# 1

# Geometric Features based on Curvatures



## 1.1. Introduction

In 1983, the authors of [HAR 83] proposed to use **differential geometry** (i.e. the mathematical analysis of the geometry of 3D curves and surfaces) to describe the shape of a discrete surface (in their case, defined as an elevation image). For this purpose, they introduced the *topographic primal sketch*, which is composed of feature points (as peaks, pits or saddles), feature lines (as ridges or ravines) and feature surface patches (which may be flat). Two years later, the authors of [BRA 85] described another differential geometry-based framework to analyze the shape of 3D surfaces and they applied it on several real examples. They also used feature lines (as curvature lines) and planar surface patches. During the same period, in his dissertation [BES 88a], Besl proposed to compute the differential parameters of a surface in order to create a *HK-sign map* that allows the segmention of a surface into homogeneous regions where some basic geometric primitives can be fitted.

Since then, extensive research has been done on the extraction and the application of geometric features based on differential geometry. In section 1.2, we propose an overview in a nutshell (but with the mathematical formulas) of the differential geometry of surfaces. In section 1.3, we present the main methods that allow us to efficiently compute the differential parameters on a discrete 3D mesh. In sections 1.4 and 1.5, we focus respectively on line and surface features.

## 1.2. Some mathematical reminders of the differential geometry of surfaces

The following reminders are essentially based on the book [HOS 92]. More details can be found in many mathematics books such as [WEA 55], [CAR 76], [SPI 99], [TOP 05] or [PAT 10].

### 1.2.1. *Fundamental forms and normal curvature*

Let $\Sigma$ be a surface of class $C^k$ with $k \geq 3$, which is parameterized by $(u, v)$. $\Sigma$ is then defined by the set of points $\mathbf{P}(u, v) = \{x(u, v), y(u, v), z(u, v)\}$.

An infinitesimal displacement around the point $\mathbf{P}$ will be modeled as the vector $d\mathbf{P}$, which will be in the tangent plane defined by the frame $(\mathbf{P}, \frac{\partial \mathbf{P}}{\partial u}, \frac{\partial \mathbf{P}}{\partial v})$:

$$d\mathbf{P} = \frac{\partial \mathbf{P}}{\partial u} du + \frac{\partial \mathbf{P}}{\partial v} dv$$

The norm of this displacement can be computed as:

$$(d\mathbf{P})^2 = \left(\frac{\partial \mathbf{P}}{\partial u}\right)^2 (du)^2 + 2\frac{\partial \mathbf{P}}{\partial u}\frac{\partial \mathbf{P}}{\partial v} dudv + \left(\frac{\partial \mathbf{P}}{\partial v}\right)^2 (dv)^2$$

If we define the terms $E$, $F$ and $G$ as:

$$E = \left(\frac{\partial \mathbf{P}}{\partial u}\right)^2 \quad F = \frac{\partial \mathbf{P}}{\partial u}\frac{\partial \mathbf{P}}{\partial v} \quad G = \left(\frac{\partial \mathbf{P}}{\partial v}\right)^2$$

we get:

$$(d\mathbf{P})^2 = E(du)^2 + 2F(dudv) + G(dv)^2 \qquad [1.1]$$

This expression is called the **first fundamental form** of the surface. Its coefficients $E$, $F$ and $G$ enable us to calculate $d\mathbf{P}$ that defines the lengths of local curves on the surface around $\mathbf{P}$. They are also used to define the area of local regions.

Let us now define $\mathbf{n}$ as the normal vector at $\mathbf{P}$, i.e. the vector going through $\mathbf{P}$ and orthogonal to the tangent plane. For any unit vector $\mathbf{t}$ in the tangent

plane, we can define the plane $\Pi_t$ that goes through $\mathbf{P}$ and contains $\mathbf{n}$ and $\mathbf{t}$. $\Pi_t$ is called a *normal plane* to $\Sigma$ and cuts $\Sigma$ along the plane curve $\mathcal{S}_t$, which is called the *normal section* along the direction $\mathbf{t}$ (see Figure 1.1).

