Università degli Studi di Palermo

Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Matematica e Informatica

- XXI ciclo -

Polygonal Mesh Segmentation by Surface Curvature Diffusion

Author

Marco Cipolla

Coordinator

Prof. Camillo Trapani

Thesis Advisor

Prof. Domenico Tegolo

Settore Scientifico Disciplinare INF/01

Polygonal Mesh Segmentation by Surface Curvature Diffusion

Abstract: One of the most popular 3D object representations in Computer Vision is the polygonal mesh, which is a sets of vertices, edges and facets having some adjacency relations. Several applications such as *shape matching*, *shape retrieval*, *3D data compression*, etc. require *mesh segmentation*, which consists in the decomposition of an object into its meaningful components. Mesh segmentation is a very hard problem since it can be reduced to the graph partition problem which is an NP-Complete. Many supervised or parameters dependent algorithms have been developed to produce sub-optimal solutions according to different paradigms such as *Region Growing*, *Clustering*, etc.

In this thesis we present a new method based on diffusion of some energy function over the surface of the object.

By miming the heat diffusion process, Surface Curvature Diffusion (SCD) classifies the vertices of a mesh by distributing the mean curvature of the object on the mesh surface. SCD uses the discretization of partial differential equations to model the diffusion of the curvature over time and it segments the mesh by analysing the trend of such a diffusion on the vertices.

SCD depends only on the initial state of the curvature and it is performed until the energy reaches the equilibrium. Then it is parameter-free and time independent.

We show some of several experiments carried out by using different kinds of meshes and we show that SCD is very fast and accurate. Moreover, it allows to rightly detect the most of the feature-edges when it is compared to other techniques present in literature. These features together with the lack of any tuning makes SCD a very interesting method for mesh segmentation.

Keywords: Three-Dimensional Polygonal Mesh, Mesh Fairing, Surface Fitting, Mesh Segmentation, Energy Diffusion, Surface Curvature Diffusion, Tensor Voting, Normal Voting.

Originality Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the test. I give consent to this copy of my thesis, when deposited in the University Library, begin available for loan and photocopying.

Signed ..... February 2011

Contents

List of Figures

Introduction

The interest of researchers in digital 3D object analysis rise up during the past decades due to the advances in computer technologies together with the birth of new fields of research. The generation, representation and manipulation of virtual objects are the principal problems in several context such as: *engineering*, *robotics*, *computer vision*, *medicine*, *molecular biology*, *entertainment*, etc.

The generation of 3D virtual objects can be accomplished by either specific software or particular acquisition devices which allow to generate virtual objects from the real world. These devices are able to gather the all the information of the objects in order to produce 3D data which is typically represented either by *volumes* or by *cloud of points*. In this thesis we are only interested in 3D data obtained by some laser scanner acquisition device, more precisely we focus on polygonal meshes which are a particular representation of the surface of the objects by means of graph-type structures whose nodes are the points acquired from the real world.

There are many applications relying on the analysis of three-dimensional meshes. For instance, the mesh representation is widely used in *Computer Gaming* and in other field of entertainment. In these contexts it is important to simulate physical phenomena involving 3D objects efficiently. Shape retrieval is another important application which exploits largely the mesh representation. Indeed, it requires to match an object against some given model in order to retrieve from a database, all the objects having *similar* shape to the input one. The key problem of many application involving 3D representation is the mesh partitioning in its meaningful (*semantic*) components. The *meaning* of a component highly depend on the type of application, thus there no exists a unique way to perform the object segmentation. A very popular segmentation exploits the discontinuities of the surface, called feature-edges, to define the different parts of the object.

Mesh decomposition is a very hard task. Starting from a mesh it is possible to define its dual graph, and the mesh segmentation problem is equivalent to the graph partition problem which is an NP-Complete problem. Hence, we need to resort to approximate solutions.

Several mesh segmentation methods are present in literature. Many algorithms, according to different paradigms such as *Region Growing*, *Clustering*, etc. produce sub-optimal solutions. Unfortunately most of these techniques can not be easily embedded in automatic segmentation systems, because they require the tuning of some parameters or threshold levels, to produce a significant segmentation. Thus, the search for parameter-free methods is particularly interesting.

Mesh Segmentation

The surface of an object is decomposed into segments according to some specific problem to be solved. Different applications require different type of segments with

different shapes and properties, and segmentation algorithms can be generally classified, according to their goals, into one of the following categories:

- **Surface Type Segmentation Algorithms**. Applications as *texture mapping*, *morphing*, *mesh simplification*, and *mesh compression* require to decompose the object into small regions satisfying some criteria. For instance, regions may present constant curvature or they could match some surfaces primitive (cylinders, spheres, etc.).

- **Component Type Segmentation Algorithms**. Several applications need to *understand* the object shape to fulfil tasks as *shape matching*, *shape retrieval*, *object reconstruction*, *collision detection*, etc. This class of mesh segmentation algorithms decompose the objects into their *meaningful parts*. For example, an object representing a human hand can be decomposed into its fingers and palm.

Mesh segmentation problem is an NP-Complete problem and different approximate solutions have been proposed, and the three principal segmentation methodologies reported in literature are:

- **Region Growing Based Methods**. This class of algorithms segment an object by growing regions starting from *seed elements*, where these seed elements can be chosen in different ways and the region growing process is ruled by the underlying geometry of the mesh surface.

- **Clustering Based Methods**. Clustering is widely used in several problems of data analysis. Here, clustering methods are used to perform segmentations by merging the mesh elements into regions, according to some cost function based on the local geometrical properties of the surface.

- **Spectral Analysis**. The *eigen* analysis of the *Laplacian* matrix of some graph associated with the mesh, is used for mesh compression purposes. Each entry in the matrix encodes the probability that two elements belong to the same segment.

Thesis Contributions

In the context of mesh segmentation based on the local surface analysis, we proposed two methods to locate the *feature edges* over the surface of an objects. Both methods first assign a weight to each edge of the mesh, where an edge is the line joining two vertices of the triangulation which describe the mesh. A neighbourhood of each edge is then considered. Such neighbourhood is decomposed into disjoint layers, according to the distance from the central edge. The first method analyses the variance of the weights within each layer and classifies the *feature edges* through linear regression. The second method defines a measure of *saliency* of each mesh element based on some fuzzy membership. The *fuzzification* process induces a segmentation of the

surface into three sets: the set of *feature edges*, the set of smooth surfaces and the set of *ramps*, namely the region of the surface close to *feature edges*.

In this thesis we propose the Surface Curvature Diffusion (SCD), which is an automatic mesh segmentation method based on the diffusion of some energy function defined over the surface of the mesh. The key idea is to reproduce the physical phenomenon of the heat diffusion, through the distribution of the mean curvature of the object over its surface. As the physical process acts in the continuous case, SCD solves the problem of curvature diffusion by means of discretized partial differential equations depending on both the spatial coordinates of the points and time.

At the initial time ($t = 0$) the energy function coincides, point by point, with the mean curvature of the object. The algorithm tracks the evolution of the energy over all the points of the surface and it classifies the object vertices according to the variation of their energy. At the end of the process ($t = t_{max}$), feature-edges are characterized by a large loss of energy released to their neighbouring points. While, points lying on smooth surfaces increase their energy.

Finally, by using the *local energy variation*, SCD defines an *height map* and applies a region growing based algorithm to locate the object components.

SCD uses as input data obtained by range images acquired by some devices, and it is both parameters-free and time independent, because the diffusion process is related only to the shape of the object and always terminates at equilibrium. Furthermore, as the proposed algorithm simulates an adiabatic process, the total curvature of the object is preserved, as in the physical phenomenon.

Surface *fairing* is a very important tool in mesh processing. Noise is typically suppressed by moving the vertices along the normals to the surface with speed equal to the mean curvature (*mean curvature flow*). On the contrary, SCD is able to perform an effective surface de-noising only by measuring the total variation of the vertices energy.

Several experiments on different kind of meshes show that SCD is robust, fast, accurate and efficient.

Thesis Outline

This thesis is organised as follows:

- Chapter 1 describes the most important 3D data generation methods. In particular we focus on *range data* representation. Here, we also define the boundary mesh representation as a set of vertices, edges and convex polygons with some adjacency relation. Furthermore, we briefly discuss about the principal mesh generation techniques.

- In Chapter 2 we discuss about the principal algorithms reported in literature to estimate differential properties of surfaces and we also introduce other surface descriptors based on *Tensor Voting*. Moreover, the basic principles of mesh de-noising are discussed.

- In Chapter 3 we summarize the state of the art of the mesh segmentation methods. In particular, here we explain *Region Growing*, *Clustering* and *Spectral Analysis* methods. In this chapter we also focus on both the *Watershed Transform* and mesh edge-detection algorithms developed so far. Finally, we discuss the *Normal Voting* approach.

- Chapter 4 introduces diffusion based image processing and it presents our SCD method. Some of the experiments carried out to validate the effectiveness of the proposed algorithm are also reported. Moreover, the SDC results are compared to the outcome of some of the most important edge-detection algorithms present in literature and future works are finally proposed.

Chapter 1

Three-Dimensional Objects

The representation of a three-dimensional object by a computer requires a *virtualization* process, typically performed by the generation through a model, or by acquisition of a *real-world object*, or by a mix of both techniques. 3D digital objects are usually divided into two categories: *solid*, where the objects are represented by a volume and *boundary*, where the objects are represented by a surface.

Since our work relies on *triangular meshes*, which are a particular type of boundary objects, in this chapter we will focus on the boundary category.

1.1 Three-Dimensional Objects Generation

3D modelling and computer gaming focuses mainly in generating of 3D models through specialized software like *3D graphical engines* and libraries for computer programs. The *rendering* of objects for the generation of very realistic 3D scenes and the interaction with *virtual 3D worlds* requires to simulate different physical phenomena as *collisions* and *motion*. Furthermore, features like *lighting* and *texturing* are widely used. Modelling is also performed in *Computer Aided Design (CAD)* for engineering purposes. A very important tool used in CAD problems is represented by *Non-Uniform Rational B-Splines (NURBS)* [51], which provide some representations of 3D geometry and are able to describe every shape, from a simple line to a very complex organic structure.

1.2 Three-Dimensional Objects Acquisition

3D virtual objects can be also obtained from the real-world by using some kinds of *acquisition systems* [67]. Different devices are able to capture the 3D shape of a real object, like a camera captures a 2D snapshot of a real scene.

Data acquisition is a necessary step to represent a real 3D object through a computer. This process gathers all the spatial information of the object by returning a collection of data that can be easily managed by an automatic system.

There exist several real 3D objects acquisition devices, and different systems produce different virtual object representations, each one with own advantages and disadvantages about memory requirement, processing simplicity and level of object details.

The main data types are: *range data* and *volumetric data*. The former is generally obtained by active range sensor systems, the latter by tomography systems, ultrasounds, satellite terrain mapping systems, Magnetic Resonance Imaging, etc.

A general range sensor system is composed by a laser device and a camera. The acquisition process uses a beam of laser light to hit the object and the measure of the light reflected by a point on the object is used to compute the spatial coordinates of such a points. The object is scanned from different point of view and the resulting images must be merged together in order to have a full representation. Figure 1.1 shows the typical range sensor system scheme. A plane-type beam of light intersects the object in P, by measuring the distance between P and the camera (*depth of the point*) it is possible to *map* the 3D coordinates of P on P' belonging the 2D camera-space. The distance between P and P' gives the depth information of P. This process returns an image (*range image*), where the point depth replaces the pixels brightness intensity information.

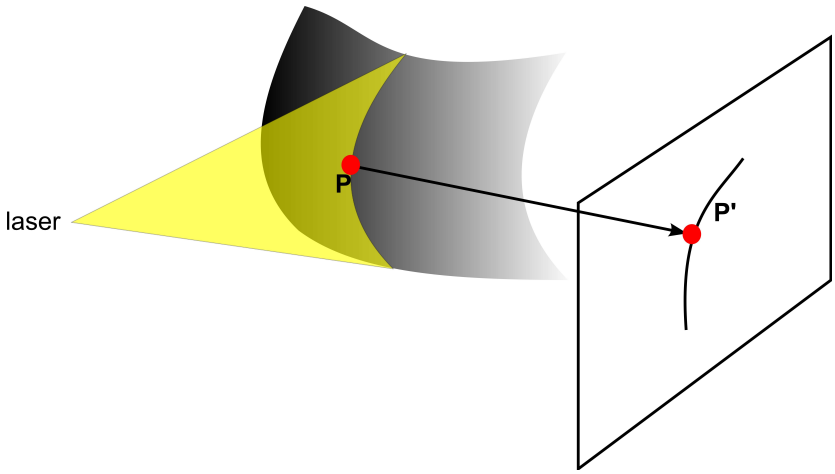

Figure 1.1: Schema of a general range sensor system.

Range images are a special class of digital images and are generally represented in two forms. The first is a list of 3D coordinates points with no specific order, usually denoted as *cloud of points*; while the second one is a matrix of *depth values* with explicit spatial organization, i.e. a matrix $A = a_{ij}$ where $a_{ij} = z(i, j)$ indicates the depth information of the point with coordinates (i, j). Figure 1.2 shows an example a two range images of the surfaces of a sculpture.

Volumetric acquisition systems acquire data by shooting the object by penetrating rays (such as X-rays). Different materials absorb different rate of radiation and, by measuring the exiting beam it is possible to obtain a set of cross-sectional 2D images (*slices*). The whole shape of the object is reconstructed by stacking the output images through the process of *voxelization*, see Figure 1.3.

Volumetric data are represented by a set of voxels. The *voxel* (**vo**lume pi**xel**) is the smallest unit in 3D volumetric data. The voxel can be represented by either a box-shaped volume or a sample point on a regular 3D grid, and it stores all the available information about the object features in that *volume* (i.e. colour, opacity, gray level, labels, etc.).

In the case of a 3D lattice a volumetric image is a subset of $\mathbb{Z}^3$, voxels are cubes

Figure 1.2: Two range images (*left and center*) representing two different points of view of a sculpture, with the reconstructed shape (*right*)

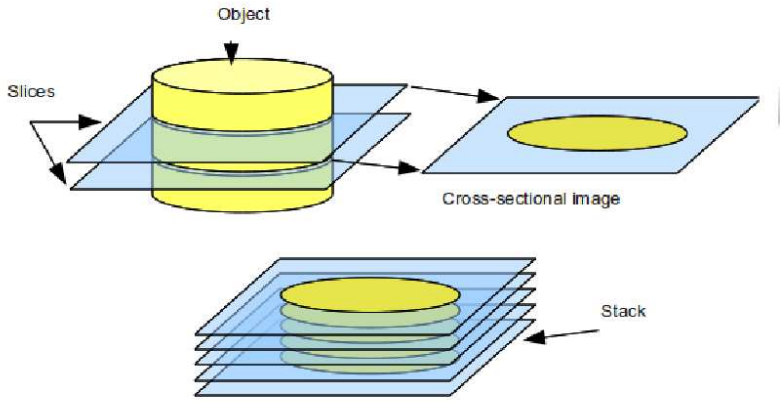

Figure 1.3: Voxelization schema.

with edges of unitary length, arranged as in Figure 1.4, and . Note that two voxels might share a surface, an edge or a vertex determining three kind of connectivity.

Objects data need often to be described by some mathematical model. Objects surfaces represented by range data or volumetric data can be modelled by *parametric surfaces* and *implicit surfaces*, respectively. Parametric models are usually given by a 2D to 3D mapping function f, while implicit surfaces are represented by three dimensional scalar field $f(x, y, z) = 0$ [29].

Several conversion techniques have been developed to transform a surface representation into another one. A well know method, for implicit to explicit conversion, is the Marching Cube (MC) algorithm [35], which performs a sampling of the implicit surface $f(x, y, z) = 0$ on a uniform spatial grid and considers the approximate intersections between the grid and the surface. The drawback of the method is the poor reconstruction of sharp features. Over the years, many improvement of the MC technique have been presented. In [61] the size of the triangles is adapted to

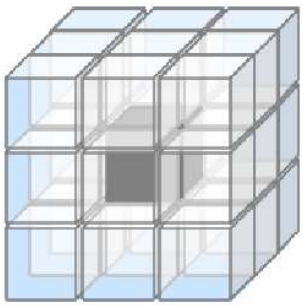

Figure 1.4: 3 x 3 three dimensional lattice. With this kind of connectivity, an internal voxel (black box) has 26 neighbours.

the shape of the object surface, while in [29] a *directed distance field* is used along the x and y directions at every grid point, resulting in a good reconstruction near sharp zones.

Several algorithms to convert from explicit to implicit forms are presented in [28], where these algorithms convert 3D geometric objects into their discrete voxel-map representation by using a *Cubic Frame Buffer (CFB)*, namely, a 3D array of voxels which stores regular volumetric datasets.

1.3 Three-Dimensional Boundary Objects Representation

Our research focus on the class of objects represented by *polygonal meshes*, which are the most popular 3D object representations approximating the objects surface by a set of simple convex polygons.

Definition 1 ***(k-simplex)*** *Given a set A of point in the $\mathbb{R}^n$ space, the convex combination of k+1 affinely independent points of $V \subseteq A$ is called k-simplex, where $k < n$.*

When it is required to highlight the number of points of a k-simplex we use the notation φ_k rather than φ.

Definition 2 ***(s-facet of a k-simplex)*** *Let φ be a k-simplex defined by $V = \{v_0, v_1, ..., v_k\}$ and let ϕ be a simplex defined by $V' \subseteq V$, where $\mid V' \mid = s + 1$. The simplex ϕ is called s-facet of φ and this relation is denoted by either $\varphi \rhd \phi$ or $\phi \lhd \varphi$.*

In other words, an *s-facet* ϕ is as a convex combination of s+1 points of V (see figure 1.6).

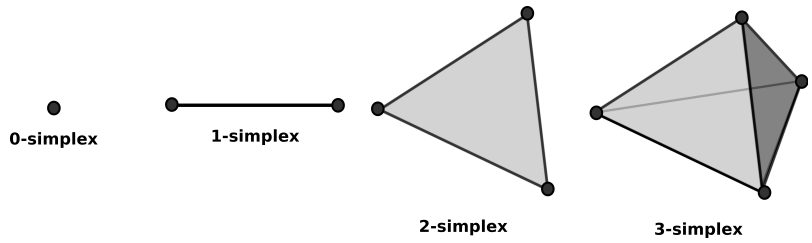

Figure 1.5: Example of k-simplices where each k-simplex is a (k-1)-facet of the simplex on the right.

Definition 3 ***(Simplicial complex)*** *A finite set H of simplices is a simplicial complex if the following conditions hold:*

1. *if $\varphi \in H$ and $\phi \lhd \varphi \Rightarrow \phi \in H$, that is, each s-facet of a simplex $\varphi \in H$ is also in H;*

2. *either $\varphi \cap \varphi' = \emptyset$ or $\varphi \cap \varphi' \lhd \varphi$ and $\varphi \cap \varphi' \lhd \varphi'$, that is, the intersection between $\varphi, \varphi' \in H$ is either empty or a common s-facet.*

Figure 1.6 shows some examples of simplicial complexes. The dimension of a simplicial complex H, $dim(H)$, is the maximal dimension of its elements. If $dim(H) = k$, then H is *k-complex*.

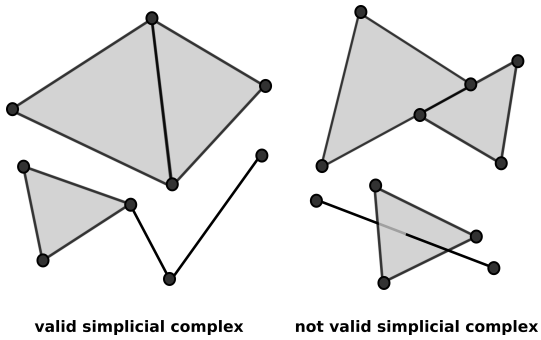

Figure 1.6: Valid simplicial complex (right) and not valid one (left)

Definition 4 ***(Simplicial k-complex)*** *A simplicial complex H is k-complex if $dim(H) = k$.*

Definition 5 ***(Pure simplicial complex)*** *A simplicial k-complex H is pure if $\forall \phi \in H \; \exists \; \varphi \in H \mid \phi \lhd \varphi$.*

Definition 6 ***(Orientation of a k-simplex)*** *An orientation of a k-simplex is an equivalence class of the permutations of its vertices obtained by an even number of transpositions.*

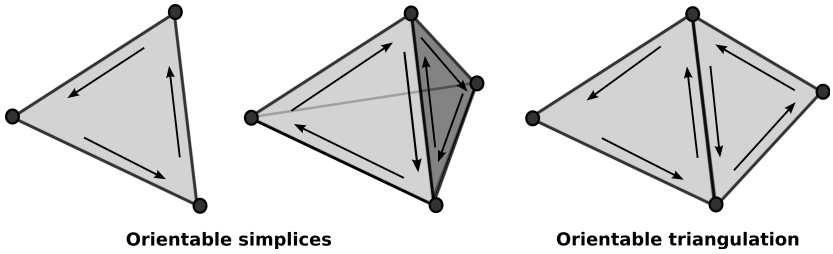

Figure 1.7: Orientations of k-simplices.

