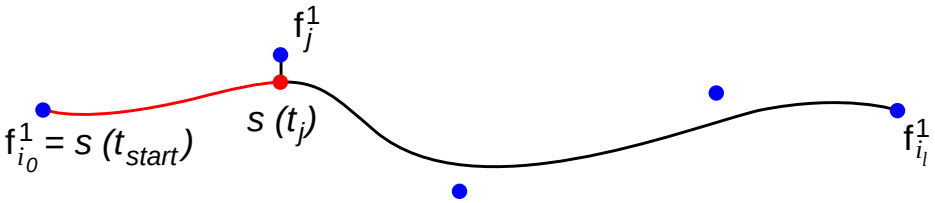

Figure 7.12: B-Spline approximation for sorting of the homologous points

We sort all four sequences $\mathbf{f}_0^k, \cdots, \mathbf{f}_{N-1}^k, k = 1, \cdots, 4$ of homologous points by ascending order of the assigned length values and get the ordered sequences $\mathbf{f}_{j_0}^k, \cdots, \mathbf{f}_{j_{N-1}}^k, k = 1, \cdots, 4$. For some $i \in \{0, \cdots, N-1\}$, $\mathbf{f}_{j_i}^k, k = 1, \cdots, 4$ denote the feature points of the source model S. For the computation of parameter values, we use the squared distance (7.1) between two affine maps. We assign the parameter value $u_{j_i} = 0$ to the feature points of the source model. The parameter values of the points

$$\mathbf{f}_{j_t}^k, \quad k = 1, \cdots, 4, \quad t > i$$

are iteratively computed as follows:

$$u_{j_t} = u_{j_{t-1}} + \sum_{k=1}^{4} \|\alpha(\mathbf{f}_{j_i}^k) - \beta(\mathbf{f}_{j_i}^k)\|^2, \quad t = i+1, \cdots, N-1,$$

where α and β are two rigid body transformations that map the feature points $\mathbf{f}_{j_i}^k$ onto the points $\mathbf{f}_{j_{t-1}}^k$ and $\mathbf{f}_{j_t}^k, k = 1, \cdots, 4$, respectively. The parameter values of the points

$$\mathbf{f}_{j_t}^k, \quad k = 1, \cdots, 4, \quad t < i$$

are computed analogously as follows:

$$u_{j_t} = u_{j_{t+1}} - \sum_{k=1}^{4} \|\alpha(\mathbf{f}_{j_i}^k) - \beta(\mathbf{f}_{j_i}^k)\|^2, \quad t = i-1, \cdots, 0,$$

where transformations α and β map the feature points $\mathbf{f}^k_{j_i}$ onto the points $\mathbf{f}^k_{j_{t+1}}$ and $\mathbf{f}^k_{j_t}, k = 1, \cdots, 4$, respectively.

After applying the same interpolating curve design algorithm to all sequences of homologous points, we get 4 curves $\mathbf{f}^1(u), \cdots, \mathbf{f}^4(u), u \in [u_{start}, u_{end}]$. For each $\bar{u} \in [u_{start}, u_{end}]$ the points $\mathbf{f}^1(\bar{u}), \cdots, \mathbf{f}^4(\bar{u})$ may be considered as affine image points of $\mathbf{f}^1, \cdots, \mathbf{f}^4$. Let $\alpha(\bar{u})$ denote this affine transformation. The curves $\mathbf{f}^1(u), \cdots, \mathbf{f}^4(u)$ determine a parameter dependent family of affine copies $S'(u) = \alpha(u)(S)$ of the source model S, a so-called affine motion (comp. [14]). Since we are interested in a rigid body motion of the source model S, we compute for each parameter value $\bar{u}$ a rigid body transformation $m(\bar{u})$, which brings the source model S as close as possible to its affine copy $S'(\bar{u}) = \alpha(\bar{u})(S)$ using the method of Horn [15]. The objective function to be minimized looks as follows:

$$\sum_i \|m(\bar{u})(\boldsymbol{v}_i) - \alpha(\bar{u})(\boldsymbol{v}_i)\|^2,$$

with the vertices $\boldsymbol{v}_i$ of the source model S. In this way we approximate the affine motion $\alpha(u)$ by a rigid body motion $m(u)$. In [14] they prove that the rigid body motion generated in this way is of the same smoothness as the interpolating curve design algorithm employed. A geometric interpretation of the approximation of an affine motion by a rigid body motion is illustrated in Figure 7.13[7].