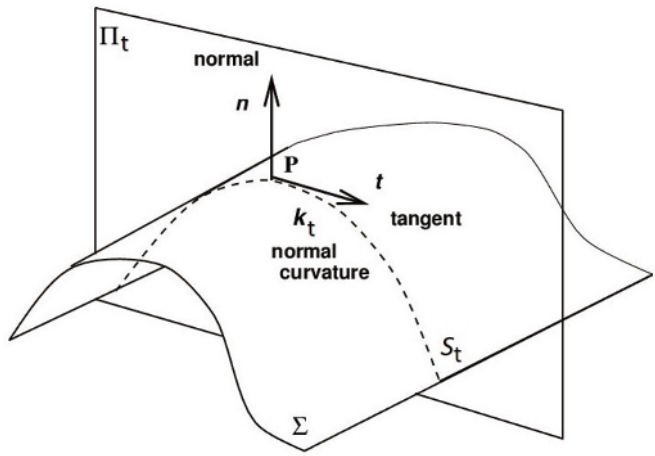



**Figure 1.1.** *At a point $\mathbf{P}$, the normal plane $\Pi_t$ is defined by the normal vector $\mathbf{n}$ and the tangent vector $\mathbf{t}$. $\Pi_t$ cuts $\Sigma$ along a plane curve $\mathcal{S}_t$. The curvature of $\mathcal{S}_t$ in $\mathbf{P}$ is called the* normal curvature *$k_\mathbf{t}$ at $\mathbf{P}$ of the surface $\Sigma$ along the direction $\mathbf{t}$*

By construction, $\mathbf{t}$ and $\mathbf{n}$ are, respectively, the tangent and the normal vector of $\mathcal{S}_t$ at $\mathbf{P}$. The curvature of the plane curve $\mathcal{S}_t$ at $\mathbf{P}$ is an interesting parameter to characterize the local shape of $\Sigma$ around $\mathbf{P}$ along the direction $\mathbf{t}$. This curvature, denoted $k_\mathbf{t}$, is given by the first Frenet–Serret formula that relates the tangent and the normal vectors of a planar curve:

$$\frac{d\mathbf{t}}{ds} = k_\mathbf{t}\,\mathbf{n} \qquad [1.2]$$

where $s$ is the arclength (also called the curvilinear abscissa) along $\mathcal{S}_t$ that is defined by $ds = |d\mathbf{P}|$.

This implies:

$$k_\mathbf{t} = \frac{d\mathbf{t}}{ds} \cdot \mathbf{n} \qquad [1.3]$$

We can develop the right expression with respect to the parameters $(u, v)$ by using the chain rule:

$$\mathbf{t} = \frac{d\mathbf{P}}{ds} = \frac{\partial \mathbf{P}}{\partial u}\left(\frac{du}{ds}\right) + \frac{\partial \mathbf{P}}{\partial v}\left(\frac{dv}{ds}\right) \quad [1.4]$$

$$\frac{d\mathbf{t}}{ds} = \frac{\partial \mathbf{P}}{\partial u}\left(\frac{d^2u}{ds^2}\right) + \frac{\partial \mathbf{P}}{\partial v}\left(\frac{d^2v}{ds^2}\right) + \frac{\partial^2 \mathbf{P}}{\partial u^2}\left(\frac{du}{ds}\right)^2 + 2\frac{\partial \mathbf{P}}{\partial u}\frac{\partial \mathbf{P}}{\partial v}\left(\frac{du}{ds}\right)\left(\frac{dv}{ds}\right) + \frac{\partial^2 \mathbf{P}}{\partial v^2}\left(\frac{dv}{ds}\right)^2$$

A dot product with $\mathbf{n}$ eliminates the two first terms because $\dfrac{\partial \mathbf{P}}{\partial u}$ and $\dfrac{\partial \mathbf{P}}{\partial v}$ belong to the tangent plane which is orthogonal to $\mathbf{n}$. We then get:

$$k_{\mathbf{t}} = \frac{d\mathbf{t}}{ds} \cdot \mathbf{n} = L\left(\frac{du}{ds}\right)^2 + 2M\left(\frac{du}{ds}\right)\left(\frac{dv}{ds}\right) + N\left(\frac{dv}{ds}\right)^2 \quad [1.5]$$

where:

$$L = \mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial u^2} \quad M = \mathbf{n} \cdot \frac{\partial \mathbf{P}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial v} \quad N = \mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial v^2}$$

The parameters $L$, $M$ and $N$ are called the coefficients of the **second fundamental form** of the surface.

As $ds^2 = (d\mathbf{P})^2$, we can rewrite equation [1.5] by using equation [1.1] as:

$$k_{\mathbf{t}} = \frac{L(du)^2 + 2Mdudv + N(dv)^2}{E(du)^2 + 2Fdudv + G(dv)^2} \quad [1.6]$$

$k_{\mathbf{t}}$ is called the **normal curvature** of $\Sigma$ at point $\mathbf{P}$ along the direction $\mathbf{t}$.

Note that another convention exists, where the sign of the curvature is inverted (i.e. $d\mathbf{t}/ds = -k_{\mathbf{t}}\,\mathbf{n}$). This changes the sign in most of the mathematical expressions given in this chapter, as explained in Table 3.2 of [PAT 10].