Definition 7 ***(Underlying space of a simplicial complex)*** *The underlying space of a simplicial complex H is $|H| = \bigcup_{\varphi \in H} \varphi$.*

Note that $|H|$ is a topological space and simplicial complexes are used to represent *manifolds*, which are topological spaces, too. Manifolds are used to define surfaces (*2-manifold*) and their properties, like differentiability, where it is typically requires that for each point within the manifold representation, there exists a neighbourhood homomorphic to the open disk.

Definition 8 ***(Triangulation)*** *Given a topological space X, a simplicial complex H is a triangulation if $|H|$ is homeomorphic to X.*

A triangulation is maximal if the addition of a new simplex violates the definition of simplicial complex.

Definition 9 ***(Orientable triangulation)*** *A triangulation T is orientable if two k-simplices $\varphi, \varphi' \in T$ define two different orientations on the shared (k-1)-facet.*

Definition 10 ***(Polyhedral mesh)*** *An orientable triangulation T is a polyhedral mesh if T is a pure orientable k-simplicial complex and if each (k-1)-simplex in T is shared by at most two k-simplices in T.*

Given a simplicial complex H let us denote $V = \bigcup_{\varphi_0 \in H} \varphi_0$, $E = \bigcup_{\varphi_1 \in H} \varphi_1$, and $F = \bigcup_{\varphi_2 \in H} \varphi_2$, that represent the sets of vertices, edges and facets of H, respectively. It is possible to define the following three relations:

- $vR_{ve}e \Leftrightarrow \exists v' | (v, v') = e$;
- $vR_{vf}f \Leftrightarrow \exists v', v'' | (v, v', v'') = f$;
- $eR_{ef}f \Leftrightarrow \exists v, v', v'' | (v, v') = e,\ and\ (v, v', v'') = f$;

where $v, v', v'' \in V$, $e \in E$, and $f \in F$.

Definition 11 ***(Triangular boundary mesh)*** *Given a polyhedral mesh defined by a set H of 2-simplices, a triangular boundary mesh is $M = \{V, E, F, R_{vf}, R_{ve}, R_{ef}\}$, $M = \{V, E, F\}$ for short.*

In the triangular boundary meshes the neighbourhood of a facet is the set of edges and vertices in relation to that facet, while the neighbourhood of a vertex is the set of edges and facets adjacent to the vertex. Such neighbourhoods are also called *cycles*. When a cycle does not contain any hole is said to be *single*. Note that two facets are in relation if they share an edge and the *connectivity* of a mesh is related to the adjacency relations of its elements. The connectivity is required to perform any kind of local analysis as it allows to reach the neighbourhood of each vertex.

Furthermore a mesh is *regular* or *structured* if each vertex has the same number of adjacent vertices, otherwise is non-regular or *unstructured*.

The suitability of a mesh depends on several geometrical features. High quality meshes are characterized by the following properties:

- the variance of the area enclosed by triangles should not be very large;
- the aspect ratio of triangles should be closed to 1, where the aspect ratio is the ratio between the diameter of the circumscribed circle and the maximal edge length of the triangle;
- in the case of unstructured meshes, the variance of the number of adjacent neighbours of the vertices should be as small as possible.

1.4 Mesh Generation

Many methodologies have been developed for generating a polyhedral mesh starting from a cloud of points. *Triangulation* is the most important and widely used approach for unstructured mesh generation; relevant techniques are: *Delaunay Triangulation (DT)* [31, 18, 46, 11], *Advancing Front Method (AFM)* [23] and *Graded Triangulation (GT)* [44].

Definition 12 ***(Delaunay Triangulation)*** *A triangulation T on a set V of points is a Delaunay Triangulation if each simplex of T is circumscribed by an hypersphere that does not contain any point in V.*

DT algorithms are classified in different groups depending on the approaches used:

- **Incremental Insertion.** This class of algorithms perform the DT by starting with a simplex containing the convex hull of the point set; then other vertices are inserted progressively. An example is the Watson's algorithm [46] for 2D triangulations, which starts with a super triangle that encompasses the whole domain.
- **Divide and Conquer.** These algorithms recursively carry out a partition and triangulation on the input points, then a merging phase is applied in order to join the resulting triangulations [31, 18, 11].

The *Advancing Front Method (AFM)* starts from the boundary of the cloud of points and adds new simplices progressively. The right location of new elements is crucial and the main issue is represented by the merging of the located advancing fronts. Note that in the three-dimensional space, this method produces tetrahedral meshes.

The *Graded Triangulation*, defined in the two-dimensional space, exploits both DT and AFM. The triangulation is improved by adapting number and size of the triangles to the shape of the starting boundary.

Different applications require different mesh quality characteristics, for instance in order to achieve very fast motion and rendering, computer gaming meshes are usually defined by a small number of elements (triangles or quadrilaterals); on the other hand, scientific applications may need to process large amount of data with high level of details for feature extraction and surface analysis. Acquisition systems produce very dense cloud of points resulting in meshes with a very large number of triangles [68]. In this scenario a very important property is the *mesh resolution*, which intuitively indicates the level of detail of the mesh surface and is it related to the number of vertices.

The simplification of the input data may improve the results of such algorithms and may reduce the execution time. *Progressive meshes* adapt the set of mesh points according to the required level of details, as in some visualization interfaces where the resolution of the virtual object is related to the zoom level allowing efficient rendering.

Several papers [36, 55, 57, 62, 21] have been written about *mesh simplification* algorithms to reduce the number of vertices by iteratively perform some operations on either vertices or edges. In order to preserve the shape of the object, the cost of each operation is usually computed as the distance between the original mesh an the simplified one. Mesh simplification algorithms are classified according to the type of operation used to reduce the number of vertices:

- **Vertex Removal.** In [57, 62] the mesh is simplified by iteratively selecting vertices for removal, then the neighbourhood of each removed vertex is re-triangulated. The cost is computed as the distance between the removed point and the fittest-plane defined on the neighbouring points;

- **Vertex Clustering.** These methods [36, 55] use a grid structure obtained from the object bounding-box and all the vertices contained in a grid cell are cluster together. The object shape is not guaranteed by these methods.

- **Edge Collapse.** Many algorithms reduce the number of vertices by collapsing the endpoints of the edges. The approach used in [19] allows the control of the object details, it computes both the upper and the lower bounds of the edges length by using two parameters p_1 and p_2. The parameter p_1 indicates the desired resolution, while p_2 specifies the deviation of the edges length from the given resolution p_1. The shape of the object is preserved by a *shape*

change measure defined as the maximum distance between the mesh before and after an operation is applied.

In Computer Vision it is often required a local object analysis for features extraction and segmentation, algorithms often require a topological representation of the object where neighbourhood operations can be easily performed. For these purposes a suitable mesh representation is useful and local analysis can be accomplished through the connectivity of the vertices. Notice that two adjacent facets could have edges with different lengths and this characteristic must be considered during the evaluation of the surface features, like its differential properties. The algorithm presented in [19] addresses also this problem and can be used to *normalize* the lengths of the edges.

1.5 Polygonal Mesh Data Structures

Basic information needed to analyse an object surface deal with the adjacency between mesh items, therefore the implementation of objects surfaces segmentation algorithms requires to access the mesh elements efficiently. Mesh data structures must describe 2-manifolds and store all the needed topological relations between elements by keeping track how an item is connected to its neighbours.

Typical mesh queries are:

- access the vertices of a facet;
- access the vertices of an edge;
- visit the edges of a facet according to some order;
- visit the edges adjacent to a vertex;
- visit the facets adjacent to a vertex;

3D data structures can be mainly distinguished into *edge-based* and *face-based*, where the topological information are related to either the edges or the facets neighbourhood, respectively.

Edge-based structures store, for each edge in the mesh, some pointers to its vertices and to its adjacent edges. On the contrary, face-based structures store, for each facet, some pointers to its adjacent facets and to its vertices. Each data structure has its own advantages in terms of memory requirements and simplicity for topological operations, their usage depends on the application needs. To the best of our knowledge, at present there are no standard face-based models, while there are two well known edge based approaches: *Winged-Edge* [2] and *Half-Edge*.

In this thesis the *Half-Edge* data structure has been adopted when the access the neighbourhood of the mesh elements in local surface analysis is needed.

The Half-Edge structure duplicates each edge into two *virtual half-edges* according to the *orientation* of the cycle of its facet (see Figure 1.8), moreover it maintains for each half-edge the pointers to:

- the opposite half edges;
- the adjacent vertex;
- the adjacent facet;
- the next half edge;
- the previous half edge.

Note that the previous half-edge can be referenced by using only the information about the next half-edge. Mesh queries can be easily implemented by using this data structure and with regular meshes they can be performed at constant time.

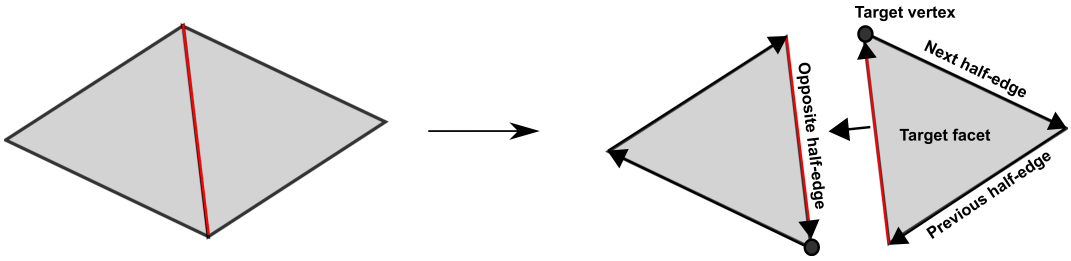

Figure 1.8: The red edge is duplicated into two half-edges each one oriented according to own adjacent facet.

Chapter 2

Mesh Surfaces

Several Computer Vision tasks require the estimation of the local geometric properties of the surfaces. Consider for example the problem of *objects segmentation*, that is, the problem of dividing the set of object elements into sub-sets having similar geometric features (*segments*). The recognition of a particular segment can be driven by using some a priori knowledge about the surface geometry. Furthermore *classification problems* need to decide if a segment belongs to some surface type, like spherical surfaces, cylindrical surfaces, etc. Such segments description is used to simplify the recognition of a given object in a 2D scene. In addition, 3D object retrieval exploit segments to match an object against a given model. In contrast, reconstruction problems need to estimate the fittest surface approximating an unorganized and sparse cloud of points, typically represented by range images.

These tasks require the analysis of the input data by means of descriptors invariant under different transformations, like rotations, translation, scaling, etc.

According to the differential geometry theory, the coordinate system used to represent surfaces does not influence their properties, which are completely described by first and second-order derivatives.

The problem of recovering surface properties depends on the input data at hand, polyhedral meshes and range images are usually processed with different approaches. In range images, the grid on which points are aligned gives a natural parametrization of the surface. In contrast, triangular meshes have no natural parametrization defined on them [63] and implicit parametrization is no suitable for approximating arbitrary data [45]. Here, mesh surfaces are typically described by second-order shapes as spheres, paraboloids, ellipsoids, cylinders, hyperboloids, etc. Furthermore, experimental results show that higher-order surfaces gain little advantages [30].

This chapter focuses on the process of estimating differential quantities on triangle meshes. Noise can heavily affect such estimation, thus many methods, as surface *smoothing* (or *fairing*) have been developed in the past decades to reduce noise due to high frequencies on the surfaces. We will also discuss the basic principles of mesh denoising.

Important surface features can be also captured through a *tensor voting* approach. As explained in section 2.4, the shape of an object can be inferred from a cloud of points. The saliency of each point can be described by tensors which encode some information and propagate it to its neighbourhood. Diffusion is implemented by a voting mechanism, where each point collects the votes (i.e. tensors) from its neighbouring elements. The shape of the object is then obtained by analysing the votes collected at each site. This technique allows to define very useful surface

descriptors that can be used for mesh segmentation purposes.

2.1 Differential Geometry Background

Differential geometry has been used in Computer Vision for the description of surfaces [50]. The local geometry properties are computed on some quadric surface approximating a set of mesh vertices. In this context the basic concept required for surface analysis is represented by the surface *fundamental forms*, which are extremely important and useful in order to determine the metric properties of surfaces.

Let us assume that S is a surface embedded in $\mathbb{R}^3$, represented by an arbitrary parametrization of two variables $\mathbf{X}(u, v)$ which is smooth in the neighbourhood of a point $p(x, y, z)$.

Each mesh vertex is characterized by a unit normal vector defined as the normalised cross product of the partial derivatives of $\mathbf{X}$:

$$\mathbf{n} = \frac{\mathbf{X}_u \times \mathbf{X}_v}{\parallel \mathbf{X}_u \times \mathbf{X}_v \parallel}$$

For small variations of the parameters (u, v), the *first fundamental form* measures the amount of movement on the surface. Such measure is rotation and translation invariant and does not depend on the surface embedding and on the parametrization. While, the *second fundamental form* depends on the embedding in the 3D space and it measures the changes of the surface normal for some movements of the parameters (u, v). Therefore, the first and second fundamental forms are considered as *implicit* and *explicit* properties of surfaces, respectively. Such forms are defined as follows [6]:

$$I(u, v, du, dv) = d\mathbf{X} \cdot d\mathbf{X} = d\mathbf{u}^T G d\mathbf{u}$$

$$II(u, v, du, dv) = -d\mathbf{X} \cdot d\mathbf{n} = d\mathbf{u}^T D d\mathbf{u}$$

where

$$d\mathbf{u} = (du, dv)^T$$

and

$$G = \begin{pmatrix} \mathbf{X}_u \cdot \mathbf{X}_u & \mathbf{X}_u \cdot \mathbf{X}_v \\ \mathbf{X}_u \cdot \mathbf{X}_v & \mathbf{X}_v \cdot \mathbf{X}_v \end{pmatrix}$$

$$D = \begin{pmatrix} \mathbf{n} \cdot \mathbf{X}_{uu} & \mathbf{n} \cdot \mathbf{X}_{uv} \\ \mathbf{n} \cdot \mathbf{X}_{uv} & \mathbf{n} \cdot \mathbf{X}_{vv} \end{pmatrix}$$

The geometric properties of surfaces are related to the Euclidean geometry of 3D space by the linear *shape operator* β, which generalises the curvature of plain curves. Such operator is a map $\beta : \Gamma(p) \rightarrow \Gamma(p)$, where $\Gamma(p)$ is the tangent (hyper)plane to the surface at the point p.

Given a vector $\mathbf{t}$ tangent to the surface S at p, the shape operator is defined as:

$$\beta(\mathbf{t}) = -\nabla_{\mathbf{t}}\mathbf{n}$$

where:

$$(\nabla_{\mathbf{t}}\mathbf{n})(p) = \lim_{\tau \to 0} \frac{\mathbf{n}(p+\tau\mathbf{t}) - \mathbf{n}(p)}{\tau}$$

In other words $\nabla_{\mathbf{t}}\mathbf{n}$ represents the directional derivative of $\mathbf{n}$ along the direction $\mathbf{t}$. The operator β can be expressed in vectorial form as:

$$\beta(\mathbf{t}) = G^{-1}D\mathbf{t}$$

With $\beta(\mathbf{t})$ at hand, the *principal curvatures* and *principal directions* of S on point p are obtained by computing the *normal curvature* of the surface at p in the direction of a vector $\mathbf{t}$. The normal curvature is defined as:

$$\kappa_n(\mathbf{t}) = \frac{\beta(\mathbf{t}) \cdot \mathbf{t}}{\|\mathbf{t}^2\|} \tag{2.1}$$

which measures the curvature of the plane curve obtained from the intersection of the plane defined by $\mathbf{t}$ and $\mathbf{n}$ with the surface.

The minimum and maximum values of κ_n are called the *principal curvatures* κ_1 and κ_2, respectively. These values are obtained according two directions, represented by the unit vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ denoted as *principal directions*. The principal curvatures are the eigenvalues of the shape operators, while the principal directions are the corresponding eigenvectors. The vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, together with $\mathbf{n}$ define an orthonormal frame at p, called *principal coordinate frame*.

According to the *Euler Formula*, the normal curvature can be defined, without loss of generality, by considering the angle θ between $\mathbf{e}_1$ and $\mathbf{t}$:

$$\kappa_n(\theta) = \kappa_1 \cos^2\theta + \kappa_2 \sin^2\theta \tag{2.2}$$

The *mean curvature* H and the *Gaussian curvature* K are important surfaces descriptors and they are derived from the principal curvatures:

$$H = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\theta) d\theta \tag{2.3}$$

and

$$K = \kappa_1 \kappa_2. \tag{2.4}$$

Note that H and K are the determinant and the half-trace of S, respectively, and they characterize a surface point p as *elliptic* (if $K > 0$), *hyperbolic* (if $K < 0$), *parabolic* (if $K = 0$ and $H \neq 0$), or *planar* (if $K = H = 0$).

A neighbourhood of a point $p(x, y, z)$ on a surface S can be approximated by a quadric surface. Let us represent such neighbourhood with $z = h(x, y)$, where the

coordinate frame is centred on p and the z axis is aligned with the normal $\mathbf{n}$ at p. The function h is differentiable and by Taylor's expansion of h at p up to order 2, the expression for the *principal quadric* Q of S can be derived:

$$h(x,y) = \frac{1}{2}(h^p_{xx}x^2 + 2h^p_{xy}xy + h^p_{yy}y^2) + R(x,y)$$

where h^p_{xx} is h_{xx} evaluated at p and $\lim_{(x,y)\to(0,0)} \frac{R(x,y)}{x^2+y^2} = 0$ and

The equation:

$$z = \frac{1}{2}(h^p_{xx}x^2 + 2h^p_{xy}xy + h^p_{yy}y^2) \tag{2.5}$$

approximates the surface S, and its *zero-set* of z defines the *principal quadric* Q of S at p.

The surface Q gives all of the important local differential properties of the surface S. The principal quadric at p can be expressed in the principal coordinate frame by a local parametrization $\mathbf{X}(x,y) = (x,y,h(x,y))^T$ with $\mathbf{n} = (0,0,1)^T$, resulting in the following matrices:

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$S = D = \begin{pmatrix} h^p_{xx} & h^p_{xy} \\ h^p_{xy} & h^p_{yy} \end{pmatrix}$$

The principal curvatures κ_1 and κ_2 are the eigenvalues of S, hence the principal quadric is:

$$z = \frac{1}{2}(\kappa_1 x^2 + \kappa_2 y^2) \tag{2.6}$$

2.2 Approximation of Local Surface Properties

Though several methods have been proposed in literature to estimate several differential properties on triangular meshes, there is no consensus on the most appropriate available techniques [50]. Furthermore, the choice of a particular method may depend on the types of data to be processed. Despite there are several points of view about the best estimation methods of the surface properties, it seems that the most suitable approaches are those using the discrete analogous of formulas in the continuous case. In this contribution, the surface properties are extracted by fittest quadric, hence in the following we will give major emphasis to these techniques.

2.2.1 Principal Quadric Estimation

Given a triangulation T (see Chapter 1), the estimation of the principal quadric at some point p on T involves the computation of the normal $\mathbf{n}$ at p. Such estimation depends on the mesh structure around each point, then different meshing might

produce different results. Many approaches have been adopted to estimate the mesh normals, and usually they are based mainly on the average of the normals belonging to the facets adjacent to p. Let us denote by $N(p)$ and $M(p)$ the sets of vertices and facets adjacent to p, respectively. The normal at p is estimated through the weighted average:

$$\mathbf{n} = \frac{\sum_{i=1}^{i=n} w_i \mathbf{n}_i}{\| \sum_{i=1}^{i=n} w_i \mathbf{n}_i \|} \tag{2.7}$$

where $n = |N(p)|$.

The weights w_i can be computed according to different approaches [43, 24, 39]. For instance it can be used the area of the barycentric cell at p (i.e. the cell obtained by joining the centre of mass of each facet with the middle point of its edges), or the angles of the facets adjacent to p ($w_i = \theta_i$), see Figure 2.1 *a* and *b*.

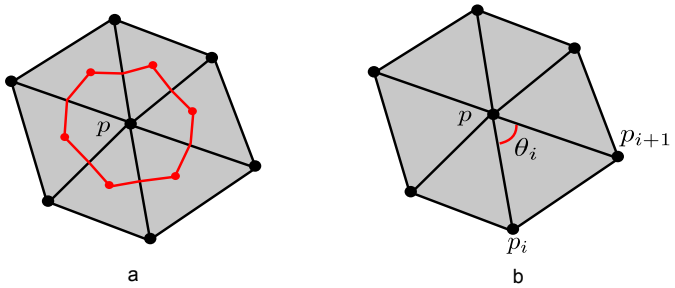

Figure 2.1: Local regions around a vertex [17]. a) Finite volume region using barycentric cells. b) External angles of a Voronoi region.

The vertices of a mesh are expressed in the *world* (global) coordinate frame. Once the normal is estimated, the quadric can be fitted by first aligning the neighbourhood $N(p)$ of a vertex p with the principal coordinate frame associated with p. The principal coordinate frame can be moved on this world by a translation and a rotation. The resulting coordinates $\mathbf{x} = (x, y, z)^T$ of a point in the principal coordinate frame centred in p are related to the its world coordinates $\mathbf{x}_w = (x_w, y_w, z_w)^T$ as following:

$$\mathbf{x} = \mathcal{R}(\mathbf{x}_w - p_w)$$

where $\mathcal{R}$ is the *attitude matrix* [41] and p_w are the global coordinates of p.