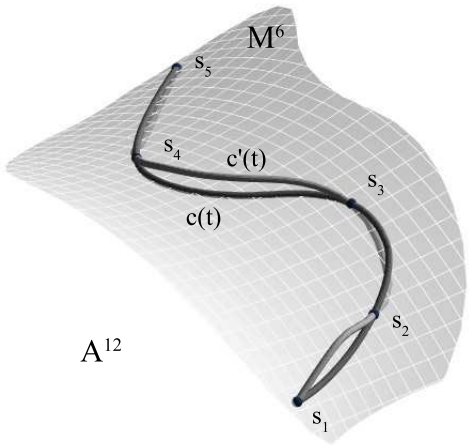

Figure 7.13: To each affine image of the source model a point in 12-dimensional affine space A^{12} is associated. The images of the source model S under rigid body motions form a 6-dimensional submanifold $\mathrm{M}^6 \subset \mathrm{A}^{12}$. The N positions of the source model S correspond to points s_i on M^6. An affine motion of the source model S interpolating the given positions s_i corresponds to a curve $c'(t)$ in A^{12} and the approximation of the affine motion by a rigid body motion to the orthogonal projection of the curve $c'(t)$ to a curve $c(t)$ in M^6.

[7] source: [14]

In order to detect the missing repetitive patches, we uniformly sample the parameter interval $[u_{start}, u_{end}]$ of the computed rigid body motion $m(u)$, obtain the set of parameter values $\{u_0, \cdots, u_{d-1}\}$ with

$$u_{start} = u_0 < u_1 < \cdots < u_{d-1} = u_{end}$$

and compute for every $j \in \{0, \cdots, d-1\}$ registration of the source model S to the target model with the initial transformation $m(u_j)$. Thus we obtain another $M \geq 0$ repetitive patches, which couldn't be detected before. The parameter d is essential for the accuracy of our method. The more the magnitude of d is, the more accurate the algorithm will be. Finally we recompute the rigid body motion that interpolates the $N + M \leq n + 1$ different positions of the source model S. The user can iterate the computation of the rigid body motion until no more improvements can be achieved.