### 1.2.2. *Principal curvatures and shape index*

Now, if we set $\gamma = dv/du$, defined for all directions except along the $u$ isoparametric lines, we can define any tangent vector $\mathbf{t}$ with respect to $\gamma$ using equation [1.4]:

$$\mathbf{t} = \frac{du}{ds}\left(\frac{\partial \mathbf{P}}{\partial v} + \gamma \frac{\partial \mathbf{P}}{\partial u}\right) \qquad [1.7]$$

Let us now parameterize the normal curvature $k_{\mathbf{t}}$ by $\gamma$ and write it as $k(\gamma)$. As $dv = \gamma du$, equation [1.6] becomes:

$$k(\gamma) = \frac{L + 2M\gamma + N\gamma^2}{E + 2F\gamma + G\gamma^2}$$

$$(L - k(\gamma)E) + 2(M - k(\gamma)F)\gamma + (N - k(\gamma)G)\gamma^2 = 0 \qquad [1.8]$$

Let us study the extremal values when the cutting plane $\Pi_t$ rotates around the axis defined by $(\mathbf{P}, \mathbf{n})$. $k(\gamma)$ is then a periodic function as the plane $\Pi_t$ is invariant after a half-turn. Moreover, as $k$ is a continuous function of $\gamma$, there are a maximum and a minimum which are defined by:

$$\frac{dk(\gamma)}{d\gamma} = 0$$

If we differentiate equation [1.8] with respect to $\gamma$ and if we apply the above extremum condition, we get:

$$(M - k(\gamma)F) + \gamma(N - k(\gamma)G) = 0 \qquad [1.9]$$

And if we substitute this result in equation [1.8]:

$$(L - k(\gamma)E) + \gamma(M - k(\gamma)F) = 0 \qquad [1.10]$$

We can now write $\gamma$ as a function of $k(\gamma)$ in the first equation and eliminate $\gamma$ in the second one, which leads to:

$$(EG - F^2)k(\gamma)^2 + (EN + LG - 2MF)k(\gamma) + LN - M^2 = 0 \qquad [1.11]$$

We obtain a quadratic equation in $k(\gamma)$. In the general case, the two roots correspond to the two extrema (one maximum and one minimum) of the normal curvature which are called the **principal curvatures** and are denoted $k_1$ and $k_2$.

The arithmetic mean of the two principal curvatures $k_m = \frac{1}{2}(k_1 + k_2)$ is called the **mean curvature**.

The product of the two principal curvatures $k_g = k_1.k_2$ is called the **Gaussian curvature**. In particular, the Gaussian curvature allows us to define the local shape of the surface as:

– $k_g > 0$: the two principal curvatures have the same sign. It implies that all the normal curvatures are of the same sign regardless of the direction. This means that the surface is either locally convex or locally concave;

– $k_g < 0$: the two principal curvatures have an opposite sign. The normal curvature changes its sign and vanishes along a direction. This means that locally the surface goes through its tangent plane.

In the case where there is a unique root, it means that $k_1 = k_2$ and that the normal curvature has a constant value regardless of the direction. This is possible only if the surface is locally plane or spherical around the point $\mathbf{P}$. Such a point is then called an **umbilic**.

In order to characterize the shape of a surface with a unique parameter, the **shape index** was proposed in [KOE 92]:

$$s = \frac{2}{\pi} \arctan \frac{k_2 + k_1}{k_2 - k_1} \qquad [1.12]$$

$s$ takes its values in $[-1, 1]$. Positive (respectively negative) values correspond to convex (resp. concave) parts and the extremal values 1 or $-1$ characterize umbilics.

### 1.2.3. *Principal directions and lines of curvature*

We call **principal directions** the directions of the tangent vectors along which the normal curvature is extremal. In the general case, we have two

distinct principal directions corresponding to the principal curvatures $k_1$ and $k_2$ which are defined by the two tangent vectors $\mathbf{t_1}$ and $\mathbf{t_2}$. Note that we talk of directions so the sense of the tangent vectors must not be taken into account. Thus, the principal direction associated with $k_1$ can be defined by either $\mathbf{t_1}$ or $-\mathbf{t_1}$. If $\mathbf{P}$ is an umbilic, we consider that all the directions are principal.

Except in this last case, we can calculate the dot product between the two principal directions. For this, we use the expression of the tangent vector given in equation [1.7]:

$$\mathbf{t_1} \cdot \mathbf{t_2} = \frac{du_1}{ds_1}\left(\frac{\partial \mathbf{P}}{\partial u} + \gamma_1 \frac{\partial \mathbf{P}}{\partial v}\right) \cdot \frac{du_2}{ds_2}\left(\frac{\partial \mathbf{P}}{\partial u} + \gamma_2 \frac{\partial \mathbf{P}}{\partial v}\right)$$

By developing and using $E$, $F$ and $G$, we get:

$$\mathbf{t_1} \cdot \mathbf{t_2} = (E + (\gamma_1 + \gamma_2)F + \gamma_1\gamma_2 G)\frac{du_1}{ds_1}\frac{du_2}{ds_2} \qquad [1.13]$$