The principal quadric at some point p can be expressed in a coordinate frame $\mathbf{x}' = (x', y', z')$ centred in p, related to the principal coordinate frame by a rotation around its normal at p:

$$\mathbf{x} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x}'$$

This yields to the *rotated principal quadric*:

$$z' = a'x'^2 + b'x'y' + c'y'^2$$

The associated shape operator matrix is:

$$S = \begin{pmatrix} 2a' & b' \\ b' & 2c' \end{pmatrix}$$

and the differential properties of the surface at point p are computed as:

$$\kappa_1 = a' + c' + \sqrt{(a' - c')^2 + b'^2}$$

$$\kappa_2 = a' + c' - \sqrt{(a' - c')^2 + b'^2}$$

$$\alpha = \frac{1}{2}\arctan(b', a' - c')$$

$$K = 4a'c' - b'^2 \; H = a' + c'$$

In order to obtain the rotated principal quadric, the rotation from the world coordinates to the rotated principal frame must be defined. This is achieved by aligning x', with the projection of $\mathbf{x}_w$ onto the tangent plane defined by $\mathbf{n}$ [40]

$$\mathbf{x}' = \mathcal{R}'(\mathbf{x}_w - p_w)$$

where the matrix $\mathcal{R}'$ is defined as:

$$\mathcal{R} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)^T$$

with

$$\mathbf{r}_1 = \frac{(\mathbf{I} - \mathbf{n}\mathbf{n}^T)\mathbf{i}}{\| (\mathbf{I} - \mathbf{n}\mathbf{n}^T)\mathbf{i} \|}, \; \mathbf{r}_3 = \mathbf{n}, \; \mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \tag{2.8}$$

The vector $\mathbf{i}$ is the first axis in the global coordinate frame, while $\mathbf{I}$ is the identity matrix.

The rotated vertices are finally fitted and the coefficients of the rotated principal quadric are obtained by solving the following system of linear equations through a least-squares method:

$$\begin{pmatrix} x_1^2 & y_1^2 & x_1 y_1 \\ \vdots & \vdots & \vdots \\ x_n^2 & y_n^2 & x_n y_n \end{pmatrix} \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix} = \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix}$$

The quadric recovery is greatly influenced by the accuracy of the estimated normal vectors at mesh vertices. According to [40], the fitting can be improved by iteratively refine the normal estimation, and the authors proposed the following procedure:

1. estimate the rotated principal extended quadric;
2. estimate the the surface normal;
3. use the new normal to compute a new rotation matrix and rotate the data;
4. repeat the above steps until the incremental change in the direction of the normal falls below some tolerance level

At each iteration the estimate of the normal is computed by using the coefficients of the extended quadric $z' = a'x^2 + b'xy + c'y^2 + d'x + e'y$:

$$\mathbf{n} = \frac{(-d', e', 1)^T}{1 + d'^2 + e'^2}$$

Finally, the mean and Gaussian curvature are computed as follows:

$$K = \frac{4a'c' - b'^2}{(1 + d'^2 + e'^2)^2}$$

$$H = \frac{a' + c' + a'e'^2 + c'd'^2 + b'd'e'}{(1 + d'^2 + e'^2)^{3/2}}$$

According to the techniques presented, the problem of recovering quadrics on triangle meshes can be divided into the following sub-problems:

1. estimation of the normal of the surface at some point p;
2. computation of the rotation matrix $\mathcal{R}'$;
3. rotation of the data expressed in the world coordinate frame;
4. fitting of the rotated data with a quadric, alternatively with an extended quadric;
5. computation of the differential properties and the angle α relating the rotated principal quadric and the principal coordinate frame;
6. estimation of the attitude matrix.

2.2.2 Spatial Averages

The definition of differential quantities in the continuous case can be extended to triangular meshes by computing some *spatial average* around each vertex p of the mesh. The work presented in [17] shows that there exist strong analogies between the continuous case and the discrete case when the averaging is performed on special regions contained in the set $M(p)$ of the facets adjacent to p. Such regions are denoted as *finite volumes* and can be defined in different ways (see Figure 2.2).

The discrete form of the Gaussian curvature can be defined as:

$$K(p) = \frac{1}{A} \iint_A K dA \tag{2.9}$$

where A is some chosen area around vertex p. The finite volume associated with A is denoted as A_M. If the *Gauss-Bonnet theorem* is applied on A_M, the discrete Gaussian curvature at p can be computed as:

$$\frac{1}{A_M} \iint_{A_M} K dA = 2\pi - \sum_{p_i \in N(p)} \theta_i \tag{2.10}$$

where θ_i is the angle at p of the ith facet in $M(p)$ (see Figure 2.2 a). The Gauss-Bonnet is an important result of differential geometry, it connects the geometry of surfaces to their topology expressed by the Euler characteristic. In order to to accurately estimate the spatial average, a suitable finite volume must be defined. Voronoi cells provide tight error bounds [17] and assuming that $M(p)$ contains only non-obtuse triangles, the total area of the patch surrounding p is:

$$A_{Voronoi} = \frac{1}{8} \sum_{p_i \in N(p)} (\cot \alpha_i + \cot \beta_i) \|p_i - p\|^2 \tag{2.11}$$

As shown in Figure 2.2b, α_i and β_i represent the angles opposite to the edge pp_i. When the patch contains obtuse triangles, the Voronoi cells are constructed by taking into account the circumcenters of obtuse facets and the barycenters of non-obtuse ones. The resulting area is denoted as *mixed area* and the expression of the discrete Gaussian curvature becomes:

$$K(p) = \frac{1}{A_{mixed}} \left(2\pi - \sum_{p_i \in N(p)} \theta_i \right) \tag{2.12}$$

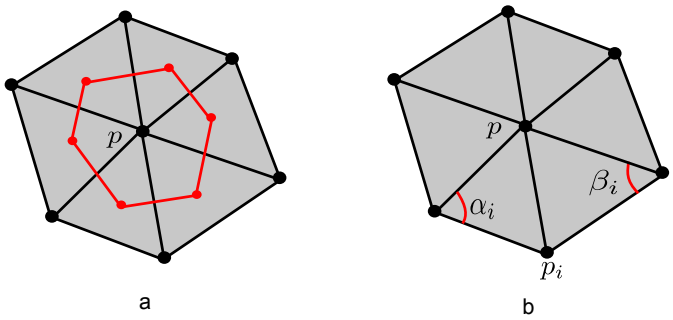

Figure 2.2: Local regions around a vertex [17]. b) Local region using Voronoi cells. d) Angles opposite to an edge.

The discrete mean curvature at some vertex p of a mesh can be derived by using the *Laplace-Beltrami* operator $\mathcal{K}$, defined as the divergence of the gradient of some function f. On smooth surfaces, $\mathcal{K}$ maps a point p to the vector $\mathcal{K} = 2H_p\mathbf{n}_p$. On

triangulations, the operator $\mathcal{K}$ over a finite volume A_M can be expressed as a line integral over the boundary of the volume:

$$\iint_{A_M} \mathcal{K} dA = \frac{1}{2} \sum_{p_i \in N(p)} (\cot \alpha_i + \cot \beta_i) \times (p_i - p) \tag{2.13}$$

Again, the mixed area is chosen and the Laplace-Beltrami operator is computed as:

$$\mathcal{K}(p) = \frac{1}{2A_{Mixed}} \sum_{p_i \in N(p)} (\cot \alpha_i + \cot \beta_i) \times (p_i - p) \tag{2.14}$$

Hence the mean curvature is $H_p = \dfrac{|\mathcal{K}(p)|}{2}$

2.2.3 Covariance Matrices

The computation of differentiable properties of surface may be not robust under additive noise, and surfaces may not present suitable *smoothness* to support differentiation. These problems led several authors to adapt the *covariance matrices* methods to triangulations [34, 3]. Given a point p of the mesh, the covariance matrix C_I is computed on the set $N(p)$ as follows:

$$C_I = \frac{1}{n} \sum_{i=1}^{i=n} (p_i - \overline{p})(p_i - \overline{p})^T \tag{2.15}$$

where $\overline{p} = \dfrac{1}{n} \sum\limits_{i=1}^{i=n} p_i$ represents the mean position vector.

The eigenvectors t_1 and t_2 of C_I define the tangent plane at p, so that the distances of the surface points in $N(p)$ to this plane are minimized. In addition, the eigenvector t_3 is an estimation of the surface normal $\mathbf{n}$ at p and thus C_I can be considered as the discrete equivalent of the first fundamental form matrix G.

According to [3] the discrete second fundamental form matrix can be defined by projecting the difference vectors $(p - p_i)$ onto the tangent plane determined by C_I. The contribute of each difference vector is weighted according to the orthogonal distance from p_i to the tangent plan:

$$C_{II} = \frac{1}{n} \sum_{i=1}^{i=n} (\mathbf{y}_i - \overline{\mathbf{y}})(\mathbf{y}_i - \overline{\mathbf{y}})^T \tag{2.16}$$

where

$$\mathbf{y}_i = [(p_i - p) \cdot \mathbf{n}] \begin{pmatrix} (p_i - p) \cdot \mathbf{t}_1 \\ (p_i - p) \cdot \mathbf{t}_2 \end{pmatrix}$$

The eigenvectors of C_{II} are an estimation of the principal direction at p.

Alternatively, since the principal directions lie on the tangent plane, the covariance matrix C'_{II} can be built by projecting the normal vectors in $N(p)$ onto the

tangent plane. Given an estimation of the normal $\mathbf{n}_i$ of the neighbouring vertices p_i obtained by C_I, the matrix C'_{II} is computed as C_{II}:

$$C'_{II} = \frac{1}{n}\sum_{i=1}^{i=n}(\mathbf{y}_i - \overline{\mathbf{y}})(\mathbf{y}_i - \overline{\mathbf{y}})^T \tag{2.17}$$

with $\mathbf{y}_i = \begin{pmatrix} \mathbf{n}_i \cdot \mathbf{t}_i \\ \mathbf{n}_i \cdot \mathbf{t}_2 \end{pmatrix}$.

2.3 Mesh De-noising Principles

Real world objects are typically characterized by smooth surfaces. Nevertheless, all the acquisition methods produce noisy and rough surfaces which need some smoothing process in order to exploit the differential property of the meshes.

Surface smoothness refers to the notion of continuous differentiability, and mesh de-noising (or fairing) is related to the *appearance* of the object surface and it is used to achieve more accuracy in the estimation of differential quantities (see Figure 2.3).

Figure 2.3: Example of mesh fairing.

The classic approach for mesh fairing uses a constrained energy minimisation on a functional $E(S)$ defined for a surface S:

$$E(S) = \iint_S (\kappa_1^2 + \kappa_2^2) dA \tag{2.18}$$

The non-linear dependence of the principal curvatures on S led to employ the *membrane* and *thin-plane* functionals denoted as $E_m(S)$ and $E_t(S)$, respectively.

$$E_m(S) = \iint_S (\mathbf{X}_u^2 + \mathbf{X}_v^2) du dv \tag{2.19}$$

$$E_t(S) = \int\int_S (\mathbf{X}_{uu}^2 + 2\mathbf{X}_{uv}^2 + \mathbf{X}_{vv}^2) du dv \tag{2.20}$$

Observing that the variational derivative corresponds to the Laplacian, fairing can be performed by integrating the diffusion equation over time:

$$\frac{\partial \mathbf{X}}{\partial t} = \lambda \mathcal{L}(\mathbf{X}) \tag{2.21}$$

where $\mathcal{L}(\mathbf{X}) = \mathbf{X}_{uu} + \mathbf{X}_{vv}$, $\mathcal{L}^2(\mathbf{X}) = \mathcal{L} \circ \mathcal{L}(\mathbf{X})$ and $\lambda > 0$
The diffusion flow reduces the noise by smoothing the high frequencies on the mesh surface. More details on the diffusion equation will be given in the Chapter 4.
At each point p the Laplacian can be approximated by the *umbrella operator* u:

$$\mathrm{u}(p) = \frac{\sum\limits_{p_i \in N(p)} w_i p_i}{\sum\limits_{p_i \in N(p)} w_i} - p \tag{2.22}$$

where the summation w_i are positive weights. In order to integrate the diffusion equation in the discrete case, an iterative process must be defined. The task is faced by generating a sequence of meshes by using the following update rule for a discrete time step $\Delta t = 1$:

$$p^{(j+1)} \leftarrow p^{(j)} + \lambda \mathrm{u}(p^{(j)}) \tag{2.23}$$

This procedure is known as *Laplacian smoothing*. At each iteration a vertex is moved by a displacement computed as the average position of the neighbouring vertices multiplied by some scale factor λ. Typical choices of the weights are $w_i = 1$, alternatively a function of the length of the edges pp_i are used.

Laplacian smoothing has several disadvantages: unnatural deformation on the mesh surface may appear if λ is not small enough, furthermore the result of smoothing depends on the sampling of the mesh vertices. The restriction on the scale factor requires hundred of iterations to smooth significantly large meshes. Moreover small details are lost due to the lack of local shape control.

Variation of the original Laplacian smoothing methods have been proposed [65, 66]. Although the deformations can be minimised by computing a weighted average of $\mathcal{L}$ and $\mathcal{L}^2$, results are still affected by scale problems. According to different authors, the umbrella operator is not adequate to approximate the Laplacian for triangular meshes.

A better approach is the *mean curvature flow* [17] which uses the Laplace-Beltrami operator to approximate the Laplacian. Here the vertices are moved along the surface normal with a speed equal to the mean curvature. Given a mesh point p, in the continuous case the displacement for a time step is:

$$\frac{\partial p}{\partial t} = -H(p)\mathbf{n}(p) \tag{2.24}$$

In the discrete setting, the local update rule is:

$$p^{(j+1)} \leftarrow p^{(j)} - H(p^{(j)})\mathbf{n}(p^{(j)}) \tag{2.25}$$

This formulation yields to *isotropic* smoothing, namely, the smoothing process has the same behaviour along all directions. The problem with this approach is the local geometry loss, surface features like boundaries, edges and ridges are smoothed in the same way as homogeneous regions. The detection of high curvature points through the principal directions can be used to perform anisotropic smoothing. This is required in order to reduce or suppress smoothing on such points and preserve small-scale features. The diffusion should be reduced or suppressed in the direction of such points. In [17] the following update rule is proposed:

$$p^{(j+1)} \leftarrow p^{(j)} - \sigma H(p^{(j)})\mathbf{n}(p^{(j)})$$

where the smoothing weight σ is defined as follows:

$$\sigma = \begin{cases} 1 & \text{if } |\kappa_1| \leq \tau \text{ and } |\kappa_2 \leq \tau \\ 0 & \text{if } |\kappa_1| > \tau \text{ and } |\kappa_2 > \tau \text{ and } K > 0 \\ \dfrac{\kappa_1}{H} & \text{if } |\kappa_1| = \min(|\kappa_1|, |\kappa_2|, |H|) \\ \dfrac{\kappa_2}{H} & \text{if } |\kappa_2| = \min(|\kappa_1|, |\kappa_2|, |H|) \\ 1 & \text{if } |H| = \min(|\kappa_1|, |\kappa_2|, |H|) \end{cases}$$

where τ is a user defined parameters. This approach is also dependent on the sampling of the mesh points and may yield to *over-smoothing* as time increases. Better results are obtained by combining the properties of Laplacian smoothing and mean curvature flow. The algorithm proposed in [47] moves the vertices both along the normal and along some direction on the tangent plane. This approach allows to smooth the surface while improving the sampling rate of the mesh vertices. The update rule thus becomes:

$$p^{(j+1)} \leftarrow p^{(j)} + \lambda(H(p^{(j)})\mathbf{n}(p^{(j)}) + C[\mathrm{u}_0(p^{(j)}) - (\mathrm{u}_0(p^{(j)} \cdot \mathbf{n}(p^{(j)})) \times \mathrm{u}_0(p^{(j)})])$$

where C is a positive constant or a function of the surface curvatures, and u_0 is the umbrella operator obtained with constant weight $w_i = 1$. See Figure 2.4 to see some results obtained though the methods presented.

The use of mesh denoising can improve the estimation of differential quantities. Since mesh processing algorithms often assume the knowledge of some descriptors as the principal curvatures, smoothing can be used as a pre-processing step although there is no suitable upper bound in the number of iterations. More considerations about this problem will be given in the last chapter.

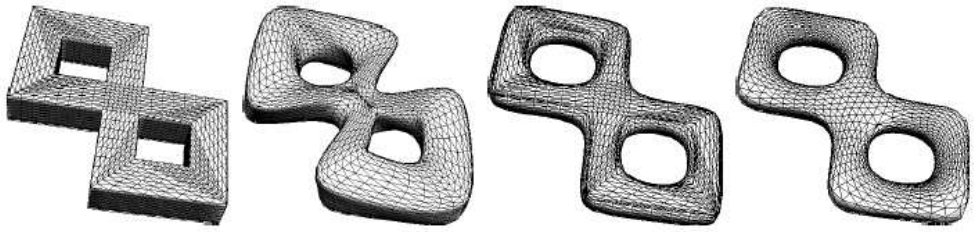

Figure 2.4: Smoothing of a mesh. From left to right: original mesh, smoothing obtained by the method presented in [65, 66], mean curvature flow, smoothing obtained by the method presented in [47].

2.4 Tensor Voting Based Surface Features Extraction

From the *Gestalt* theory we know that there exist several rules driving the recognition process of the objects, according to the spatial organization of the elements composing the scene. For instance, in Figure 2.5 *a*, it is possible to recognise two plain curves surrounded by some isolated points. *Proximity* and *good continuity* are just two examples of principles, used to aggregate the elements that compose *higher level* structures in the image, and Tensor Voting [42] allows us to simulate the human recognition process.

Given an object described by an unorganised set of points in both 2D and 3D space, its shape can be inferred by propagating the information encoded within each point through a voting process. Hence, voting produces new information about the underlying global structure of the object. For example, by referring again to Figure 2.5, the input image is just a collection of coordinates (x, y), after voting, an estimation of the tangent at each point is obtained. Through a given confidence measure, isolated point present *negligible* tangent information. The same reasoning can be applied for a three-dimensional images, where the normal at each point can be estimated through voting, and used to infer the whole shape of the object.

Note that, Tensor Voting theory also defines suitable surface descriptors useful to perform some objects segmentation, as described in Chapter 3.

The Tensor Voting approach presented in [42] is a set of procedures called the **salient feature inference engine**. Each point in the input image maintains its spatial information together with the estimates of its tangent and normal vectors. Note that the method requires at least the spatial position of the input points. Input elements will be denoted as *tokens*.

The whole algorithm can be summarised into three stages:

1. Each input token is encoded as a second order symmetric tensor. When the token maintains only the position information, the relative tensor is an isotropic ball of unitary radius.

2. *First voting step.* The tokens within in a neighbourhood communicate each other their information. During this stage they are transformed into generic

second order tensors encoding the confidence of the curve and surface orientation information.

3. *Second voting step.* A dense tensor map is computed by diffusing the information of each token to its neighbours. This tensor map encodes the saliency of each token, and it is used to infer the token *point-ness*, *curve-ness* and *surface-ness*.

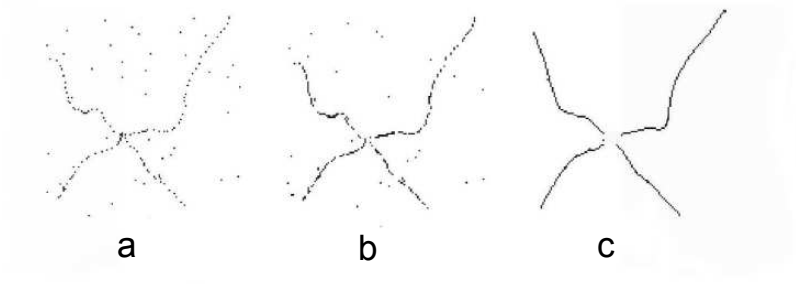

Figure 2.5: Shape reconstruction from a cloud of point in 2D. a) Original data. b), c) Voting steps.

The choice of tensors can be roughly argument as follows. A token may represent different types of entities: a point or a curve, or a surface, or these entities at the same time. In Figure 2.5 the intersection of the curves is both a point with no associated tangent, and two curves. Tensors allow to maintain all the possible information at the same time.

A second order symmetric K tensor in matrix form is written as:

$$K = \begin{pmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{pmatrix} \tag{2.26}$$

where $\lambda_1 \geq \lambda_2 \geq \lambda_3$ are the eigenvalues of K and $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$ are their relative orthonormal eigenvectors.

In order to make explicit the information encoded by each token, the tensor K can be decomposed into three components representing three different types of tensors, namely, the *ball tensor*, the *plate tensor* and the *stick tensor*. These tensors encode the *point-ness*, *curve-ness* and *surface-ness*, respectively. From the spectrum theorem [38], K can be decomposed as follows:

$$K = (\lambda_1 - \lambda_2)\mathbf{e}_1\mathbf{e}_1^T + (\lambda_2 - \lambda_3)(\mathbf{e}_1\mathbf{e}_1^T + \mathbf{e}_2\mathbf{e}_2^T) + \lambda_3(\mathbf{e}_1\mathbf{e}_1^T + \mathbf{e}_2\mathbf{e}_2^T + \mathbf{e}_3\mathbf{e}_3^T) \tag{2.27}$$

where $\mathbf{e}_1\mathbf{e}_1^T$ describes a stick, $\mathbf{e}_1\mathbf{e}_1^T + \mathbf{e}_2\mathbf{e}_2^T$ describes a plate, and $\mathbf{e}_1\mathbf{e}_1^T + \mathbf{e}_2\mathbf{e}_2^T + \mathbf{e}_3\mathbf{e}_3^T$ describes a ball.