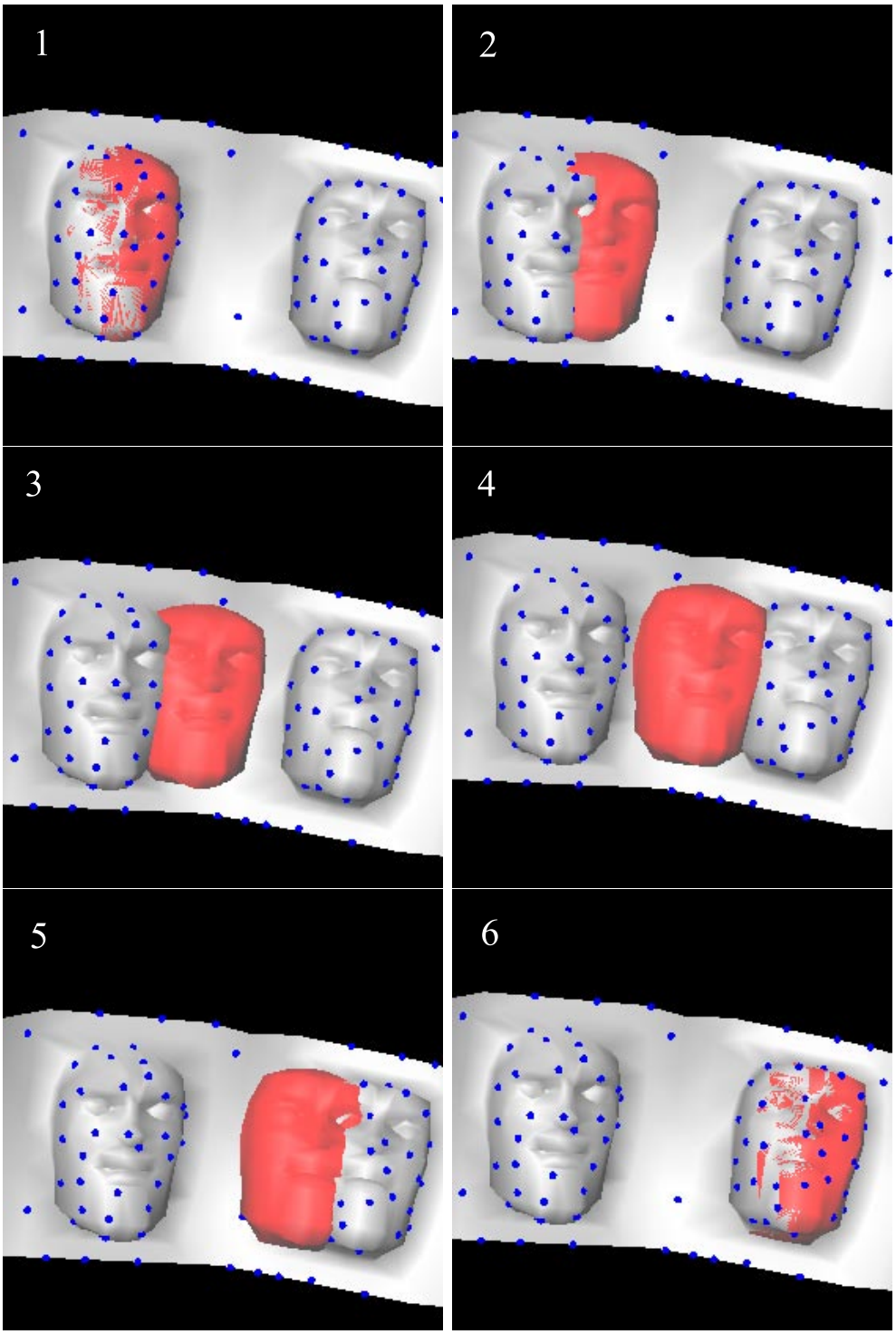

Figure 7.14: Reconstructed rigid body motion generates 3D geometric texture on the target model.