Now, we are going to define a quadratic equation whose solutions are the values corresponding to $\gamma_1$ and $\gamma_2$. For this, we write $k(\gamma)$ as a function of $\gamma$ by using equation [1.9] and eliminate it in equation [1.10]. We obtain:

$$(MG - NF)\gamma^2 + (GL - NE)\gamma + FL - ME = 0 \qquad [1.14]$$

This allows us to compute the sum and the product of the roots of this equation as:

$$\gamma_1 + \gamma_2 = \frac{NE - GL}{MG - NF}$$
$$\gamma_1\gamma_2 = \frac{FL - ME}{MG - NF}$$

If we use these two expressions in equation [1.13], we get:

$$\mathbf{t_1} \cdot \mathbf{t_2} = 0$$

This proves that the principal directions are **orthogonal** (except when the point is an umbilic).

We can also draw on the surface $\Sigma$ the curves which are tangent, at each point, to the direction of one of the principal curvatures $\mathbf{t_1}$ or $\mathbf{t_2}$. These curves are called **lines of curvature**. Due to the properties of the principal curvatures, we can conclude that two lines of curvature pass through each non-umbilic point and that they are orthogonal.

As we can see on the example of the ellipsoid in Figure 1.2, the lines of curvature form a network of orthogonal curves all over the surface except at umbilics. Note that for this last particular case, the local pattern of lines of curvature has been intensively studied in [SOT 08].

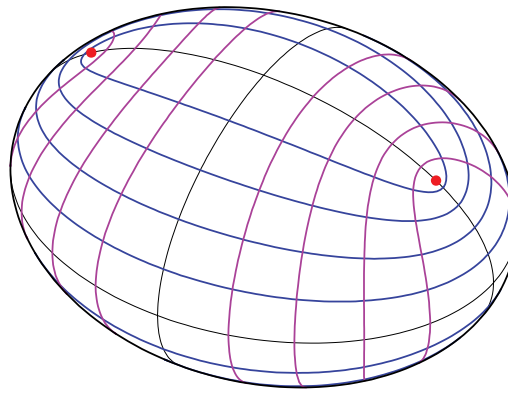


**Figure 1.2.** *Network of lines of curvatures on an ellipsoid: some lines of curvature are visualized in blue and pink, the two visible umbilics (the two others are on the opposite) are localized as red dots*[1]*. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

At any point $\mathbf{P}$ of the surface $\Sigma$, we have an orthogonality relationship between the surface normal $\mathbf{n}$ and any tangent vector $\mathbf{t}$: $\mathbf{n} \cdot \mathbf{t} = 0$.

Now, if we move on the surface along $\mathbf{t}$ by a distance $ds$, we can then write:

$$\frac{d(\mathbf{n} \cdot \mathbf{t})}{ds} = \frac{d\mathbf{n}}{ds} \cdot \mathbf{t} + \mathbf{n} \cdot \frac{d\mathbf{t}}{ds} = 0$$

1 http://commons.wikimedia.org/wiki/File%3AEllipso-KL-NP.svg by Ag2gaeh [CC BY-SA 4.0]

By using equation [1.3], we get:

$$\frac{d\mathbf{n}}{ds} \cdot \mathbf{t} + k_{\mathbf{t}} = 0 \quad \text{or} \quad \frac{d\mathbf{n}}{ds} \cdot \mathbf{t} = -k_{\mathbf{t}} \qquad [1.15]$$

If $\mathbf{t}$ is a principal direction, we have seen that $k_{\mathbf{t}}$ is extremal. This implies that the variation of $\mathbf{n}$ according to $\mathbf{t}$ is extremal along the principal directions. Lines of curvature correspond then to the curves of the surfaces where the normal vector of a surface varies the most.

### 1.2.4. *Weingarten equations and shape operator*

We can write for any point $\mathbf{P}$ of the surface $\Sigma$:

$$\frac{\partial}{\partial u}\left(\mathbf{n} \cdot \frac{\partial \mathbf{P}}{\partial u}\right) = \frac{\partial \mathbf{n}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial u} + \mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial u^2} \qquad [1.16]$$

However, as $\mathbf{n}$ is orthogonal to $\frac{\partial \mathbf{P}}{\partial u}$ and $\frac{\partial \mathbf{P}}{\partial v}$, which belong to the tangent plane, we have:

$$\mathbf{n} \cdot \frac{\partial \mathbf{P}}{\partial u} = 0$$

which implies that equation [1.16] is always equal to 0. We have then by using the coefficients of the fundamental forms (see section 1.2.1):

$$\frac{\partial \mathbf{n}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial u} = -\mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial u^2} = -L \qquad [1.17]$$

and by exchanging $u$ and $v$ in equation [1.16]:

$$\frac{\partial \mathbf{n}}{\partial v} \cdot \frac{\partial \mathbf{P}}{\partial v} = -\mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial v^2} = -N$$

$$\frac{\partial \mathbf{n}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial v} = -\mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial u \partial v} = -M$$

$$\frac{\partial \mathbf{n}}{\partial v} \cdot \frac{\partial \mathbf{P}}{\partial u} = -\mathbf{n} \cdot \frac{\partial^2 \mathbf{P}}{\partial u \partial v} = -M$$