The first voting step uses a *tensor voting field* for each type of tensor and produce a tensor map. The tensor K relative to some data point p accumulates votes by

summing the tensors contributions from neighbouring points. This is achieved by matrix summation.

After the first step the tensor K is decomposed into the corresponding eigensystem, a second voting step is then applied to estimate the orientation of the features. The ball tensor is not oriented and does not propagate any information. At the end of the whole algorithm, a tensor encodes likelihood (saliency) of a point belonging to a particular type of feature and the orientation of such feature. This means that:

- if a token has a relevant saliency value λ_3 related to its *point-ness*, there is no orientation information. This condition characterizes *junction points*;
- if a token has a relevant saliency value $(\lambda_2 - \lambda_3)$ related to its *curve-ness*, the estimate tangent is obtained by $\mathbf{t} = \mathbf{e}_3$. This condition characterizes points belonging to either smooth curves or surface junctions;
- if a token has a relevant saliency value $(\lambda_1 - \lambda_2)$ related to its *surface-ness*, the estimate normal is obtained by $\mathbf{n} = \mathbf{e}_1$. This condition characterizes points belonging to smooth surfaces.

Finally the global structure of the input object is recovered by inspecting the behaviour of tensors along a particular direction. For instance, a point belongs to the surface of the object if its saliency is locally extremal along the direction of such a normal. *Surface extremality* and *curve extremality* are the principal conditions used to infer the shape of the input points set. Extremal surface points can be triangulated to obtain a polygonal mesh (see Figure 2.6).

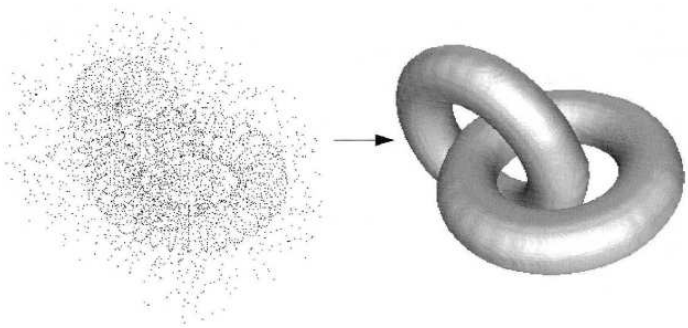

Figure 2.6: Inferring three-dimensional shape of an object from a cloud of point.

Features orientation is estimated by voting within tensor fields. According to human perception principles, these fields are designed to describe the orientation that a surface (or a curve) should have, when joining the centre of the field an another generic point influenced by such field.

We will see in Chapter 3 that tensor voting can be employed to perform *edge detection* on triangle meshes.

Chapter 3

Mesh Segmentation

3.1 The Mesh Segmentation Problem

Computer Vision problems usually refer to the automatic analysis and understanding of both 2D and 3D images. The segmentation task concerns with the partitioning of an object into a set of meaningful *segments* (i.e. non-empty and not overlapping regions) according to some criteria. Each segment must contain elements of the object having similar features and the set of segments have to cover the whole input data. The resulting segments are used to represent data by higher-level structures and can be used as input for other tasks.

When an object is represented by a 3D mesh, its segmentation produces a finite set of *sub-meshes* that are collections of elements of the mesh. Segmentation can be carried out starting from either the vertices, the edges or the facets of the mesh. More formally, given a mesh $M = \{V, E, F\}$, a sub-mesh $M' = \{V', E', F'\}$ of M is obtained by selecting one target subset S of either V, E or F and by gathering the other subsets so that their elements are in relation with the target one. For instance, when the target set is $S = V' \subseteq V$, then $E' \subseteq E$ and $F' \subseteq F$ are the subsets of elements adjacent to some vertices of V'.

Let $\mathcal{M} = \{M_0, ..., M_{t-1}\}$ the set of sub-meshes obtained by some segmentation of M. The elements of $\mathcal{M}$ must satisfy the following conditions:

1. $M = \bigcup_{i=0}^{t-1} M_i$;
2. $P(M_i \cup M_j) = 0$ for any pair of *adjacent* regions M_i, M_j, with $i \neq j$

where P is a predicate defined on each M_i and it indicates if some criterion function is satisfied by all elements within the same region.

Mesh segmentation can be also stated as an optimization problem [58]. In this scenario we needs to define a criterion function $J : \mathcal{P}(S) \rightarrow \mathbb{R}$, where $\mathcal{P}(S)$ is the *power set* of S, and the goal is the minimization of J under a set of constraints. Note that J induces a partitioning of S into t disjoint sub-sets, $S_0, ..., S_{t-1}$, by associating each S_i to a score.

The problem of mesh segmentation is strictly related to *constrained graph partitioning*. Indeed it is possible to define the *dual graph* [16] G of a mesh M by representing each element of the target set S as a node of G and exploiting the adjacent relation among the elements of S to link the nodes of the graph (see Figure 3.1). The segmentation of a mesh is equivalent to the partitioning of the dual graph by minimizing the number of *cut-edges* which is an NP-Complete problem [20, 7].

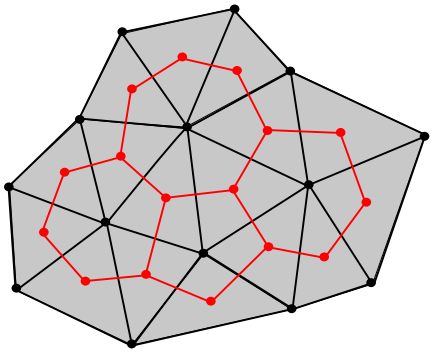

Figure 3.1: Dual graph of a mesh where $S = F$.

The computational complexity of mesh segmentation requires to address approximate solutions in feasible time. Different strategies have been proposed and the most important are: *region growing*, *hierarchical clustering* and *spectral analysis*, which will described more accurately in the next sections. Furthermore, mesh segmentation algorithms have several aims and can be mainly distinguished into two kinds according to the principal objectives: *surface-type* methods and *component-type* methods.

Different works use variants of the L_∞ and L_2 norms to measure the planarity of segments. Let $ax + by + cz + d = 0$ denote the fittest plane of the elements of a patch and suppose $S = V$, the above norms are defined as follows:

Definition 13 *(L_∞ **distance norm**) The maximum distance of a vertex $v = (v_x, v_y, v_z) \in V$ from a plane $ax + by + cz + d = 0$ is computed as $|(v_x, v_y, v_z, 1) \cdot (a, b, c, d)| \leq \varepsilon$*

Definition 14 *(L_2 **distance norm**) The average distance of vertices $\{v_1, ..., v_t | v_i \in V\}$ to a plane $ax+by+cz+d = 0$ is computed as $\frac{1}{t} \sum_{i=1}^{t} ((v_x, v_y, v_z, 1) \cdot (a, b, c, d))^2 \leq \varepsilon$*

3.1.1 Surface-base segmentation

Surface-based methods locate *patches*, i.e. surface regions whose elements satisfy some conditions (e.g. a constant curvature).

Many works [15, 5, 73, 22] refer to patch segmentation for mesh simplification and re-meshing problems. The basic idea is the replacement of a planar patches with one or more polygons.

Some authors [72] define a specific segmentation locating regions having small distortion after their parametrization onto the 2D space to solve the texture mapping problem. In computer graphic applications, textures can be considered as 2D images employed to give more realism to 3D objects. The texture mapping problem consists in the *mapping* of texture points onto the mesh previously *unfolded* on a plane. The unfolding process applied on complex surface yields to big distortions errors, thus the division into small patches usually improves the result.

Morphing is used to turn an object into another one through a fluid transformation of the surface and it is another field of computer graphic that takes advantage of patch segmentation. Indeed morphing algorithms can be enhanced by accomplishing transformations between surface patches [73].

Surface-based methods have been also used to improve the performances of those compression algorithms [71] relying on the Laplacian of a graph. Performances are largely improved when the Laplacian is evaluated on small patches.

3.1.2 Component-based segmentation

Understanding an object is often achieved by the recognition of its different *semantic components*. For example, a human body model can be divided into different part related to the head, harms, legs, etc. Component-type segmentation is used in several contexts and is usually related to the decomposition of an image into its *meaningful* sub-parts.

The disassembly of an object allows the matching of its sub-parts and improves the automatic recognition process [73, 4, 50]. For example, shapes comparison is required in database retrieval. Furthermore, several applications deal with the recognition of objects against a given model. The *3D jigsaw problem* concerns the reconstruction of an object starting from its parts. The set of objects to be "glued" can be located by first recognizing matching sub-regions [48].

Computer games often require to detect collision between complex models. The *bounding box* (*BB*) of the whole objects is inappropriate, thus more precise collision detection can be performed by considering BBs enclosing each single components [69].

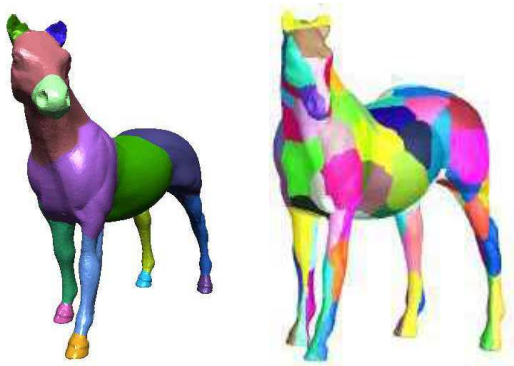

Figure 3.2: Example of a component-type segmentation [70] (left) and surface-type segmentation [56] (right).

3.2 Mesh Segmentation Methods

We have sketched different segmentation strategies have been mentioned in order to describe the main issues related with different applications. Although there exist a

variety of methods, segmentation rely principally on two factors. First, the criteria used to identify regions, namely, the rules adopted to assign an element to them. Criteria are usually defined by assuming some a priori knowledge about the objects, which is related to some descriptor as curvature, symmetry, angles between polygons, convexity and many other. Second, the constraints used to control the dimension and the shape of the regions during the segmentation.

Regions identification is typically performed by applying a *bottom-up* process: regions are generated by starting from one element and by successively adding new candidate elements. The insertion order is important, and different orderings yield to different results. To find sub-optimal solutions a common approach associates some cost to each insertion, thus the optimal ordering is typically achieved by employing *priority queues* of elements, where the priority of an element is in inverse proportion to its cost.

3.2.1 Region Growing

Region growing is a technique to locate sub-sets of elements in a input data set, satisfying some criteria. Let $\sigma \in S$ be an element of the target set and let $N_S(\sigma)$ be the set of elements $\sigma' \in S$ adjacent to σ. The growth of a region Φ starts by inserting the *seed* element σ and its expansion is then performed by adding those elements of $N_S(\sigma)$ satisfying some criterion function. Region growing continues by testing the neighbourhood of each new inserted element until no more insertions can be accomplished. The order used to check for a valid element to insert is usually managed by a priority queue on the boundary of Φ (see Figure 3.3). Once a region is located, another growing process begins from another seed not yet considered. The number of seed elements to initialize a region can be arbitrary. Note that the expansion of a region can be implemented by a *breadth first search* on the dual graph of S.

The region growing segmentation methods might depend on the choice of the seed elements, furthermore the regions are expanded separately during the execution of the algorithm. Thus region growing impose some limitations on the results from a global point of view.

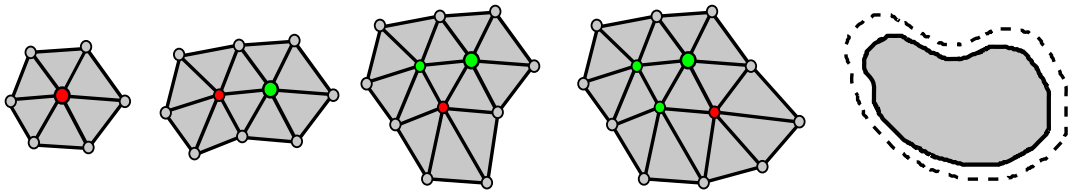

Figure 3.3: Example of region growing. The neighbourhood of the red vertices is analysed. Grey vertices are inserted into the priority queue for future processing. Green vertices have been processed and inserted into the region which are expanded from their boundary.

The texture mapping problem introduced previously can be solved through a

region growing approach. Texture mapping *atlases* are obtained by a two-phase algorithm [33]: first, the features contours are located, then regions are expanded inward from the boundaries by adding mesh elements. This approach simplifies the test used to associate an element with an existing region because its boundaries have been defined, already.

A general model used by many region growing segmentation algorithms is the *watershed transform*. Although there exist several watershed methods, only a few algorithms apply it on 3D meshes. Watershed segmentation is based on the definition of an *height map* $f : S \to \mathbb{R}$, obtained by different techniques, where S is the target set. The method can be described by using the analogy with the *flooding* process of adjacent *catchment-basins*. The segmentation algorithms proposed in this thesis defines a height map that we have used together with the watershed algorithm and whose detailed explanation is provided in section 3.3.

3.2.2 Greedy Algorithms for Clustering

Clustering algorithms are widely used in different contexts of data analysis, and segmentation can be considered as a particular clustering of the target set.

Clustering methods for mesh segmentation do not focus on any particular regions since segments are not located by independent processes. Clustering algorithms proceed toward a *greedy global* solution: regions are *assembled* by merging adjacent elements or already located segments, and the algorithm always chooses the best merging operation according to some cost function (see Figure 3.4).

Hierarchical clustering methods start by generating a cluster for each element of the target set, then clusters are progressively merged until no more operations can be done. The *hierarchical face clustering* method [22] performs a partitioning on the dual graph of $S = F$. The algorithm produces a sequence of segmentations such that for each step, the located regions are larger and contain more elements than the previous step. Merging of two clusters is performed by an *edge contract* operation on the dual graph. The merging cost is computed by using an L_2 based norm on the new generated cluster. This method has been used for different applications as progressive-meshes, surface simplification and collision detection.

Iterative clustering approaches assume that the number of output clusters is known a priori. *K-means* methods are examples of iterative clustering. A set of t representative elements of the target set is initially used to represent t different clusters. At each iteration the remaining elements are assigned to one of the t clusters according to the criterion function and the representatives are recomputed. A common strategy considers as representative, the center of mass of each cluster, hence the elements are assigned to the cluster if their distance from its representative is shorter than the distances from all the other representatives. Note that each region must be a connected component, while the non-planarity of the surfaces makes the Euclidean distance unsuitable. Most iterative clustering algorithms overcome this issues by performing a region growing step before recomputing the new representatives. In [60] the k-means method is used in face-based segmentation of two objects

for *morphing*.

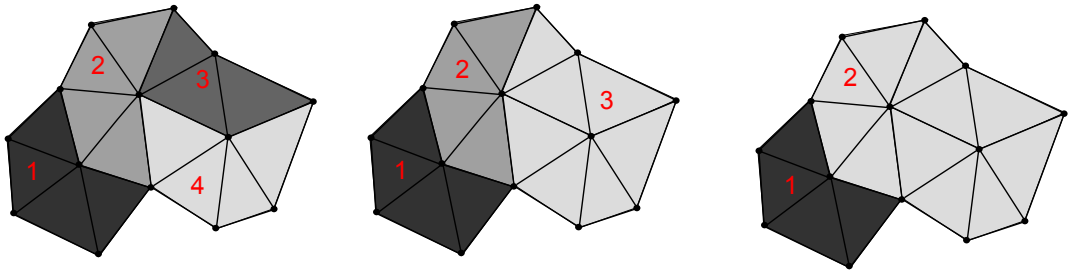

Figure 3.4: Example of clustering. Regions labelled with 1,2,3 and 4 are found through region growing and assembled by using an optimal merge operations ordering.

3.2.3 Spectral Analysis

The combinatorial graph partitioning problem can be reduced to geometric space partitioning problem by embedding a graph G into the space $\mathbb{R}^n$ by using the first eigenvectors of the Laplacian matrix of G [59].

The algorithm presented in [71] uses the Laplacian matrix to perform a compression of the mesh. In order to reduce the execution time, smaller sub-meshes are processed separately.

The method [53] uses a symmetric affinity matrix $W \in mat_{n\times n}$, where n is the number of facets in the mesh. The element w_{ij} encodes the probability that the facets i and j are clustered in the same region. Such matrix can be defined in two different ways, according to the required type of *segmentability* [54].

In the case of *structural segmentability* $W \in mat_{n\times n}(0,1)$ is the adjacency matrix, then

$$w_{ij} = \begin{cases} 1 & \text{if } \exists\, e_{ij} \in E \\ 0 & \text{otherwise.} \end{cases}$$

For *geometrical segmentability*, each element w_{ij} is computed by considering the minimal principal curvature $\overrightarrow{\kappa_i}$ and $\overrightarrow{\kappa_j}$ of the vertices i and j, respectively as following:

$$w_{ij} = \begin{cases} 0 & \text{if } e_{ij} \notin E \\ (|\overrightarrow{\kappa_i}| + |\overrightarrow{\kappa_j}|) \cdot \langle \overrightarrow{e}, \overrightarrow{\kappa} \rangle \cdot l & \text{if } \kappa_i < 0 \text{ or } \kappa_j < 0 \\ \varepsilon & \text{otherwise.} \end{cases}$$

Where $\overrightarrow{e_i}$ is the direction of the edge $e_{ij} \in E$ and l is the normalized length of e.

3.3 The Watershed Transform

Watershed [52] is one of the most important region-based approaches. Such method employs some height map f, defined on the image elements, and a graph representation of the image.

Let V be a sub-set of the lattice $\mathbb{Z}^2$, and $E \subseteq \mathbb{Z}^2 \times \mathbb{Z}^2$ be the set of edges defining the adjacency relations among the elements of V, we define the *graph* as the set $G = (V, E)$.

Definition 15 ***(Geodesic distance)*** *The geodesic distance $d_A(a, b)$ within A between two points $a, b \in A \subseteq \mathbb{Z}^n$ is the minimum path length among all paths from a to b within A. Moreover, the geodesic distance between the point $a \in A$ and a set $B \subseteq A$ is defined as $d_A(a, B) = \min_{b \in B} d_A(a, b)$.*

Definition 16 ***(Geodesic influence zone)*** *Given a set $A \subseteq \mathbb{Z}^n$ and a subset $B \subseteq A$ partitioned into t connected components $B_0, ..., B_{t-1}$, the geodesic influence zone $giz_A(B_i)$ of a set B_i within A is defined as:*

$$giz_A(B_i) = \{a \in A \mid d_A(a, B_i) < d_A(a, B_j)\ \forall j \in [0,\ \ldots,\ t-1] \text{ and } j \neq i\}$$

The union of all the influence zones of the sub-sets B_i is defined as:

$$GIZ_A(B) = \bigcup_{i=0}^{t-1} giz_A(B_i)$$

The set of points having the same geodesic distance from at least two nearest connected components induces a structure called skeleton by influence zones. Such structure is defined as the complement of the set $GIZ_A(B)$:

$$SGIZ_A(B) = A \setminus GIZ_A(B)$$

Let $V \subseteq \mathbb{Z}^2$ be a connected domain, then $C(V)$ denotes the space of real twice continuously differentiable functions on a V with only isolated critical points.

Definition 17 *Given a function $f : V \to \mathbb{R}$ belonging to $C(V)$ and $h \in \mathbb{R}^+ \cup \{0\}$, a h-level threshold set is:*

$$L_h = \{p \in V | f(p) \leq h\}.$$

Definition 18 ***(Topographical distance)*** *Given $f \in C(V)$ and $p, q \in V$, the topographical distance between p and q is:*

$$D_f(p, q) = \inf_{\nu \in V} \int_\nu \|\nabla f(\nu(s))\| ds,$$

where ν is a generic path (smooth curve) in V such that $\nu(0) = p$ and $\nu(1) = q$.

The watershed transform typically assigns a different *label* to each region. Such a process starts from the minima for f and it propagates until the segmentation is completed, then all the regions are labelled. In order to simplify the next definitions we denote by m_i a minimum of $f \in C(V)$ with label i, and with $\{m_i\}_{i \in I}$ the set of minima whose labels are in $I \subset \mathbb{N}$.

Definition 19 ***(Catchment basin)*** *Given a function $f : V \to \mathbb{R}$ belonging to $C(V)$ with minima $\{m_i\}_{i \in I}$ for some set of indices I, a catchment-basin Q relative to the minimum m_i of f is defined as:*

$$Q(m_i) = \{p \in V | \forall j \in I, i \neq j : f(m_i) + D_f(p, m_i) < f(m_j) + D_f(p, m_j)$$

Definition 20 ***(Watershed transform)*** *According to definition 19, the watershed of the function f is defined as the set of points not belonging to any catchment basin, formally:*

$$wshed(f) = V \cap (\bigcup Q(m_i))^c$$

The the watershed transform of f is a mapping $\gamma : V \to I \cup W$, where $W \notin I$ is a label:

$$\gamma(p) = \begin{cases} i & \text{if } \; p \in Q(m_i) \\ \\ W & \text{if } \; p \in wshed(f) \end{cases}$$

In the discrete case, the above definition of *watershed transform* is unsuitable if the function f exhibits plateaus, i.e. zones where f is constant. Plateaus are very common features on images and objects surfaces. In order to label plateaus properly, two different algorithmic definitions have been proposed: *watershed by immersion* and *watershed by topographical distance*. The former automatically takes care of plateaus, the latter needs a pre-processing on the image.