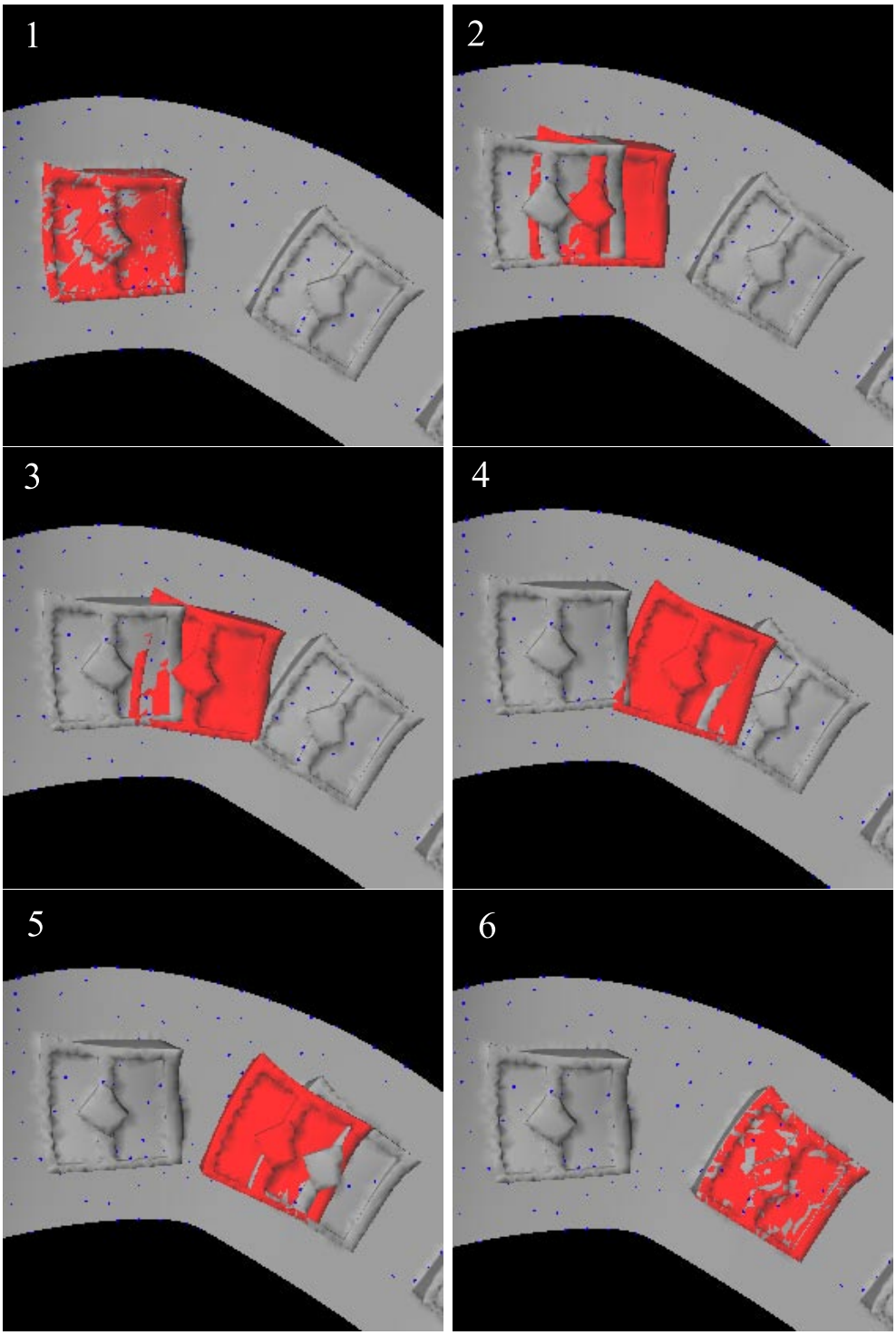

Figure 7.15: Reconstructed rigid body motion generates 3D geometric texture on the target model.

8 Conclusion

We have shown how to detect repetitive structures generated by rigid body motions in 3D mesh-based models. The user provides a 3D triangular mesh as input, selects an arbitrary repetitive structure and specifies the number of sample points. Additional parameters include a range query radius for pairing and thresholds for point pruning and registration. The output of our method is the set of repetitive patches $\mathcal{P} = \{P_0, P_1, \cdots, P_{n-1}\} \cup S$, i.e., the selected patch S, the reconstructed rigid body motion $m(u)$ and the set of parameter values $\mathcal{U} = \{u_0, u_1, \cdots, u_{n-1}\}$ such that for every $j \in \{0, \cdots, n-1\}$ the transformed patch $m(u_j)(S)$ is in best alignment with the patch P_j.

8.1 Texture Generation

Applications of our method include generation of 3D geometric texture. The images shown on the last two pages illustrate how to generate 3D geometic texture using the reconstructed rigid body motion.

8.2 Limitations

Our method does not ensure to detect all repetitive patches, which are present in the given 3D triangular mesh. An example of such a situation is illustrated below in Figure 8.1.

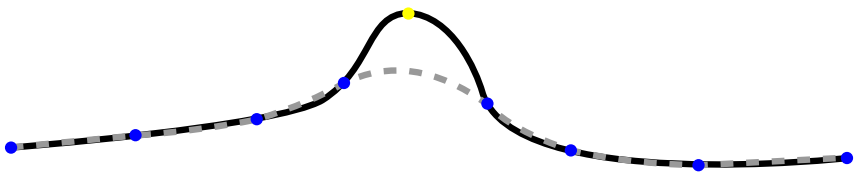

Figure 8.1: Illustration of a limitation of our method

In Figure 8.1 the blue points represent detected positions of the source model S and the yellow point a repetitive patch, which couldn't be detected by applying registrations. The solid line stands for the rigid body motion interpolating all repetitive patches, which are present in the model. The dashed line depicts the motion that interpolates the detected positions of the source model. If an undetected patch (yellow) would be too far from the rigid body motion interpolating the detected positions of the source model, it won't be possible to detect this patch by applying registration with an initial transformation computed with our method.