This set of four equations will allow us to write around $\mathbf{P}$ a linear relationship between the variations of the normal vector and of the point

position. This linear relationship can be represented by a set of two equations called **Weingarten equations**:

$$\frac{\partial \mathbf{n}}{\partial u} = a\frac{\partial \mathbf{P}}{\partial u} + b\frac{\partial \mathbf{P}}{\partial v}$$
$$\frac{\partial \mathbf{n}}{\partial v} = c\frac{\partial \mathbf{P}}{\partial u} + d\frac{\partial \mathbf{P}}{\partial v}$$

or by a linear map called the **shape operator**, which can be represented by the matrix $S(\mathbf{P})$:

$$\begin{pmatrix} \frac{\partial \mathbf{n}}{\partial u} \\ \frac{\partial \mathbf{n}}{\partial v} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \frac{\partial \mathbf{P}}{\partial u} \\ \frac{\partial \mathbf{P}}{\partial v} \end{pmatrix} \quad [1.18]$$

By using equation [1.4], we can then write:

$$\begin{aligned} S(\mathbf{P})\mathbf{t} &= S(\mathbf{P}) \left( \frac{\partial \mathbf{P}}{\partial u} \left( \frac{du}{ds} \right) + \frac{\partial \mathbf{P}}{\partial v} \left( \frac{dv}{ds} \right) \right) \\ &= \frac{\partial \mathbf{n}}{\partial u} \left( \frac{du}{ds} \right) + \frac{\partial \mathbf{n}}{\partial v} \left( \frac{dv}{ds} \right) = \frac{d\mathbf{n}}{ds} \end{aligned}$$

And then, by using equation [1.15], we get the main property of the shape operator:

$$S(\mathbf{P})\mathbf{t} \cdot \mathbf{t} = -k_{\mathbf{t}}$$

for any tangent vector $\mathbf{t}$.

Now, let us compute the coefficients $a$, $b$, $c$ and $d$. If we take the first Weingarten equation (that corresponds to the first line of the shape operator) and if we apply a dot product with $\frac{\partial \mathbf{P}}{\partial u}$, we get:

$$\frac{\partial \mathbf{n}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial u} = a \left( \frac{\partial \mathbf{P}}{\partial u} \right)^2 + b\frac{\partial \mathbf{P}}{\partial v} \cdot \frac{\partial \mathbf{P}}{\partial u}$$

By using equation [1.17] and the coefficients of the first fundamental form (see section 1.2.1), we end up with:

$$-L = aE + bF$$

Similarly, it is easy to show that:

$$-M = aF + bG \quad M = cE + dF \quad N = cF + dG$$

By solving the system of two equations for $a$ and $b$, we obtain:

$$a = \frac{MF - LG}{EG - F^2} \quad b = \frac{LF - ME}{EG - F^2}$$

And with the system of two equations for $c$ and $d$, we have:

$$c = \frac{NF - MG}{EG - F^2} \quad d = \frac{MF - NE}{EG - F^2}$$

We can then write explicitly the shape operator matrix:

$$S(\mathbf{P}) = \frac{1}{EG - F^2} \begin{pmatrix} MF - LG & LF - ME \\ NF - MG & MF - NE \end{pmatrix} \qquad [1.19]$$

By computing the trace and the determinant, it is easy to see that the eigenvalues of the matrix $S(\mathbf{P})$ are given by equation [1.11]. It means that the eigenvalues correspond to the principal curvatures $k_1(\mathbf{P})$ and $k_2(\mathbf{P})$. This implies that:

$$k_m(\mathbf{P}) = \frac{1}{2}\text{Tr}(S(\mathbf{P})) \qquad \text{and} \qquad k_g(\mathbf{P}) = \det(S(\mathbf{P}))$$

Now let us find the eigenvectors $\mathbf{e_1}$ and $\mathbf{e_2}$. We can write their coordinates in the tangent frame $(\frac{\partial \mathbf{P}}{\partial u}, \frac{\partial \mathbf{P}}{\partial v})$ by using equation [1.7]:

$$\mathbf{e_i} = \frac{du}{ds}\left(\frac{\partial \mathbf{P}}{\partial u} + \gamma_i \frac{\partial \mathbf{P}}{\partial v}\right)$$

By writing the eigenvalue condition: $S(\mathbf{P})\mathbf{e_i} = \lambda_i \mathbf{e_i}$ where $\lambda_i \in \mathbb{R}$, we get:

$$\begin{cases} a + b\gamma_i = \lambda_i \\ c + d\gamma_i = \lambda_i \gamma_i \end{cases}$$

which gives $b{\gamma_i}^2 + (a - d)\gamma_i - c = 0$.