Let h_{min} and h_{max} be the minimum and maximum values for f, respectively, the problem of *plateaus* can be solved by defining a recursive process that increases the *level* of f from h_{min} to h_{max}. Let X_h denote the union of the catchment basins at level h, the set X_h can be expanded by considering a connected component obtained from the threshold set L_{h+1} at level $h+1$. Such expansion is performed by computing the influence zone of X_h within L_{h+1} resulting in an update X_{h+1}. Note that such connected component can be also a new regional minima.

The *watershed by immersion* is defined recursively as follows:

Definition 21 ***(Watershed by immersion)*** *Let MIN_h be the union of all regional minima at level h, then*

$$\begin{cases} X_{h_{min}} = L_{h_{min}} = \{p \in V | f(p) = h_{min}\} \\ \\ X_{h+1} = MIN_{h+1} \cup GIZ_{L_{h+1}}(X_h), \; h \in [h_{min}, h_{max}) \end{cases}$$

The watershed by immersion is the complement of the set $X_{h_{max}}$ in V:

$$wshed(f) = V \setminus X_{h_{max}}$$

According the definition 21, all non-basins elements contained in L_{h+1} but not in X_h are potential candidates to be assigned to a catchment basin in step $h+1$. A definitive labelling as watershed pixel can only happen after all levels have been processed.

The *watershed by topographical distance* assumes that the function f is plateaus-free, more precisely, each non minimum element has a neighbourhood having lower values for f. In 2D images this restriction is relaxed by introducing the *lower completion* $lc(f)$ of the function, which transform f into *lower complete* $f^* = lc(f)$.

The set of neighbours of a point $p \in V$ on $G = \{V, E\}$ is denoted by $N_G(p)$.

Definition 22 *Given a function $f : V \to \mathbb{R}$ belonging to $C(V)$, the maximal slope linking a point $p \in V$ to any of its neighbours is called the lower slope $ls(p)$, where*

$$ls(p) = \max_{q \in N_G(p) \cup p} \frac{f(p) - f(q)}{d(p,q)}$$

and $d(p,q)$ is the length of $e_{pq} \in E$.

By considering a cost $c(p,q)$ for each edge $e_{pq} \in E$, the topographical distance along a path $\pi_{pq} = \{p_0, .., p_k \mid p_0 = p \text{ and } p_k = q\}$ of points is computed as:

$$D_f^{\pi}(p,q) = \sum_{i=0}^{k-1} d(p_i, p_{i+1}) c(p_i, p_{i+1})$$

Definition 23 ***(topographical distance)*** *Let $B_{p,q}$ denote the set of all possible paths joining p and q, the topographical distance between p and q is defined as the minimum distance path in B_{pq}:*

$$D_f(p,q) = \min_{\pi \in B_{pq}} D_f^{\pi}(p,q)$$

Definition 24 ***(Path of steepest descend)*** *A path π_{pq} is called path of steepest descend if p_{i+1} belongs to the set of neighbours q of p_i such that $\frac{f(p_i)-f(q)}{d(p_i,q)} = ls(p_i)$.*

Catchment-basins need to consider the *lower completition* f^* of f. A valued graph is called lower complete when each node which is not in a minimum has a neighbouring node of lower value. By employing f^*, the *watershed by topographical distance* follows definition 20.

Both definitions of watershed yield to two different kinds of approaches typically known as *bottom-up* and *top-down* watershed methods, which may produce unsuitable results on noisy data, that over-segment the image into many small regions. This problem is usually solved by a successive merging process, where adjacent regions are merged together according to some metric indicating the saliency of a segment. Those can be mainly distinguished in *area-based* and *boundary-based* metrics. In the former case the saliency is evaluated by computing the area of the regions, while in the latter one the boundary of the regions is considered.

In [37] the watershed method is generalized to arbitrary meshes, the authors used the discrete curvature at each vertex as height map and a top-down approach. Each vertex v follows its steepest descend path until it reaches either a labelled minimum or a labelled vertex, in both cases their label is assigned to v. The saliency of a region S is computed as the difference between the vertex $v \in S$ of lowest curvature and the vertex on the boundary of S having lower curvature than all the other boundary vertices (*watershed depth*).
The whole algorithm can be summarized as follows:

1. compute the curvature;
2. locate and give an unique label to those vertices v such that $\forall v' \in N(v) : f(v) < f(v')$ (local minima);
3. Minimum plateaus (i.e surrounded by vertices having a greater value of f) are labelled;
4. descend each unlabelled plateaus to a labelled region;
5. descend all remaining unlabelled vertices;
6. merge regions whose watershed depth is smaller than a given threshold.

A region S_i is merged into the region S_j adjacent to the lowest curvature vertex within the boundary of S_i. This merging process is repeated until all regions have depth greater than the threshold.

The work presented in [8] uses a bottom-up approach and for each vertex v the height map is computed according to the concavity of the vertex. Those vertices v having Gaussian curvature $K(v) < 0$ are classified as boundary vertices and such vertices define the boundary regions (or *peaks* of the mesh). The height map f is then computed as $f(v) = 0$ when v is a boundary vertex, $f(v) = 1$ otherwise.

Region merging is based on two criteria. First the regions to be merged are located by considering the number of vertices within the region. Then the regions found with the first criterion are merged to their adjacent regions with longest boundary.
The whole algorithm can be summarized as:

1. each local minima is labelled, the remaining areas are considered as peaks;
2. peaks are eroded starting from the boundary between minima and peaks;
3. regions are merged.

3.4 Polygonal Mesh Edge-Detection

Two-dimensional images are characterized by sharp changes in brightness, while the surface of 3D objects may present *ridges* and other types of discontinuities [50].

The goal of *edge-detection* is the location of *feature-edges*, which provides the most important structural information about the object, thus reducing the overall amount of data.

In order to avoid any ambiguity we denote with *edges* and *feature-edges* the edges in E and discontinuities, respectively.

Likewise many other segmentation techniques, a *saliency functions* f must be defined on the target mesh elements. Edge-detection algorithms can be considered *component-based* because a region may be not discovered explicitly, rather it can be defined through a set of mesh elements bounded by some feature-edges. For this reason, edge-detection methods can be also considered as *implicit methods*.

Depending on the nature of the objects, *explicit* segmentation algorithms can produce regions such that their boundaries describe the feature-edges. For instance, by considering the boundaries of the broken surfaces on simple fragmented objects, feature-edges can be retrieved by region growing based on the polygon connectivity and the face normal distribution [48]. Therefore, region growing segmentation methods, based on the local concavity of the surface [8] (see section 3.3), are unsuitable for feature-edges detection as ridges can not be extracted.

3.4.1 Thresholding-based Edge-Detection

Filtering of f by classic and hysteresis thresholding, can be suitable for simple meshes, but usually it is unsuitable for very large and noisy meshes. In [25] the authors cope this problem by performing a threshold in a multi-resolution setting, and their algorithm returns a set of line-type features by the following three steps:

1. *classification step* - the saliency functions is used to assign some *weight* to the edges of the mesh;
2. *detection step* - the threshold produces a set of feature-edges which is successively decomposed into connected components (patches);
3. *erosion step* - the patches are reduced to lines by some *skeletonizing* method.

Several operators have been proposed for step 1 (see Figure 3.5), each one computing the weights on the neighbourhood of the edges. The simplest way to assign a weight to an edge is to compute the dihedral angle between the unitary normals $\mathbf{n}_i$ and $\mathbf{n}_j$ of the adjacent facets f_i and f_j. A *second order operator* (SOD) on a edge e is defined as:

$$w(e) = \arccos\left(\frac{\mathbf{n}_i}{\|\mathbf{n}_i\|} \cdot \frac{\mathbf{n}_j}{\|\mathbf{n}_j\|}\right) \tag{3.1}$$

This operator can be applied also on the average of the normal vectors of triangles adjacent to the vertices of f_i and f_j opposite to e. In this particular case it is named *Extended Second Order Operator* (ESOD).

Other operators are based on fitting polynomials. Let π denote the plane perpendicular to e and passing trough its middle point, it defines the set of points

P_e resulting by intersecting the mesh edges with π. P_e is used to compute the fittest polynomial function $p(e)$ belonging to π. The weight of e is then defined as $w(e) = p''(e)$.

Since P_e can be split into two subsets, each one lying on the semi-planes defined by e, a different method computes the two fittest polynomial functions $p_l(e)$ and $p_r(e)$, obtained from such sets, and the weight is computed as:

$$w(e) = \arccos\left(\frac{(1, p_l'(e))}{\|(1, p_l'(e))\|} \cdot \frac{(1, p_r'(e))}{\|(1, p_r'(e))\|}\right) \tag{3.2}$$

These operators are denoted as *Best Fit Polynomial* operator (BFP) and *Angle Between Best Fit Polynomials* operator (ABBFP), respectively (see Figure 3.5).

The algorithm stores the *progressive mesh* representation, i.e, a coarse mesh obtained through simplification, and a set of vertex split operations that allow to successively reconstruct the original object. The above steps are applied to the coarse mesh and during the reconstruction process, feature-edges are adapted to the surface changes.

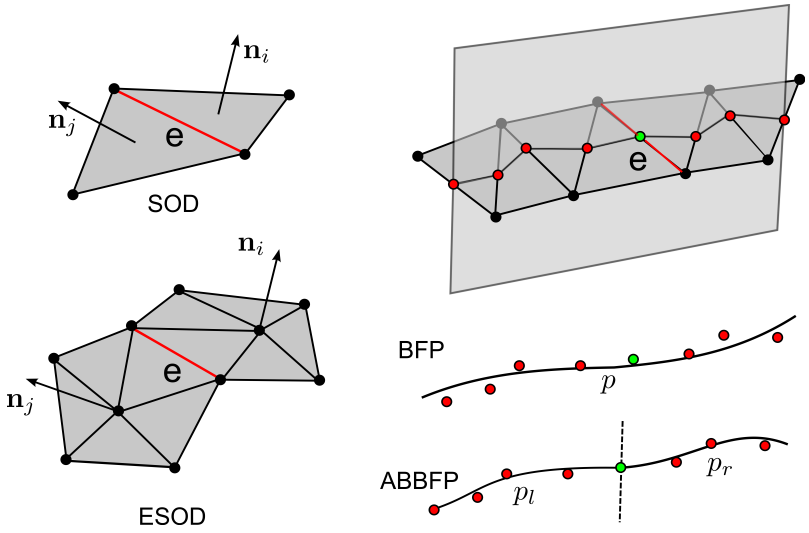

Figure 3.5: Weighting operators defined in [25].

3.4.2 Edge-Detection Based on Local Surface Analysis

The algorithm proposed in [12] computes the so called saliency of each arc on the surface, through a fuzzy membership defined on a continuous domain. This domain is automatically generated and the fuzzification process infers a natural segmentation of the surface, useful for locating contours and edge-type features to represent the objects. The key idea is to exploit the simple operators proposed in [25] to perform a local analysis over the mesh elements. The method computes a weight $w(v)$ for each vertex v and analyses the distribution of the weights with respect to some neighbourhood.

Let $M = \{V, E, F\}$ denote the input mesh and $\pi_{ee'}$ be a path from e to e' where $e, e' \in E$. Let us denote by $L_i(e) = \{e' \mid \min_{e' \in E} length(\pi_{ee'}) = i\}$, then $L_0(e) = e$ and the L_i are disjoint layers. For the sake of simplicity, each $L_i(e)$ is the i-th layer of the breadth first search tree with root e. The neighbourhood $N_r(e)$ of radius r is defined as follows:

$$N_r(e) = \bigcup_{i=0}^{i=r} L_i(e) \tag{3.3}$$

the area of $N_r(e)$ is denoted as $A_r(e)$.

Let $\nu_r(e)$ be the variance of the weights w relative to the elements included in $N_r(e)$ centred on e and with radius r. The *saliency* of an edge e is defined as:

$$s(e) = \sum_{i>0} \phi_i(e) \tag{3.4}$$

$$\text{with} \quad \phi_i(e) = \eta_i(e) \frac{\partial \nu_i(e)}{\partial r} |\nu_1(e) - \nu_i(e)|$$

where $\eta_i(e) = e^{A_{min}(e) - A_i(e)}$ is used to lower the resulting values ϕ_i, when moving far from e, with respect to the smallest window of radius $r = 1$, that is $A_{min} = \min_{e \in E}\{A_1(e)\}$.

Actually, negative values of s correspond to edges, positive values indicate ramps (i.e. surface elements near to feature-edges), while values close to zero denote smooth surfaces. Therefore, the saliency formula s assigns a score to each arc e, thus to discriminate among edges, ramps and smooth surfaces.

A similar approach is used in [13], where the classification rule is defined according to the slope of the regression straight line of the points (i, ν_i):

$$\text{slope}\big(N_r(a)\big) \triangleq \frac{\sum_{i=1}^{r} \left(i - \frac{r+1}{2}\right)(\nu_i - \overline{\nu})}{\sum_{i=1}^{r} \left(i - \frac{r+1}{2}\right)^2} \quad \text{with } r > 1 \tag{3.5}$$

where $\overline{\nu}$ is the mean value of the variances ν_i.

In the particular case of low resolution meshes, it can be useful to consider small values of r. When $r = 1$, it is imposed:

$$\text{slope}\big(N_1(e)\big) \triangleq \nu_1' - \nu_1,$$

where ν_1' is the variance of $N_1(e)$ and ν_1 is the variance of the first layer $L_1(e)$. That is, we check if e is relevant with respect to its smallest neighbourhood.

An arc e is classified as edge if both the following conditions hold:

1. $w(e) \times \big|L_1(e)\big| > \sum_j w(e_{1j})$;
2. $\text{slope}\big(N_r(e)\big) < \varepsilon$,

where ε is a threshold value, close to 0, used to take into account also very small variations of the weights w.

3.4.3 Normal Voting and Watershed Segmentation

Another method to assign a weight to the target set of vertices is the so called *normal voting* [64], which adopts the watershed segmentation on a height map defined by means of a normal voting scheme. Each vertex v collects votes from the normals of those facets having geodesic distance from v less than a given value. The vote $\mathbf{n}_i'$ given by the facet T_i to v depends both on the normal $\mathbf{n}_i$ of T_i in its centroid v' and a weight τ_i (see Figure 3.6), in particular the authors consider

$$\mathbf{n}_i' = 2(\mathbf{n}_i \cdot \tau_i)\tau_i - \mathbf{n}_i \tag{3.6}$$

where $\tau_i = \dfrac{(\overrightarrow{v'v} \wedge \mathbf{n}_i) \wedge \overrightarrow{v'v}}{\|(\overrightarrow{v'v} \wedge \mathbf{n}_i) \wedge \overrightarrow{v'v}\|}$

If M is the number of the facets included in the geodesic window, the vertex v collects a tensor T computed as:

$$T = \sum_{i=0}^{M} \mu_i \mathbf{n}_i' \mathbf{n}_i'^T \tag{3.7}$$

where μ_i decreases exponentially with the geodesic distance from v to v'.

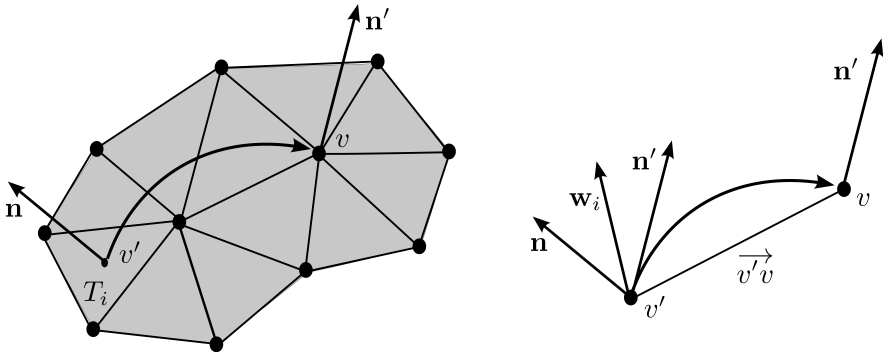

Figure 3.6: The normal voting scheme presented in [64].

In other words $\mathbf{n}_i'$ is obtained by transposing $\mathbf{n}_i$ along the arc connecting v and v'.

Let $\lambda_1 \geq \lambda_2 \geq \lambda_3$ be the eigenvalues of T corresponding to the eigenvectors γ_1, γ_2 and γ_3. The weight of v is defined as:

$$w(v) = \begin{cases} 1 & \text{if } |\overline{\mathbf{n}} \cdot \gamma_1| < \delta \\ 1 & \text{if } \lambda_3 > \alpha(\lambda_1 - \lambda_2) \textit{ and } \lambda_3 > \beta(\lambda_2 - \lambda_3) \\ (\lambda_2 - \lambda_3)/\lambda_2 & \text{otherwise} \end{cases} \tag{3.8}$$

where $\overline{\mathbf{n}} = \sum_{i=1}^{M} \mu_i \mathbf{n}_i$ and α, β and δ are pre-set values.

The conditions in the above definition are based on the tensor voting theory and, as already exposed in Chapter 1, tensor voting can be used to obtain surface descriptors. More precisely, the second condition gives maximum value to *corner* vertices, while the third one is used to approximate the weights in $[0, 1]$. Finally, the first condition ensures sharp features detection where tensor voting fails.

3.4.4 Active Contours

Snake curves [27] or *active contours* represent another relevant technique in the two-dimensional Computer Vision field to describe a previously located *contour area* of the image. The segmentation is driven by the energy minimization of a deformable model subjected to a set of forces that achieve the equilibrium state. This process is not carried out on the entire image, but on a region of it, located by either a manual or automatic procedure. Needless to say that the energy minimization process highly depends on the initial position of the snake curve, which is defined in parametric form, as:

$$\nu(s,t) = (x(s,t), y(s,t))$$

where t is the evolution time and $s \in [0, 1]$.

The basic mathematical model used to describe active contours is represented by the *splines curves*, whose evolution over time is ruled by both *internal* and *external* energies. Internal forces rely on the shape of the curve, while external forces are related to the underlying image.

The internal energy $E_{int}(\nu)$ is characterized by both an *elasticity* term and a *rigidity* term. These values are computed by the first-order derivative ν_s and the second-order derivative ν_{ss} of ν, respectively.

In order to control the evolution of the snake, two pre-set weights α and β must be provided, and the internal energy is defined as:

$$E_{int}(\nu) = \frac{\alpha(s)\|\nu_s\|^2 + \beta(s)\|\nu_{ss}\|^2}{2} \tag{3.9}$$

The external energy $E_{ext}(\nu)$ is chosen according to the image features as gradient, curvature, etc. Thus the total energy is described as the functional:

$$E(\nu) = \int_0^1 (E_{int}(\nu) + E_{ext}(\nu))ds \tag{3.10}$$

Note that an active contour can develop a corner only if $\beta(s) = 0$.

The minimum of this functional is detected by considering the zeros set of its first-order derivative. The resulting Euler equations in the continuous space are:

$$\begin{cases} \alpha\dfrac{\partial}{\partial s}x_s + \beta\dfrac{\partial^2}{\partial s^2}x_{ss} + \dfrac{\partial}{\partial x}E_{ext} = 0 \\ -\alpha\dfrac{\partial}{\partial s}y_s + \beta\dfrac{\partial^2}{\partial s^2}y_{ss} + \dfrac{\partial}{\partial y}E_{ext} = 0 \end{cases}$$

In the discrete case, snakes are represented by a set of n vertices (x_i, y_i), and the energy functional becomes:

$$E(\nu) = \sum_{i=1}^{n} E_{int}(i) + E_{ext}(i) \tag{3.11}$$

The derivatives are approximated by finite differences, and the Euler equations become:

$$\alpha(x_i - x_{i+1}) - \alpha(x_{i+1} - x_i) + \beta(x_{i+2} - 2x_{i+1} + x_i) - 2\beta(x_{i+1} - 2x_i + x_{i-1}) +$$
$$+\beta(x_i - 2x_{i-1} + x_{i-2}) + \frac{\partial}{\partial x} E_{ext}$$
$$\alpha(y_i - y_{i+1}) - \alpha(y_{i+1} - y_i) + \beta(y_{i+2} - 2y_{i+1} + y_i) - 2\beta(y_{i+1} - 2y_i + y_{i-1}) +$$
$$+\beta(y_i - 2y_{i-1} + y_{i-2}) + \frac{\partial}{\partial y} E_{ext}$$

We can also represent the energy minimization problem in matrix form:

$$\begin{cases} Ax + \dfrac{\partial}{\partial x} E_{ext} = 0 \\ \\ Ay + \dfrac{\partial}{\partial y} E_{ext} = 0 \end{cases}$$

Here A is a pentadiagonal banded matrix related to the discrete Euler equations above:

$$A = \begin{pmatrix} c & d & e & 0 & \dots & 0 & a & b \\ b & c & d & e & 0 & \dots & 0 & a \\ a & b & c & d & e & 0 & \dots & 0 \\ 0 & a & b & c & d & e & 0 & \dots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ e & 0 & \dots & 0 & a & b & c & d \\ d & e & 0 & \dots & 0 & a & b & c \end{pmatrix}$$

$$a = \beta,\ b = -4\beta - \alpha,\ c = 6\beta + 2\alpha,\ d = -4\beta - \alpha,\ e = \beta$$

The snake evolution can be implemented as an iterative process. If the snake deformation in t results in a decrease of its energy with respect to s, $\dfrac{d}{dt}\nu(s,t) = -\dfrac{dE(\nu(s,t))}{ds}$, then the energy variation $dE(\nu(s,t)) = -\dfrac{d}{dt}\nu(s,t)ds$ yields to the following equations:

$$\begin{cases} \gamma(x_t - x_{t-1}) = Ax_t + \dfrac{\partial}{\partial x} E_{ext}(x_{t-1}, y_{t-1}) \\ \\ \gamma(y_t - y_{t-1}) = Ay_t + \dfrac{\partial}{\partial y} E_{ext}(x_{t-1}, y_{t-1}) \end{cases}$$

x_t and y_t are the coordinates of a point at time t, and the parameter γ controls the convergence rate of the algorithm. Since the snake at time t depends on its energy at time $t-1$ the position (or deformation) update is given by:

$$\begin{cases} x_t = (A+\gamma I)^{-1}(\gamma x_{t-1} + \dfrac{\partial}{\partial x}E_{ext}(x_{t-1}, y_{t-1})) \\ \\ y_t = (A+\gamma I)^{-1}(\gamma y_{t-1} + \dfrac{\partial}{\partial y}E_{ext}(x_{t-1}, y_{t-1})) \end{cases} \tag{3.12}$$

To avoid the snake stagnates into energy local minima, the evolution process needs to be supervised to place the snake by moving some of its vertices.