We don't consider similarity transformations, but our method can be extended with a minor change of the implemented registration algorithms to consider patterns of repetitive patches generated by equiform motions. A one-parameter equiform motion $e(u)$ maps points $\boldsymbol{x} \in \mathbb{R}^3$ according to

$$\boldsymbol{y}(u) = \alpha(u)R(u)\boldsymbol{x} + \boldsymbol{a}(u) = e(u)(\boldsymbol{x}),$$

with a rotation matrix $R(u)$, a translation vector $\boldsymbol{a}(u)$ and a scaling factor $\alpha(u)$. If $\alpha(u) = const = 1$, we obtain a rigid body motion.

List of Figures

Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003.

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[3] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[4] R. W. Bulterman, F. W. van der Sommen, G. Zwaan, T. Verhoeff, A. J. M. van Gasteren, and W. H. J. Feijen. On computing a longest path in a tree. *Inf. Process. Lett.*, 81(2):93–96, 2002.

[5] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 177–187, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[6] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image Vision Comput.*, 10(3):145–155, 1992.

[7] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[8] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[9] Simon Flöry. Fitting B-Spline Curves to Point Clouds in the Presence of Obstacles. Master's thesis, Technische Universität Wien.

[10] G. Godin, M. Rioux, and R. Baribeau. Three-dimensional registration using range and intensity information. In S. F. El-Hakim, editor, *Proc. SPIE Vol. 2350, p. 279-290, Videometrics III, Sabry F. El-Hakim; Ed.*, volume 2350 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 279–290, October 1994.

[11] Donald D. Hearn and M. Pauline Baker. *Computer Graphics with OpenGL*. Prentice Hall Professional Technical Reference, 2003.

[12] M. Hofer, B Odehnal, H. Pottmann, T. Steiner, and J. Wallner. 3D shape recognition and reconstruction based on line element geometry. In *Tenth IEEE International Conference on Computer Vision*, volume 2, pages 1532–1538. IEEE Computer Society, 2005.

[13] M. Hofer, H. Pottmann, and B. Ravani. Geometric design of motions constrained by a contacting surface pair. *Comput. Aided Geom. Design*, 20:523–547, 2003.

[14] M. Hofer, H. Pottmann, and B. Ravani. From curve design algorithms to the design of rigid body motions. *The Visual Computer*, 20(5):279–297, 2004.

[15] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, 4(4):629, 1987.

[16] A. Lorusso, D. W. Eggert, and R. B. Fisher. A comparison of four algorithms for estimating 3-d rigid transformations. In *BMVC '95: Proceedings of the 1995 British conference on Machine vision (Vol. 1)*, pages 237–246, Surrey, UK, UK, 1995. BMVA Press.

[17] Dennis Maier, Jürgen Hesser, and Reinhard Männer. Fast and accurate closest point search on triangulated surfaces and its application to head motion estimation.

[18] Manfredo. *Differentialgeometrie von Kurven und Flächen*. vieweg, 1993.

[19] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas. Registration of point cloud data from a geometric optimization perspective. In R. Scopigno and D. Zorin, editors, *Eurographics Symposium on Geometry Processing*, pages 23–32, 2004.

[20] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics*, volume 25, pages 560–568, 2006.

[21] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. Discovering structural regularity in 3d geometry. In *ACM Transactions on Graphics*, volume 26, page to appear, 2008.

[22] H. Pottmann and M. Hofer. Geometry of the squared distance function to curves and surfaces. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 223–244. Springer, 2003.

[23] H. Pottmann, S. Leopoldseder, and M. Hofer. Simultaneous registration of multiple views of a 3D object. *ISPRS Archives*, 34(3A):265–270, 2002.

[24] H. Pottmann and J. Wallner. *Computational Line Geometry*. Mathematics + Visualization. Springer, Heidelberg, 2001.

[25] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Registration without icp. *Comput. Vis. Image Underst.*, 95(1):54–71, 2004.

[26] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001.

[27] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition) (OpenGL)*. Addison-Wesley Professional, 2005.

[28] David A. Simon. *Fast and accurate shape-based registration*. PhD thesis, Pittsburgh, PA, USA, 1996. Chair-Takeo Kanade.

[29] Wikipedia. Kruskal's algorithm — wikipedia, the free encyclopedia, 2008. [Online; accessed 29-May-2008].

[30] Wikipedia. Prim's algorithm — wikipedia, the free encyclopedia, 2008. [Online; accessed 29-May-2008].