If we set $x_i = -\frac{1}{\gamma_i}$, we obtain the equation $-cx_i^2 + (d-a)x_i + b = 0$ which has the same coefficients as equation [1.14]. The two solutions $x_i$ then correspond to the principal direction vectors $\mathbf{t_i}$ whose coordinates are $(1, x_i)$ or $(1, -\frac{1}{\gamma_i})$ in the tangent frame defined below. By construction, the vectors $\mathbf{e_i}$ are orthogonal to $\mathbf{t_i}$ so they also define the principal direction frame. The eigenvectors of the shape operator matrix $S(\mathbf{P})$ are then the principal directions $\mathbf{t_1}$ and $\mathbf{t_2}$ and we can write in the principal frame defined by $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$:

$$\frac{\partial \mathbf{n}}{\partial u} = k_1 \frac{\partial \mathbf{P}}{\partial u} \qquad \frac{\partial \mathbf{n}}{\partial v} = k_2 \frac{\partial \mathbf{P}}{\partial v} \qquad [1.20]$$

### 1.2.5. *Practical computation of differential parameters*

It is straightforward to compute differential parameters on the surface of geometric primitives by considering some basic geometric properties.

Thus, on a sphere of radius $r$, any intersection by a normal plane gives a circle centered at the center of the sphere. The normal curvature along any direction is then equal to $k_\mathbf{t} = 1/r$, which means that all the points are umbilics with $k_1 = k_2 = k_m = 1/r$ and $k_g = 1/r^2$.

At any point of a right cylinder of radius $r$, the surface is locally convex. This means that the sign of the normal curvature is positive regardless of the direction. If we intersect the cylinder at a point by a normal plane which goes through the axis, we obtain a straight line. The normal curvature at this point is then equal to $0$, and this is an extremal value as all the normal curvatures are positive. This implies that, at any point, we have $k_1 = 0$ and that the corresponding principal direction is along the cylinder generatrix. The other principal direction is orthogonal to the first principal direction, which implies that it defines a transverse plane, cutting the cylinder as a circle of radius $r$. We deduce then that $k_2 = 1/r$.

For more complex geometric surfaces given by a parametric representation $(u, v)$, we usually compute the parameters of the first and second fundamental forms. This then allows us to solve equation [1.11] to compute the principal curvatures $k_1$ and $k_2$ (only one value in the case of an umbilic). Then, we can solve equation [1.14] to get two values (except for umbilics) of $\gamma$ in order to compute the two principal directions $\mathbf{t_1}$ and $\mathbf{t_2}$ by equation [1.7].

We can also compute the coefficients of the shape operator matrix by using formulas given in equation [1.19]. The eigenvalues and the eigenvectors of this $2 \times 2$ matrix give the principal curvatures and directions.

Note that in the specific case, called *Monge form*, where the surface $\Sigma$ is defined by $\mathbf{P}(u,v) = (u,v,w(u,v))$, the equations become simpler (see, for example, [HAM 93]) as we have:

$$E = 1 + \left(\frac{\partial w}{\partial u}\right)^2 \quad F = \frac{\partial w}{\partial u}\frac{\partial w}{\partial v} \quad G = 1 + \left(\frac{\partial w}{\partial v}\right)^2 \qquad [1.21]$$

With $D = 1 + \left(\frac{\partial w}{\partial u}\right)^2 + \left(\frac{\partial w}{\partial v}\right)^2$ we get:

$$L = \frac{1}{\sqrt{D}}\frac{\partial^2 w}{\partial u^2} \quad M = \frac{1}{\sqrt{D}}\frac{\partial^2 w}{\partial u \partial v} \quad N = \frac{1}{\sqrt{D}}\frac{\partial^2 w}{\partial v^2}$$

$$k_g(\mathbf{P}) = \frac{1}{D^2}\left(\frac{\partial^2 w}{\partial u^2}\frac{\partial^2 w}{\partial v^2} - (\frac{\partial^2 w}{\partial u \partial v})^2\right)$$

$$k_m(\mathbf{P}) = \frac{1}{2D^{\frac{3}{2}}}\left(\left(1 + \frac{\partial w}{\partial v}^2\right)\frac{\partial^2 w}{\partial u^2} - 2\frac{\partial w}{\partial u}\frac{\partial w}{\partial v}\frac{\partial^2 w}{\partial u \partial v} + \left(1 + \frac{\partial w}{\partial u}^2\right)\frac{\partial^2 w}{\partial v^2}\right)$$

which allows us to easily compute the principal curvatures $k_1(\mathbf{P})$ and $k_2(\mathbf{P})$.

We can also obtain mathematical formulations when we use other type of parameterization as an implicit equation involving the three coordinates (see, for example, [GOL 05]).