The snake definition above can not be applied directly on the surface of a mesh because it does not include the constraints required so that the snake lies on the surface. Three types of approaches have been proposed to evolve active contours on triangle meshes: *level-set methods*, *2D snake projection*, *3D snake evolution*.

Level-set methods represent snakes as the zero-level set of some level-set function, and partial differential equation are used to control the curve over time. In [10] the motion of the snake is managed implicitly by the scalar level-set function. This approach may give poor results on open curves, furthermore it is unsuitable for interactive snake repositioning.

2D snake projection is achieved by extracting the surface patch that encloses it (see Figure 3.7 *top*), and such patch is parametrized on a 2D plane and the snake is evolved as above. Fast parametrization requires the surface to be *holes-free*, and patches should be small enough to avoid distortion errors. In [32] the authors proposed a general framework where the snake is partitioned and each sub-snake is evolved independently. Discontinuities arising at border points are then processed to smooth the whole active contour. Let us consider that $\nu(s)$ is partitioned in $\nu_l(s)$ and $\nu_r(s)$, and the sub-snake $\nu_c(s)$ partially overlaps both $\nu_l(s)$ and $\nu_r(s)$ (see Figure 3.7 *bottom*). The algorithm first evolves $\nu_l(s)$ and $\nu_r(s)$ independently, then it refines the result by moving $\nu_c(s)$.

Active contours can be directly applied on the mesh surface by a dynamic programming approach [1]. The energy of a snake $\nu = \{v_0, ..., v_{k-1}\}$ can be approximated by the sum of the energy of its vertices:

$$E(\nu) = \sum_{i=0}^{k-1} E(v_i) \tag{3.13}$$

The minimization process of $E(\nu)$ assigns a new position to each snake vertex v_i in order to minimize the overall energy. In [26], to update a snake vertex $v_i = (x_i, y_i, z_i)$, the algorithm does not check for all possible neighbours, but it considers only those vertices $PDir(v_i)$ lying along the principal directions of v_i. Then the external energy of a snake vertex is computed as:

$$E_{ext}(v_i) = \begin{cases} -\kappa_1(v_i) & \text{if } \kappa_1(v_i) > \kappa_1(p),\ \forall p \in PDir(v_i) \\ \\ C \gg 0 & \text{otherwise.} \end{cases}$$

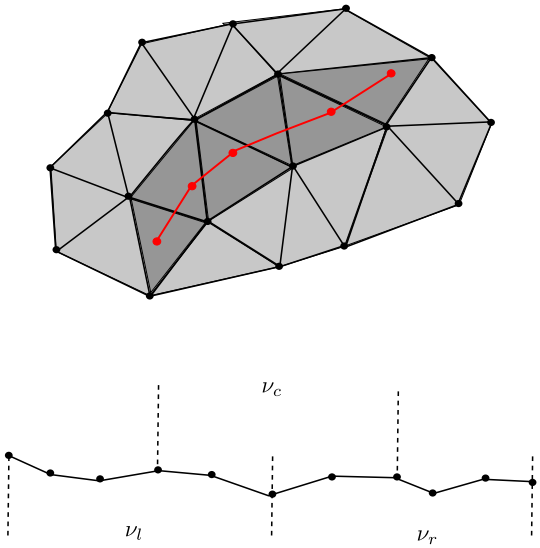

Figure 3.7: A patch enclosing the snake (top). Sub-snakes processed after parametrization (bottom)

The internal energy is:

$$E_{int}(v_i) = \alpha \parallel v_i - v_{i-1} \parallel + \beta \parallel v_{i+1} - 2v_i + v_{i-1} \parallel$$

By using this approach the topology of the snake must be controlled to prevent its vertices to be disconnected. Furthermore, once the snake reaches its final position, results can be jugged as the snake vertices are constrained on the mesh vertices and they can not be moved on the surface of a facet. Then results depend both on the resolution of the mesh and on the sampling of its points. Thus segmentation can be improved only by refining the mesh area surrounding the active contours.

Chapter 4

Diffusion-Based Mesh Edge Detection

There exist many physical problems strictly related to the evolution of curves and surfaces. Material interface propagation, fluid motion and crystal growth are some examples of problems whose solution is modelled by non-linear partial differential equations. *Geometric evolution* problems such as surface fairing, lead to similar variational approaches which allow the mathematical modelling on a continuous space, without considering any domain discretization. Mean curvature flow belongs to the class of *scale-space* approaches, it is related with the heat diffusion process, and it is one of the most important representatives in the context of surface fairing. This multi-scale approach allows to derive, from the input image, a family of images each one characterized by a different level of detail. This formulation of the fairing problem has been proposed to perform both surface elements classification and surface fairing.

4.1 Diffusion-Based Image Processing

The linear heat equation has been widely used to describe the spatial variations of some function f over time. Let $\Omega \subset \mathbb{R}^2$ and f_0 denote the domain of f and its initial value at time $t = 0$, respectively. The heat equation is:

$$\partial_t f - \triangle f = 0 \tag{4.1}$$

where $\triangle f = f_{xx} + f_{yy}$ is the Laplacian of f.
By evolving the system, a sequence of images $\{f(t)\}_{t\in\mathbb{R}^+}$ is obtained. When $\Omega = \mathbb{R}^2$ the solution coincides with the filtering of the original data with a Gaussian filter $G_\sigma^\infty(x) = (2\pi\sigma^2)^{-1}e^{-x^2}/2\sigma^2$ with standard deviation σ:

$$f(\sigma^2/2) = G_\sigma^\infty(x) * f_0$$

In section 2.3 we presented the *Laplacian smoothing* and the *mean curvature flow*. As explained, the major drawbacks of these approaches are object shape deformation and loss of small-scale features. The same problems arise in two-dimensional image processing. Gaussian filtering does not preserve the boundaries of the image, furthermore the regions boundary can not be located easily when processing at coarse scales because such boundaries shift from their original positions during the smoothing process. In [49], anisotropic diffusion has been used in order to reduce the

blurring on image edges, and the diffusion coefficient $G(\cdot)$ is modified according to the gradient of the edges on a particular location. The anisotropic diffusion process is:

$$\partial_t f - div\left(G\left(\frac{\|\nabla f\|}{\lambda}\right)\nabla f\right) = 0 \tag{4.2}$$

where *div* denotes the *divergence* operator and $\lambda \in \mathbb{R}^+$. This results in the suppression of smoothing on areas with high gradient, and it allows to perform both fairing and edge detection on noisy images. The position of edges is preserved during the smoothing process, and this simplifies the comparison among images at different scales.

4.2 Generalized Mean Curvature Flow

Given a surface S, the *Laplace-Beltrami* operator $\mathcal{K}_S$ presented in section 2.3 generalizes the Euclidean Laplacian operator. The geometric diffusion of the coordinates of a points p of the family of surfaces $S(t)$ (by varying time t) is written as:

$$\partial_t p = \mathcal{K}_{S(t)} p$$

The mean curvature vector $H(p)\mathbf{n}(p)$ is opposite to the Laplace-Beltrami operator, where $\mathbf{n}$ is the normal at p:

$$H(p)\mathbf{n}(p) = -\mathcal{K}_S p$$

then mean curvature motion can be written as:

$$\partial_t p = -H(p)\mathbf{n}(p)$$

Let $\mathcal{M}$ be an orientable manifold of dimension d and $p : \mathcal{M} \to \mathbb{R}^{d+1}$ be an immersion with normal $n : \mathcal{M} \to \beta^d$, where β is the shape operator. The generalized curvature motion [14] can be defined by considering general endomorphisms of the tangent space $\mathcal{T}$

$$a : \mathcal{T}\mathcal{M} \to \mathcal{T}\mathcal{M}$$

and the corresponding generalized mean curvature flow:

$$\partial_t p = H_a n, \;\; H_a = trace(a \circ \beta)$$

From variational calculus the geometric diffusion problem can be formulated as:

$$(\partial_t p, \vartheta) = -\int_{\mathcal{M}} H_a (n \cdot \vartheta) dA \tag{4.3}$$

for all $\vartheta \in C_0^1(\mathcal{M}, \mathbb{R}^{d+1})$.
Such a problem can be re-formulated by the relation $\mathcal{K}_S p = -H(p)n(p)$ as:

$$(\partial_t p, \vartheta) = \int_{\mathcal{M}} \mathcal{K}_S p \cdot \vartheta dA \tag{4.4}$$

and the generalized mean curvature is obtained by the relation above. The operator $\triangle_a(\cdot) = div_{(M)}(a\nabla_{(M)}\cdot)$, applied to p, leads to tangential components given by the divergence of the endomorphism a.

Theorem 1 *Let $\mathcal{M}$ be an orientable manifold of dimension d and $p : \mathcal{M} \to \mathbb{R}^{d+1}$ be an immersion. If $a : \mathcal{TM} \to \mathcal{TM}$ is differentiable, linear and symmetric to each tangent space $\mathcal{TM}$, then there exists a second-order differential operator Θ_a such that*

$$\Theta_a p = -H_a n$$

where $H_a = trace(a \circ \beta)$.
Moreover

$$\Theta_a(\cdot) = \triangle_a(\cdot) - (div_{\mathcal{M}} a)(\cdot)$$

This theorem allows to express $-H_a n$ by projecting $div_{\mathcal{M}}(a\nabla_{\mathcal{M}} p)$ onto the space spanned by the normal n. The equation $\partial_t p = H_a n$ can be written as:

$$\partial_t p = (v \cdot n)n$$

where $v = div_{\mathcal{M}}(a\nabla_{\mathcal{M}} p)$.

The classification of the surface elements as *edges*, *corners* and *smooth surfaces* employs a tensor which depends on the shape operator $\beta_{\mathcal{T}_p\mathcal{M}_\sigma}$. The surface $\mathcal{M}_\sigma$ is obtained by filtering $\mathcal{M}$ with a Gaussian-type filter, implemented by a short-time step $\tau = \sigma^2/2$ of mean curvature motion, where σ denotes the width of the filter.

In [14] the authors defined this tensor as a symmetric, positive, linear mapping on the tangent space $\mathcal{T}_p\mathcal{M}_\sigma$:

$$a^\sigma_{\mathcal{T}_p\mathcal{M}_\sigma} : \mathcal{T}_p\mathcal{M}_\sigma \to \mathcal{T}_p\mathcal{M}_\sigma$$

The symmetric shape operator can be represented by means of the orthonormal basis $\{\kappa_1^\sigma, \kappa_2^\sigma\}$ of $\mathcal{T}_p\mathcal{M}_\sigma$:

$$\beta_{\mathcal{T}_x\mathcal{M}_\sigma} = \begin{pmatrix} \kappa_1^\sigma & 0 \\ 0 & \kappa_2^\sigma \end{pmatrix}$$

and it is then used to define the above tensor as:

$$a^\sigma_{\mathcal{T}_p\mathcal{M}_\sigma} = \begin{pmatrix} G\left(\frac{\kappa_1^\sigma}{\lambda}\right) & 0 \\ 0 & G\left(\frac{\kappa_2^\sigma}{\lambda}\right) \end{pmatrix} \tag{4.5}$$

where λ is a user-defined threshold. This tensor classifies surface elements as follows:

- if $a^{\sigma}_{\mathcal{T}_p\mathcal{M}_\sigma} \sim diag[1,1]$, then p belongs to a smooth surface;
- if $a^{\sigma}_{\mathcal{T}_p\mathcal{M}_\sigma} \sim diag[1,0]$, then p belongs to a feature-edge, having direction κ_2^σ assumed $\kappa_1^\sigma \gg \kappa_2^\sigma$;
- if $a^{\sigma}_{\mathcal{T}_p\mathcal{M}_\sigma} \sim diag[0,0]$, then p belongs to a corner.

Figure 4.1 shows an example of mesh processed with the generalized mean curvature motion. The mesh is smoothed and the surface elements are classified.

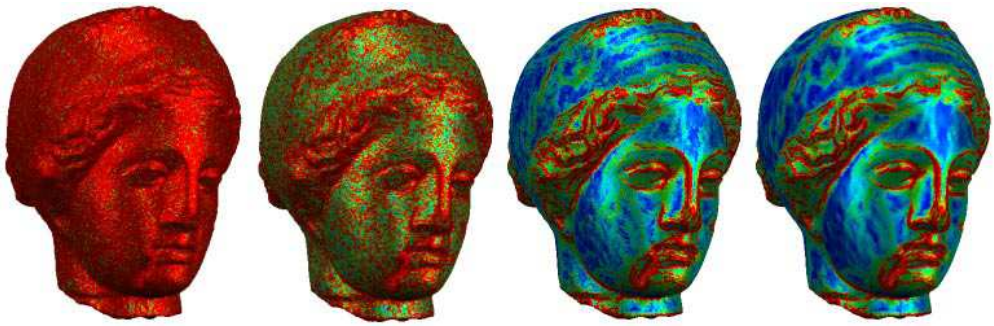

Figure 4.1: Classification performed at four, interleaved time steps during the generalized mean curvature motion [14] (*from left to right*).

4.3 Surface Curvature Diffusion

In several applications it is often required to automatize the feature extraction process. For instance, shape matching and database retrieval algorithms should be able to segment the object without any user intervention. Nevertheless, some user-given thresholds, weights or tolerance levels, seem to be necessary for all different types of segmentation strategies so far developed. Although the methods presented in the previous sections are effective and produce good results, the tuning of the required parameters is a drawback for a fully automatic object analysis.

We propose a parameter-free and fast heuristic method for mesh edge-detection, called *Surface Curvature Diffusion* (SCD). We exploit here the heat diffusion on the surface of an object, that is a problem mathematically solved in the continuous case by partial differential equations.

In the continuous case the heat diffusion is described by the equation:

$$\frac{\partial f}{\partial t} = \nabla^2 f \tag{4.6}$$

where f is the so called *heat function* defined on all the surface points.

On discrete lattices it is possible to solve the heat equation by an iterative process that updates the *temperature* of the mesh vertices over time. Let $M = \{V, E, F\}$ be a triangular mesh and $f : S \to \mathbb{R}$ denote a real function on each elements of the target set $S = V$, then the total variation of f can be discretized by:

$$f(t_{i+1}) = f(t_i) + \triangle(f_{t \to t+1})$$

Indeed, the PDE above is equivalent to $\partial f = \nabla^2 f \partial t$, and for each time step we compute the value of f at time step t_{i+1}, $f(t_{i+1})$, by summing the value of f at time step t_i, $f(t_i)$, with the total variation of f from time step t to $t+1$, $\triangle(f_{t \to t+1})$. In order to discretize the diffusion equation we need to compute the matrix $\mathcal{L}' = \nabla^2$ and to apply the update rule for some time step τ:

$$f(t_{i+1}) = f(t_i) + \mathcal{L}' f(t_i)\tau$$

To compute $\mathcal{L}'$, SCD needs to define a weight matrix W associated to the mesh M, as follows:

$$W_{ij} = \begin{cases} 1 & if \;\; \exists v_i, v_j \in V \mid (v_i, v_j) \in E \\ \\ 0 & otherwise \end{cases}$$

Let $d(v_i)$, with $i = 1 \ldots |V|$, denote the number of edges in E adjacent to a vertex v_i. In order to derive the non normalized symmetric Laplacian matrix $\mathcal{L}$, we use the diagonal matrices D and D', defined as follows:

$$D_{ij} = \begin{cases} d(v_i) & if \; i = j \\ \\ 0 & otherwise \end{cases}$$

and

$$D'_{ij} = \begin{cases} \dfrac{1}{d(v_i)} & if \; i = j \\ \\ 0 & otherwise \end{cases}$$

Then $\mathcal{L}$ is computed as:

$$\mathcal{L} = D - W$$

Now we use $\mathcal{L}$ to define the normalized non-symmetric Laplacian $\mathcal{L}'$:

$$\mathcal{L}' = D' - \mathcal{L} \tag{4.7}$$

SCD updates the values of f for the vertices v over time, by performing the following operation for each iteration:

$$f(v) = \mathcal{L}' f(v)\tau \tag{4.8}$$

By miming the heat diffusion for the adiabatic processes, where the total heat is conserved and it is redistributed on the surface object, SCD conserves the total value of some energy function φ and it redistributes φ on the vertices over time.

The function used by the proposed algorithm is the absolute value of the mean curvature H_v of the vertex v, whose value is known at time $t = 0$:

$$\varphi_v(t = 0) = |H_v| \tag{4.9}$$

where H_v is computed by using the *extended quadric methods* presented in Chapter 2. Figure 4.2 shows a mesh, the mean curvature H on it and the frequency of H. SCD redistributes the curvature on the object over time, and it observes asymptotically the behaviour of mesh vertices with respect to their neighbourhood.

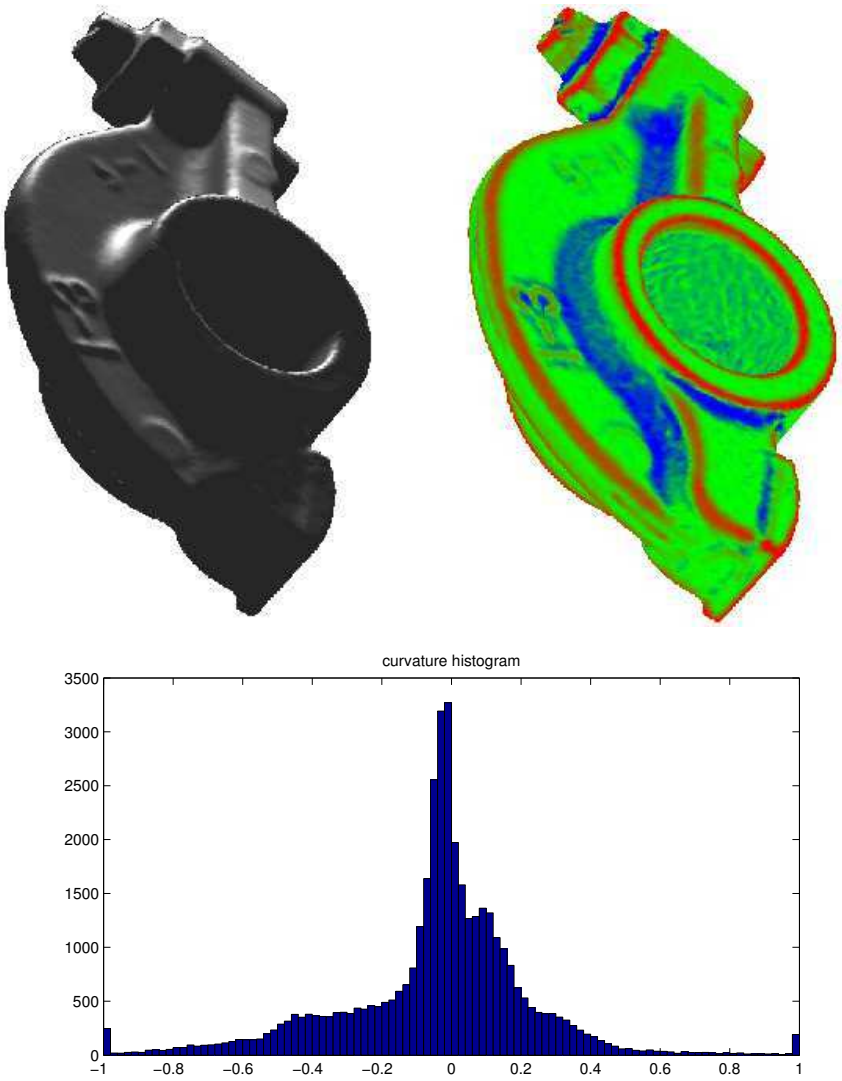

Figure 4.2: Input mesh (*top-left*). Mean curvature H distribution of the input mesh, where red vertices have $H > 0$, blue vertices have $H < 0$ and green vertices have $H \sim 0$ (*top right)*. Frequency histogram of H (*bottom*).

In other words, as in the real physical phenomenon, we can consider the mesh

as influenced by some heat or energy (curvature) sources. At time $t = 0$, the value of H on each vertex v is known, and the sources are disabled instantaneously. Then the heat (curvature) propagates over the surface object until the whole object temperature reaches the equilibrium state at time $t = t_{max}$. In the following, we will study the ideal diffusion of the energy function φ on two particular cases.

In the heat diffusion, at the starting time $t = 0$, heat is concentrated on the edges and corners of the object, which dissipate energy isotropically towards their neighbourhood during the diffusion process. Similarly, feature vertices have greater curvature than their local neighbourhood at $t = 0$ and exchange a few energy (curvature) among themselves, then they must release their curvature to the adjacent vertices on smooth surfaces. On the contrary, as in the physical phenomenon, surface points absorb energy, both the vertices belonging to some surface and their φ_v value must increase over time. Moreover, the absorbed energy is directly proportional to the distance between the surface vertices and some feature-edge.
Since in these two cases the curve $\varphi_v(t)$ is either monotonically decreasing or monotonically increasing, respectively, then SCD classifies the vertices by analysing the curve support described by $\varphi_v(t)$.

In order to improve the segmentation quality, smoothing method can be applied. The main difficulty of mesh fairing is the choice of the suitable number of iterations, thus it can not be embedded in a fully automatic segmentation system. One of the advantages of the proposed method is the implicit curvature fairing, because high frequency noise is typically characterized by low energy which is quickly dissipated during the diffusion process. Furthermore, no anisotropic diffusion is required as we are interested in the total variation of energy on a vertex, over time.

The energy $\varphi_v(t)$ of a vertex v may oscillate over time. For instance, let us consider a vertex v lying on a smooth surface and near some boundary having higher level of curvature. At the beginning of the diffusion process v absorbs energy from the feature-edges of the boundary until some time step $t = t_c$. Nevertheless, for $t > t_c$, v dissipates most of its energy towards the neighbouring vertices characterized by a lower level of curvature. Thus, the trend of $\varphi_v(t)$ should be the same as in Figure 4.3. Furthermore, the trend of energy in a vertex could present several critical points, as a vertex near to several feature-edges with different curvature levels.

The behaviour of this kind of vertices, which is clearly time-scale, imposes that SCD must consider the trend of $\varphi_v(t)$ for the total time interval, until the equilibrium is reached ($t = t_{max}$).

The analysis of the whole trend of the energy curve is required also to suppress high frequency noise that could be present on the surfaces. Indeed, when v is a *noise-vertex*, its energy is rapidly dissipated until some time step $t = t_c$. Then, for the remaining time, it will usually absorb energy from its neighbourhood. This allows SCD to be a powerful unsupervised denoising tool for noise-vertices (false positives) close to the mesh boundaries.

By considering $t = t_{max}$, SCD is also able to classify properly surface features composed by vertices having different ranges of curvature. In fact, if v is a feature vertex with a low value of curvature and close to other vertices with higher energy,

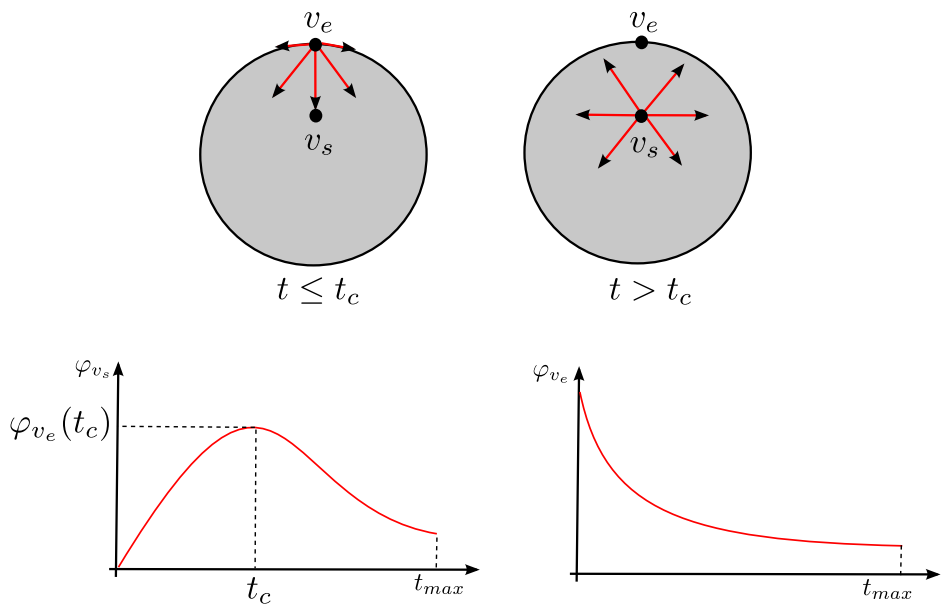

Figure 4.3: Examples of energy diffusion, where the red arrows represent the energy diffusion directions (*top*). Energy diffusion curves for the vertex v_s, on surface, and v_e, on a feature-edge, respectively (*bottom*).

then v will absorbs energy from the neighbouring feature vertices, and then it will release such energy to its neighbourhood.

According to what has been said so far, the change rate of $\varphi_v(t)$, together with the associated time interval, are necessary to evaluate the saliency of a vertex, with respect to its neighbours at different time scales. Specifically, Figure 4.4 shows the diffusion curve of energy for three different vertices on the mesh of Figure 4.2 over time.

Thus, in order to classify surface vertices, the key idea of SCD is to measure, for each vertex, the global energy variation before the diffusion reaches the equilibrium (i.e. the final state). Formally, in the continuous case:

$$\Delta\varphi_v = \int_0^{t_{max}} d\varphi_v = \int_0^{t_{max}} \varphi_v' dt = \varphi_v(t_{max}) - \varphi_v(0)$$

Hence, in the discrete case, the total variation of φ_v can be computed as follows:

$$\Delta\varphi_v = \sum_{i=0}^{t_{max}-1} (\varphi_v(t_{i+1}) - \varphi_v(t_i)) =$$

$$\varphi_v(1) - \varphi_v(0) + \varphi_v(2) - \varphi_v(1) + \cdots + \varphi_v(t_{max}) - \varphi_v(t_{max} - 1) =$$

$$\varphi_v(t_{max}) - \varphi_v(0)$$

The value of $\Delta\varphi_v$ induces a partitioning of the target set S into two sets. The set of vertices lying on smooth surfaces, which are characterized by $\Delta\varphi_v \geq 0$, because

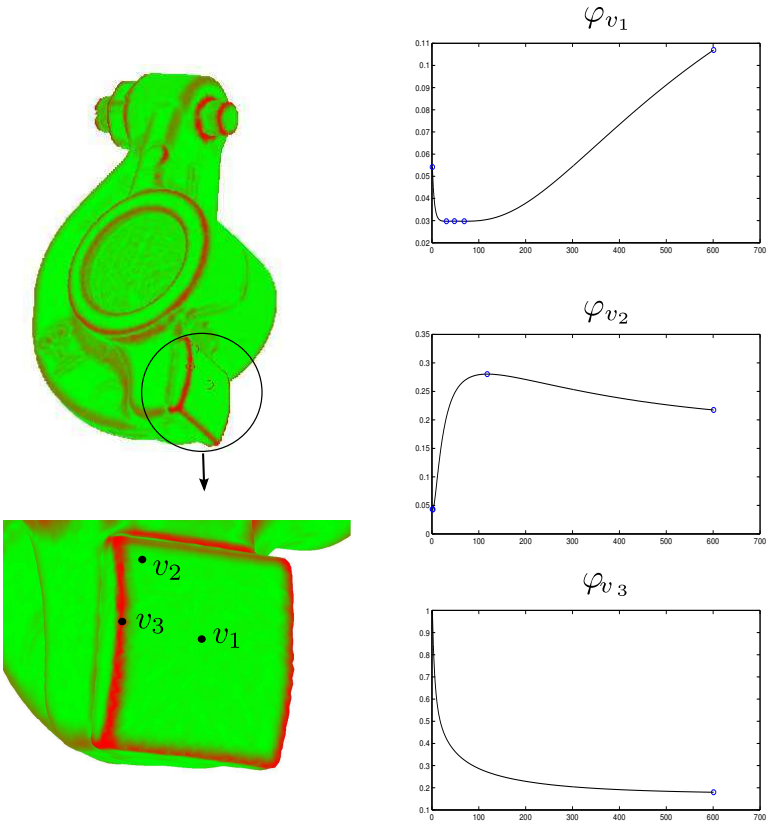

Figure 4.4: Distribution of the absolute value of H on the input mesh (*top-left*). Detail of the input mesh (*bottom-left*) and energy curve support for three different mesh vertices (*right*). The vertices are coloured from green ($|H| \sim 0$) to red ($|H| > 0$).

they receive energy from the neighbourhood, and the set of feature vertices diffusing their energy to the neighbourhood, for which $\Delta\varphi_v < 0$ (see Figure 4.5)

Figure 4.6 shows the partitioning induced by $\Delta\varphi_v$ for three different values of t_{max} and their associated histograms.

For segmentation purposes, we define the following height map:

$$w_\varphi(v) = \begin{cases} -\Delta\varphi_v & if \quad \Delta\varphi_v < 0 \\ \\ 0 & otherwise \end{cases} \tag{4.10}$$

Figure 4.7 shows the feature edges detected by SCD and the relative histograms, by using three different thresholds $t_1 < t_2 < t_3$ of t_{max}. It is clear that $w_\varphi(v)$ is not sensitive to the value of t_{max}. On the contrary, the map $w_{nv}(v)$ obtained by normal voting, strongly depends on the radius of the geodesic window according to the scale of the mesh (see Figure 4.8).

The main steps of the SDC can be summarized as follows:

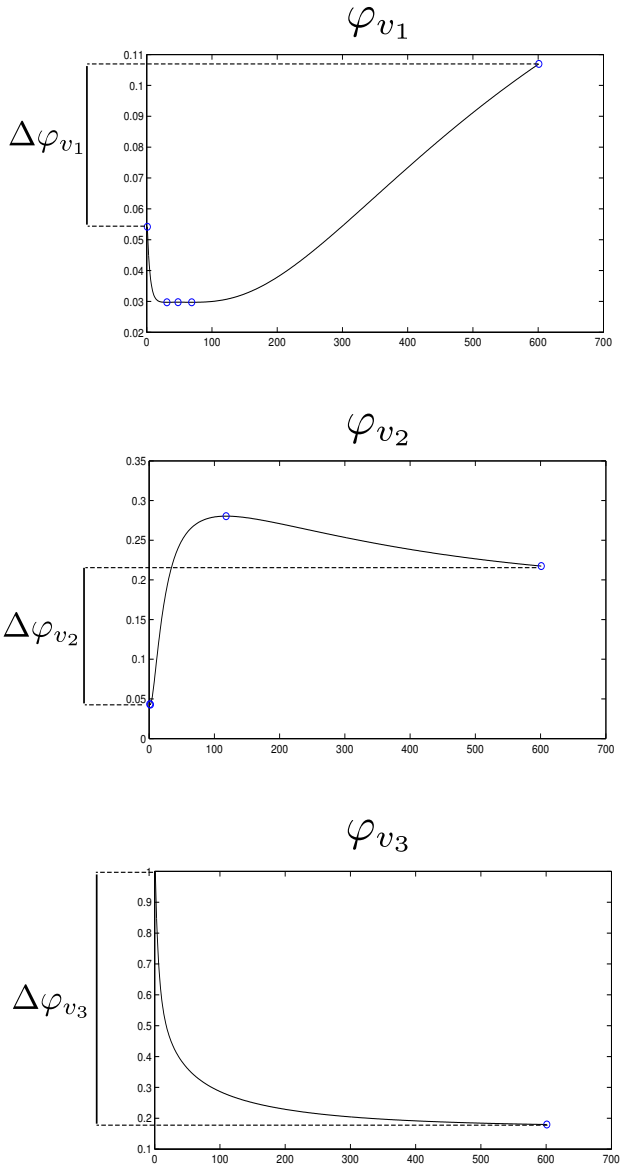

Figure 4.5: Total variation of energy, $\Delta\varphi_v$, used to classify the mesh vertices in Figure 4.4.

1. Compute the energy function on the mesh vertices. In particular, we have used the absolute value of the mean curvature which has been estimated by fitting the mesh vertices with the extended quadric method presented in Section 2.
2. Diffuse the curvature over the mesh and compute $\Delta\varphi_v$.
3. Compute the height map $w_\varphi(v)$.
4. Segment the object by using the watershed algorithm. In particular, we have applied the watershed implementation [64] and the region growing [8].

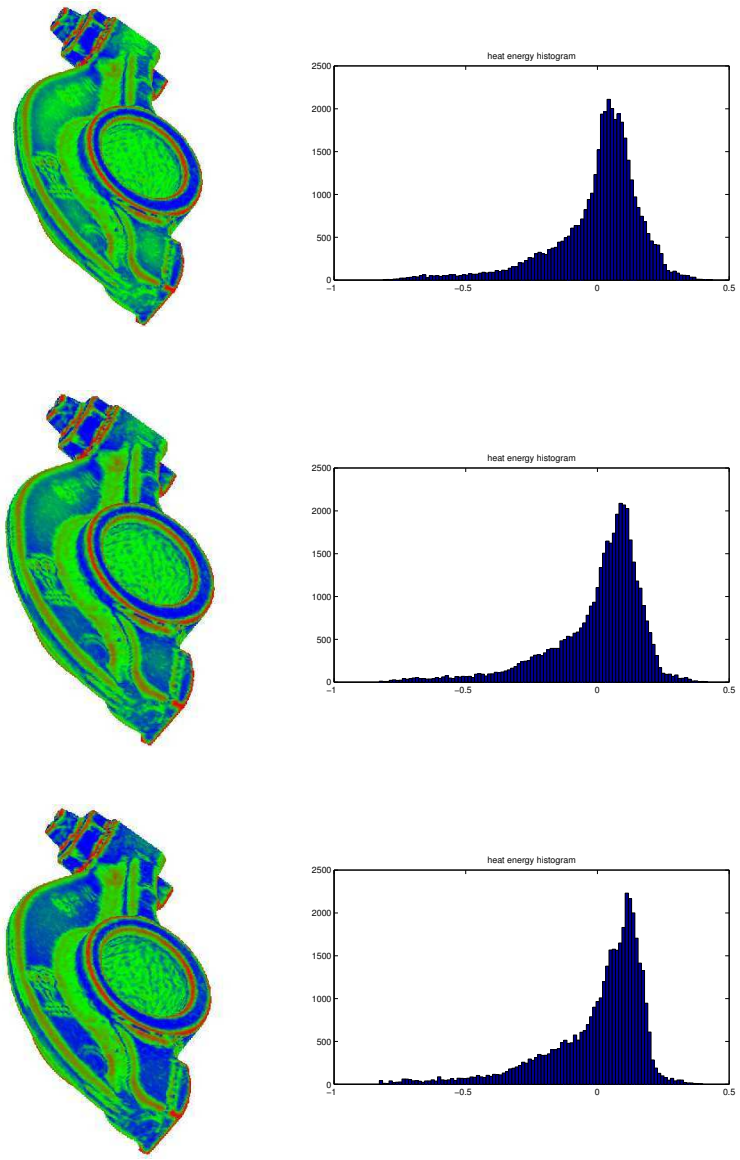

Figure 4.6: Partitioning of a mesh surface using three different value of t_{max}, $t_1 < t_2 < t_3$ (*from top to bottom*). The more intense the blue is, the smaller $\Delta\varphi_v$ is; green color represents $\Delta\varphi_v \sim 0$; the more intense the red is, the greater $\Delta\varphi_v$ is.

4.4 Experimental Results

We have tested SCD on several kinds of noisy triangle meshes coming from datasets publicly available in http://www.cyberware.com/products/scanners/index.html, http://www-rech.telecom-lille1.eu:8080/3dsegbenchmark/dataset.html, and http://shape.cs.princeton.edu/benchmark/.

The watershed transform produces suitable mesh segmentations if the feature-edges describe closed curves. This behaviour represents both an advantage and a

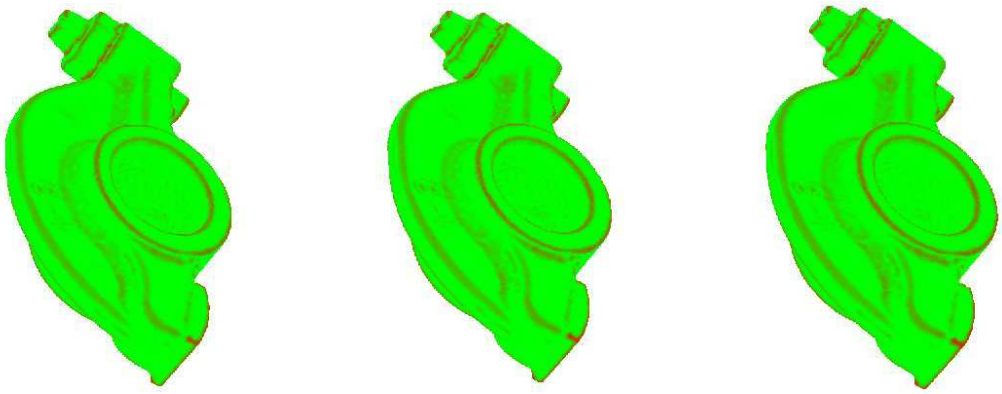

Figure 4.7: Feature-edges located by SCD at time steps $t_1 < t_2 < t_3$ (*from left to right*). The red vertices have $w_\varphi(v) > 0$ while green vertices have $w_\varphi(v) \sim 0$

disadvantage. It is advantageous because the spurious noisy feature vertices which have not been suppressed during the diffusion process are *flooded* and eliminated by the region growing process. When the important feature-edges are not described by closed curves due to noise or defects on the mesh surface, the watershed may break the *segmentation integrity* and merge different areas. In order to reduce this side effect we improve the feature-edges by adding those surface points which are adjacent to at least two feature vertices, and assigning to them the lowest value of $w_\varphi(v)$.

Note that the region merging procedure needs a threshold value to identify the small regions to be merged. We have used the same threshold value for all types of meshes.

The segmentation integrity problem has been also addressed in [8], where the *concave* points are added to the feature-edges. As shown in Figure 4.9 this approach may be very noise sensitive and add a lot of feature vertices lying on smooth surfaces.

As observed in [8], different components of the object are composed by vertices which have elliptic (positive Gaussian curvature, $K > 0$) or parabolic behaviour (null Gaussian curvature, $K = 0$), while the vertices belonging to the regions boundary have hyperbolic behaviour (negative Gaussian curvature, $K < 0$) (see Section 2.1). Since SCD does not distinguish between negative and positive curvature, the segmentation can be considered both component-based and surface-based. Indeed, SCD could use negative Gaussian curvatures to extract the components and successively these components can be further divided according to the positive mean curvature values. Hence, this approach can be used to produce an hierarchy of segments.

Despite no standard methodology exists to compare the outcomes of different segmentation algorithms, some authors proposed to measure the differences between the results of segmentation algorithms and the ground truth segmentation obtained by averaging the manually produced and supervised partitioning performed by human users over the Internet [9].

In the following we will present some results of mesh segmentation. Each located

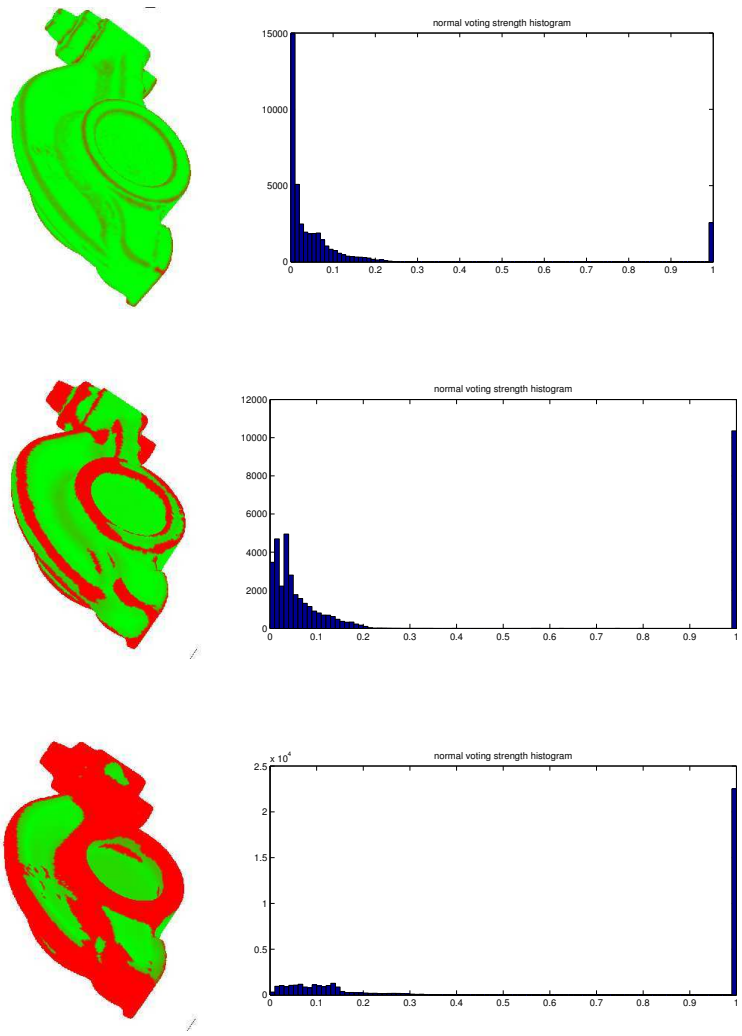

Figure 4.8: Feature-edges located by normal voting [64] (*left*), and relative histograms (*right*). The red vertices have $w_\varphi(v) > 0$, while green vertices have $w_\varphi(v) \sim 0$.

region is identified by a different colour, and its boundary is coloured in red (see Figures from 4.10 to 4.13). SCD is able to locate all important regions of the objects and it suppresses most of the noise. The algorithm maintains small features and very few boundaries are flooded due to their lack of integrity. Only small regions, due to noise, are maintained, as SCD fails to suppress their vertices.