### 1.2.6. *Euler's theorem*

Let us parameterize the surface $\Sigma$ by the lines of curvature. In this case, at a point $\mathbf{P}$, the tangent vectors of the isoparametric curves correspond to the principal directions $(\mathbf{t_1}, \mathbf{t_2})$ which are orthogonal. We then have:

$$F = \frac{\partial \mathbf{P}}{\partial u} \cdot \frac{\partial \mathbf{P}}{\partial v} = 0$$

and by writing equation [1.9] with $\gamma = 0$, we get $M = 0$. Equation 1.3 which gives the curvature along any tangent direction $\mathbf{t}$ then becomes:

$$k_{\mathbf{t}} = \frac{L(du)^2 + N(dv)^2}{E(du)^2 + G(dv)^2} \qquad [1.22]$$

In particular, the principal curvatures that correspond respectively to $dv = 0$ and $du = 0$ are given by:

$$k_1 = L/E \qquad k_2 = N/G \qquad [1.23]$$

Any tangent vector $\mathbf{t}(\theta)$ which forms an angle $\theta$ with the principal direction $\mathbf{t_1}$ can be written as $\mathbf{t}(\theta) = \cos\theta\ \mathbf{t_1} + \sin\theta\ \mathbf{t_2}$ and we can write according to equation [1.4]:

$$\mathbf{t}(\theta) \cdot \mathbf{t_1} = \cos\theta = \left( \frac{\partial \mathbf{P}}{\partial u} \left( \frac{du}{ds} \right) + \frac{\partial \mathbf{P}}{\partial v} \left( \frac{dv}{ds} \right) \right) \cdot \mathbf{t_1}$$

As $\mathbf{t_1}$ is a unit vector, we have:

$$\frac{\partial \mathbf{P}}{\partial u} = \left\| \frac{\partial \mathbf{P}}{\partial u} \right\| \mathbf{t_1}$$

As $\dfrac{\partial \mathbf{P}}{\partial v}$ is orthogonal to $\mathbf{t_1}$, this gives:

$$\mathbf{t}(\theta) \cdot \mathbf{t_1} = \left\| \frac{\partial \mathbf{P}}{\partial u} \right\| \frac{du}{ds} = \sqrt{E}\, \frac{du}{ds} = \cos\theta$$

Identically, we get:

$$\sqrt{G}\, \frac{dv}{ds} = \sin\theta$$

Based on the above results, we can now write:

$$k_1 \cos^2\theta + k_2 \sin^2\theta = L \left( \frac{du}{ds} \right)^2 + N \left( \frac{dv}{ds} \right)^2$$

As $F = 0$, we have by equation [1.1]: $ds^2 = (d\mathbf{P})^2 = E(du)^2 + G(dv)^2$. We can then replace the denominator and we obtain by equation [1.22]:

$$k_1 cos^2\theta + k_2 sin^2\theta = \frac{L(du)^2 + N(dv)^2}{E(du)^2 + G(dv)^2} = k_{\mathbf{t}}(\theta) \quad [1.24]$$

This formula, known also as **Euler's theorem**, allows us to compute the value of the directional curvature at a point, along any tangent vector, given only the principal curvatures and directions.

### 1.2.7. *Meusnier's theorem*

Let us study the curve $\mathcal{S}_t$ around $\mathbf{P}$. Locally, we can represent this plane curve in the frame $(\mathbf{P}, \mathbf{t}, \mathbf{n})$ by a Cartesian representation $y = f(x)$. We can write Taylor's expansion of $f$ around $\mathbf{P}$:

$$f(x) = f(0) + f'(0)x + \frac{1}{2}f''(0)x^2 + \epsilon$$

At $\mathbf{P}$, $f(0) = 0$ and $f'(0) = 0$ as $\mathbf{t}$ is a tangent vector to $\mathcal{S}_t$. We can then write:

$$f''(0) = \lim_{\mathbf{P}} \frac{2f(x)}{x^2}$$

The general formula for the curvature $k$ of a 2D function $f$ is given by:

$$k = \frac{f''(x)}{(1 + f'^2(x))^{\frac{3}{2}}}$$

If we apply it to the curve $\mathcal{S}_t$ at $\mathbf{P}$, we get:

$$k_{\mathcal{S}_t}(\mathbf{P}) = f''(0) = \lim_{\mathbf{P}} \frac{2f(x)}{x^2} \quad [1.25]$$

Now, let us consider, at point $\mathbf{P}$, a normal plane $\Pi_t$ directed by the tangent vector $\mathbf{t}$. If we rotate this plane along the axis $(\mathbf{P}, \mathbf{t})$ by an angle $\theta$, we get a new plane that we call $\Pi_t^\theta$. This plane cuts the surface $\Sigma$ along the curve $\mathcal{S}_t^\theta$.