As explained above, feature edges can be composed by vertices having different ranges of curvature. This characteristic heavily affects the threshold-based segmentation which does not perform any local surface analysis. Threshold-based segmentation strongly depends on the functions defined over the mesh vertices and to choose a suitable threshold it needs to observe their histograms.

Figure 4.15 shows three different segmentations obtained on a mechanical object

Figure 4.9: Concave vertices on the mechanic object.

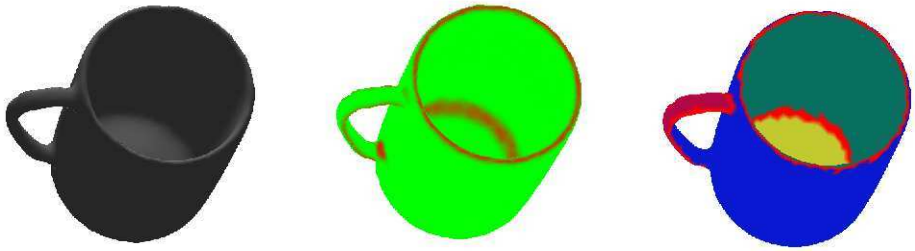

Figure 4.10: Original cup object (*left*). Feature-edges detected by SCD (*middle*). Watershed segmentation (*right*).

by using three different threshold values of the mean curvature. We did not use the operators defined in [25] because they are very noise sensitive. These results also demonstrate that threshold-based segmentation is unstable when compared to SCD.

4.4.1 Conclusions and Future Works

Three-dimensional object analysis is required in several field of research and, according to different applications, objects can have different representations. 3D data are typically obtained from the real world by acquisition devices able to gather either volumetric information (*volumetric data*) about the objects or spatial information about the points belonging to the surface of the object (*range data*).

In this thesis we focused on polygonal meshes which are a very popular 3D data representation, where the object is stored by a cloud of points together with the sets

Figure 4.11: Original screwdriver object (*left*). Feature-edges detected by SCD (*middle*). Watershed segmentation (*right*).

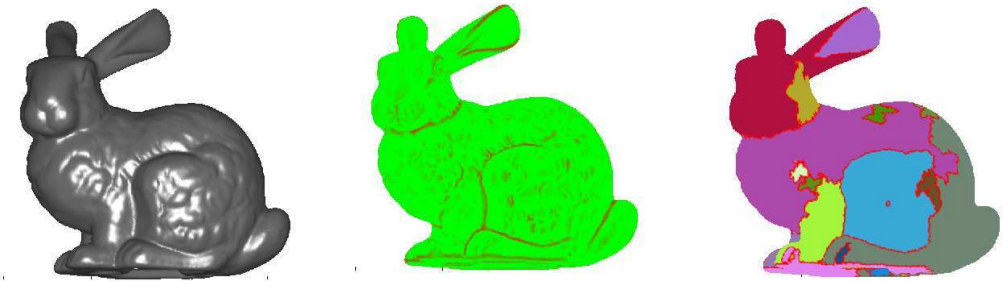

Figure 4.12: Original bunny object (*left*). Feature-edges detected by SCD (*middle*). Watershed segmentation (*right*).

of edges and facets.

Many applications require the segmentation of the objects in order to obtain an higher level representation which simplifies several successive Computer Vision tasks. According to the problem at hand, the decomposition can be either *surface-based* or *component-based*. The former locates those areas on the mesh surface having similar features such as constant curvature, etc. The latter individuates all the semantic components of the object.

In order to segment an object, it needs some *a priori* knowledge about it. The rules used to identify the mesh regions depend on some feature defined over the elements of the surface. The required features can be computed by using different techniques. The differential properties of surfaces are very useful and can be mainly estimated by recovering quadric surfaces fitting the mesh vertices and by spatial averages (finite volumes). In addition, tensor voting is a very powerful method that

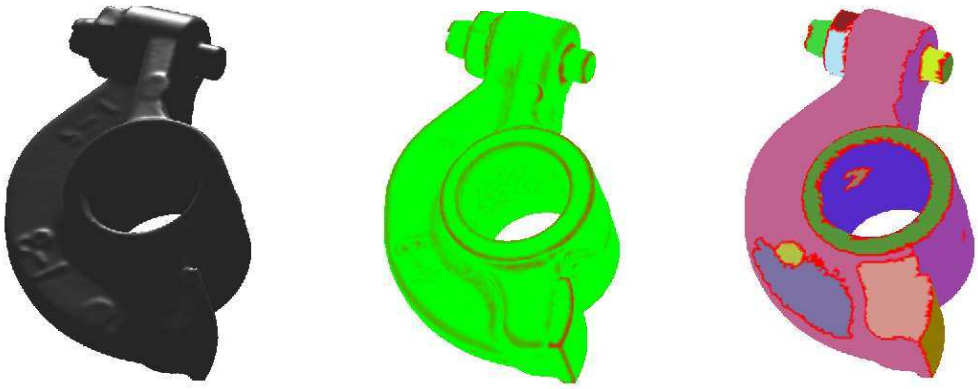

Figure 4.13: Original mechanical object (*left*). Feature-edges detected by SCD (*middle*). Watershed segmentation (*right*).

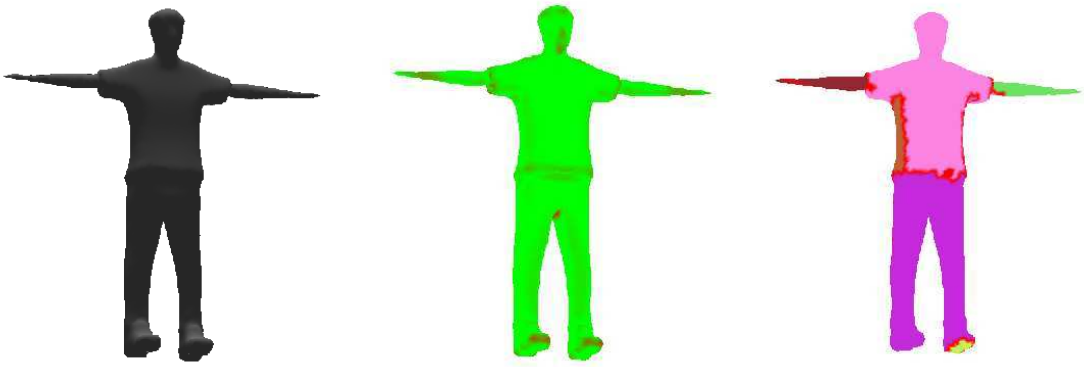

Figure 4.14: Original human model (*left*). Feature-edges detected by SCD (*middle*). Watershed segmentation (*right*).

can be employed to compute robust surface descriptors. Unfortunately there is no general consensus about the best approach to compute surface features. Surface properties are noise dependent and in order to improve their estimation, different surface denoising algorithms have been developed.

Since the segmentation of a mesh is strictly related to graph partitioning which is a NP-Complete problem, then it is required to find approximate solutions efficiently. Several mesh processing algorithms have been presented in literature, each one relying on a particular segmentation paradigm such as *region growing*, *clustering* or *Spectral Analysis*. Mesh segments can be also located by extracting the feature-edges of the mesh, i.e. the discontinuities of its surfaces. Most of the segmentation approaches require to set some parameters or threshold values, and this does not allow to insert these algorithms in automatic segmentation systems.

In this thesis we proposed SCD, that is, a new automatic edge-detection algorithm based on the diffusion process of some energy function defined over the objects surfaces. SCD simulates the physical phenomenon of heat diffusion by using

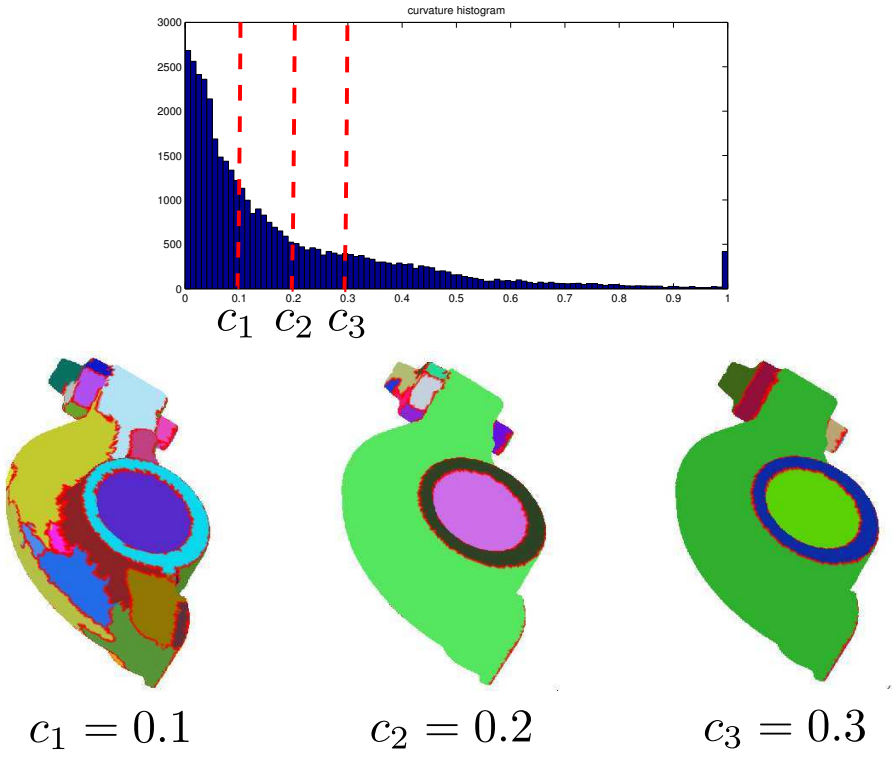

Figure 4.15: Histogram of the mechanic object showing three threshold values (*top*). Watershed segmentation for the chosen threshold levels (*bottom*).

the curvature as power energy and distributes it on the object over time, until the equilibrium state is reached. The total variation of energy is used to classify the surface vertices and to suppress most of the noise. We validated our algorithm on different types of meshes and the results obtained show that SCD is robust, accurate and efficient.

Tensor voting requires to set some parameters to compute the height map and it needs to set the radius of the geodesic window. Thus it depends on the particular scale of the object and moreover the evaluation of the geodesic distance is very time consuming. Threshold-based segmentations highly depend on the energy function defined over the mesh vertices. They do not perform any local analysis in the neighbourhood of the vertices, thus they are not able to locate those important features having low levels of curvature.

These highlighted limitations make SCD attractive for edge-detection and segmentation of meshes obtained from range data.

In the next steps of research we will improve SCD as denoising tool by applying some statistical evaluations of the height map on the feature vertices. Moreover we will perform an hierarchical segmentation by merging the negative Gaussian curvature and the height map produced by SCD.

Some authors proposed recently a benchmark to test the results of segmentation

algorithms against the segmentation ground truth obtained by users [9]. In future works we will use such benchmark to make a quantitative comparison with both the manual segmentation and the outcome of SCD.

Bibliography

[1] A. A. Amini, S. Tehrani, and T. E. Weymouth. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints. In *Computer Vision., Second International Conference on*, pages 95 –99, 1988. 47

[2] B. G. Baumgart. Winged–edge polyhedron representation. Technical report, Department of Computer Science, Stanford University, 1972. 13

[3] J. Berkmann and t. Caelli. Computation of surface geometry and segmentation using covariance techniques. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(11):1114–1116, 1994. 23

[4] I. Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94:115–147, 1987. 33

[5] I. M. Boier-Martin. Domain decomposition for multiresolution analysis. In *SGP '03: Proceedings of the 2003 EurographicsACM SIGGRAPH symposium on Geometry processing*, pages 31–40, 2003. 32

[6] M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Springer-Verlag, London, UK, 1976. 16

[7] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: an experimental study. In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 297–305, 1995. 31

[8] L. Chen and N. D. Georganas. An efficient and robust algorithm for 3d mesh segmentation. *Multimedia Tools Appl.*, 29(2):109–125, 2006. 40, 41, 58, 60

[9] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3d mesh segmentation. *ACM Trans. Graph.*, 28:1–12, 2009. 60, 66

[10] L. Cheng, P. Burchard, B. Merriman, and S. Osher. Motion of curves constrained on surfaces using a level-set approach. *J. Comput. Phys.*, 175(2):604–644, 2002. 47

[11] P. Cignoni, C. Montani, and R. Scopigno. Dewall: A fast divide & conquer Delaunay Triangulation algorithm in ed. *Computer–Aided Design*, 30(5):333–341, 1998. 11

[12] M. Cipolla, F. Bellavia, and C. Valenti. An automatic three-dimensional fuzzy edge detector. In *WILF '09: Fuzzy Logic and Applications, Springer*, pages 221–228, 2009. 42

[13] M. Cipolla, V. Di Gesú, and C. Valenti. A new automatic algorithm for three-dimensional edge detection. In *in Proc. of the Symposium GRID Open Days at the University of Palermo, Consorzio COMETA (Eds)*, pages 221–226, 2007. 43

[14] U. Clarenz, G. Dziuk, and M. Rumpf. *Geometric Analysis and Nonlinear Partial Differential Equations*. S.Hildebrandt /H. Karcher (Eds.), 2003. viii, 50, 51, 52

[15] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 905–914, 2004. 32

[16] H .Delingette. General object reconstruction based on simplex meshes. *Int. J. Comput. Vision*, 32(2):111–146, 1999. 31

[17] M. Desbrun, M. Meyer, P. SchrÃ¶der, and H. B. Alan. Discrete differential-geometry operators in nd. pages 35–57, 2000. vii, 19, 21, 22, 25, 26

[18] R. Dwyer. A faster divide & conquer algorithm for constructing Delaunay Triangulations. *Algorithmica*, 2:137–151, 1987. 11

[19] A. E, Johnson., and M. Herbert. Control of polygonal mesh resolution for 3d computer vision. *Graph. Models Image Process.*, 60(4):261–285, 1998. 12, 13

[20] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974. 31

[21] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997. 12

[22] M. Garland, A. Willmott, and P.S Heckbert. Hierarchical face clustering on polygonal surfaces. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58, 2001. 32, 35

[23] P. L. George. *Automatic Mesh Generation: Applications to Finite Element Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1992. 11

[24] H. Gouraud. Continuous shading of curved surfaces. *IEEE Trans. Comput.*, 20(6):623–629, 1971. 19

[25] A. Hubeli and M. Gross. Multiresolution feature extraction for unstructured meshes. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 287–294, 2001. vii, 41, 42, 62

[26] M. Jung and H. Kim. Snaking across 3d meshes. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 87–93, 2004. 47

[27] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 1(4):321–331, 1988. 45

[28] A. Kaufman and E. Shimony. 3d scan conversion algorithms for voxel based graphics. In *Proceedings of ACM Workshop Interactive 3D Graphics*, pages 45–76, 1986. 8

[29] L. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel. Feature–sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH*, pages 57–66, 2001. 7, 8

[30] L. M. Krsek, P. Krsek, G. LukÃ¡cs, and R. R. Martin. Algorithms for computing curvatures from range data. In *In The Mathematics of Surfaces VIII, Information Geometers*, pages 1–16, 1998. 15

[31] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay Triangulation. *International Journal of Parallel Programming*, 9:219–242, 1980. 11

[32] Y. Lee and S. Lee. Geometric snakes for triangular meshes, 2002. 47

[33] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002. 35

[34] P. Liang and J. S. Todhunter. Representation and recognition of surface shapes in range images: a differential geometry approach. *Comput. Vision Graph. Image Process.*, 52(1):78–109, 1990. 23

[35] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH 87*, pages 163–169, 1987. 7

[36] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 199–208, 1997. 12

[37] A. P. Mangan and R. T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999. 40

[38] S. Mauch. Calculating closest point and distance to a curve. Technical report, AM A Caltech preprint, 1972. 28

[39] N. Max. Weights for computing vertex normals from facet normals. *J. Graph. Tools*, 4(2):1–6, 1999. 19

[40] A. McIvor and P. T. Waltenberg. Recognition of simple curved surfaces from 3d surface data. In *ACCV '98: Proceedings of the Third Asian Conference on Computer Vision-Volume I*, pages 434–441, 1997. 20

[41] A. M. McIvor and R. J. Valkenburg. A comparison of local surface geometry estimation methods. *Mach. Vision Appl.*, 10:17–26, 1997. 19

[42] G. Medioni, M. Lee, and C. Tang. *Computational Framework for Segmentation and Grouping*. Elsevier Science Inc., 2000. 27

[43] D. S. Meek and D. J. Walton. On surface normal and gaussian curvature approximations given data sampled from a smooth surface. *Comput. Aided Geom. Des.*, 17(6):521–543, 2000. 19

[44] A. Modi. Unstructured mesh generation on planes surfaces using graded triangulation. Technical report, Department of Aerospace Engineering, Indian Institute of Bombay, 1997. 11

[45] D. Moore and J. Warren. Approximation of dense scattered data using algebraic surfaces. In *System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference on*, pages 681–690, 1991. 15

[46] M. Nanal. *Mesh Generation for FEM*. PhD thesis, Indian Institute of Technology of Bombay, Department of Civil Engineering, 1995. 11

[47] Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski. Polyhedral surface smoothing with simultaneous mesh regularization. In *GMP '00: Proceedings of the Geometric Modeling and Processing 2000*, page 229, 2000. vii, 26, 27

[48] G. Papaioannou and T. Theoharis. Fast fragment assemblage using boundary line and surface matching. In *Computer Vision and Pattern Recognition Workshop. CVPRW '03. Conference on*, pages 2–2, 2003. 33, 41

[49] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:629–639, 1990. 49

[50] S. Petitjean. A survey of methods for recovering quadrics in triangle meshes. *ACM Comput. Surv.*, 34(2):211–262, 2002. 16, 18, 33, 40

[51] L. Piegl and W. Tiller. *The NURBS book*. Springer-Verlag, London, UK, 1995. 5

[52] J. B. T. Roerdink and A. Meijster. The watershed transform: definitions, algorithms and parallelization strategies. *Fundam. Inf.*, 41(1-2):187–228, 2000. 37

[53] L. Rong and Z. Hao. Segmentation of 3d meshes through spectral clustering. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 298–305, 2004. 36

[54] L. Rong and Z. Hao. Mesh segmentation via spectral embedding and contour analysis, computer graphics forum (eurographics proceedings. *Computer Graphics Forum (Special Issue of Eurographics*, 26(3):2007, 2007. 36

[55] J. Rossignac, , and P. Borrel. Multi–resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993. 12

[56] P. V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, 2001. vii, 33

[57] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, 1992. 12

[58] A. Shamir. A formulation of boundary mesh segmentation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 82–89, 2004. 31

[59] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 36

[60] S. Shlafman, T. Ayellet, and K. Sagi. Metamorphosis of polyhedral surfaces using decomposition. In *Computer Graphics Forum*, pages 219–228, 2002. 35

[61] R. Shu, C. Zhou, and M.S. Kankanhalli. Adaptive Marching Cubes. *The Visual Computer*, 11(4):202–217, 1995. 7

[62] M. Soucy and D. Laurendeau. Multire–solution surface modeling based on hierarchical triangulation. *Comput. Vis. Image Underst.*, 63(1):1–14, 1996. 12

[63] E. M. Stokely and S. Y. Wu. Surface parametrization and curvature measurement of arbitrary 3-d objects: Five practical methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(8):833–840, 1992. 15

[64] Y. Sun, D. L. Page, J. K. Paik, A. Koschan, and M. A. Abidi. Triangle mesh-based edge detection and its application to surface segmentation and adaptive surface smoothing. In *Image Processing. 2002. Proceedings of 2002 International Conference on*, pages 825–828, 2002. vii, viii, 44, 58, 61

[65] G. Taubin. Curve and surface smoothing without shrinkage. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 852, 1995. vii, 25, 27

[66] G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, 1995. vii, 25, 27

[67] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, New Jersey, 1998. 5

[68] Stanford University. The digital Michelangelo Project, 2010. 12

[69] L. Xuetao, T. W. Woon, T. S. Tan, and Z.Huang. Decomposing polygon meshes for interactive applications. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 35–42, 2001. 33

[70] L. Yunjin, L. Seungyong, A. Shamir, D. Cohen-Or, and H. Seidel. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des.*, 22(5):444–465, 2005. vii, 33

[71] z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, 2000. 33, 36

[72] E. Zhang, K. Mischaikow, and G. Turk. Feature–based surface parameterization and texture mapping. *ACM Trans. Graph.*, 24(1):1–27, 2005. 32

[73] E. Zuckerberger. Polyhedral surface decomposition with applications. *Computers & Graphics*, 26(5):733–743, 2002. 32, 33