The normal vector of $\mathcal{S}_t^\theta$ at $\mathbf{P}$ is $\mathbf{n}^\theta$ whereas the tangent vector is, by construction, $\mathbf{t}$.

The plane curve $\mathcal{S}_t^\theta$ can be parameterized by $(x(t), y(t))$ in the frame of the plane $\Pi_t^\theta$ defined by $(\mathbf{t}, \mathbf{n}^\theta)$. We can compute its 2D curvature at $\mathbf{P}$ called $k_t^\theta$ using equation [1.25]:

$$k_t^\theta = \lim_{\mathbf{P}} \frac{2y(t)}{x^2(t)}$$

However, we can also parameterize $\mathcal{S}_t^\theta$ as a 3D curve $(X(u), Y(u), Z(u))$ in the frame $(\mathbf{t}, \mathbf{n}, \mathbf{t} \times \mathbf{n})$. It is then easy to see that $X(t) = x(t)$ and $Y(t) = y(t)/\cos\theta$.

The previous equation then becomes:

$$k_t^\theta = \frac{1}{\cos\theta} \lim_{\mathbf{P}} \frac{2Y(t)}{X^2(t)}$$

But we have seen in equation [1.25] that such a limit expression corresponds to the curvature at $\mathbf{P}$ of a curve tangent to $\mathbf{t}$, traced in the plane $\Pi_t$. In other words, this is the normal curvature $k_\mathbf{t}$ and we then get:

$$k_t^\theta = \frac{k_\mathbf{t}}{\cos\theta} \qquad [1.26]$$

This equation is called **Meusnier's theorem**. It is useful to compute the curvature of any inclined section of the surface $\Sigma$ with respect to the curvature of the normal section with the same tangent vector.

### 1.2.8. *Local approximation of the surface*

In a general way, we can approximate the shape of the surface $\Sigma$ around the point $\mathbf{P}$ by the second order expansion:

$$d\mathbf{P} = \frac{\partial \mathbf{P}}{\partial u} du + \frac{\partial \mathbf{P}}{\partial v} dv + \frac{1}{2}\left( \frac{\partial^2 \mathbf{P}}{\partial u^2} du^2 + 2\frac{\partial \mathbf{P}}{\partial u}\frac{\partial \mathbf{P}}{\partial v} du dv + \frac{\partial^2 \mathbf{P}}{\partial v^2} dv^2 \right)$$

If we perform a projection on the normal vector $\mathbf{n}$, we eliminate the tangent vectors $\frac{\partial \mathbf{P}}{\partial u}$ and $\frac{\partial \mathbf{P}}{\partial v}$. We then get an equation of the shape relatively to $z$, which is the height with respect to the tangential plane at $\mathbf{P}$. With the coefficients of the second fundamental form, we can write it as:

$$z(u,v) = d\mathbf{P} \cdot \mathbf{n} = \frac{1}{2}(Ldu^2 + 2Mdudv + Ndv^2)$$

Now, let us assume that the surface $\Sigma$ is parameterized by the lines of curvature. For each point $\mathbf{P}$, we have the local frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$. If we move $\mathbf{P}$ along the isoparametric line $dv = 0$, we can define the $x$ coordinate as:

$$x = d\mathbf{P} \cdot \mathbf{t_1} = \frac{\partial \mathbf{P}}{\partial u} du \cdot \mathbf{t_1} = \sqrt{E}\, du$$

Similarly, we can define $y = \sqrt{G}\, dv$.

We have seen in section 1.2.6 that $M = 0$ in the case of this specific parameterization which gives:

$$z(x,y) = \frac{1}{2}\left(L\frac{x^2}{E} + N\frac{y^2}{G}\right)$$

and from equations [1.23], we infer:

$$z(x,y) = \frac{1}{2}(k_1 x^2 + k_2 y^2)$$

Such an equation allows us to represent the local shape of the surface at point $\mathbf{P}$ at order 2. In particular, if $k_1$ and $k_2$ share the same sign (which is equivalent to $k_g > 0$), the shape is locally an ellipsoid and the point is said to be *elliptical*. If the signs are opposite (i.e. $k_g < 0$), the point is said to be *hyperbolic* and the shape looks locally like a saddle. If one principal curvature is equal to zero (i.e. $k_g = 0$), the point is said to be *parabolic*.

### 1.2.9. *Focal surfaces*

For any point $\mathbf{P}$ of $\Sigma$ where the principal curvatures are not equal to 0, we can define the **centers of principal curvatures** $\mathbf{C_1}(\mathbf{P})$ and $\mathbf{C_2}(\mathbf{P})$ as:

$$\mathbf{C_1}(\mathbf{P}) = \mathbf{P} - \frac{1}{k_1(\mathbf{P})}\mathbf{n}(\mathbf{P}) \quad \mathbf{C_2}(\mathbf{P}) = \mathbf{P} - \frac{1}{k_2(\mathbf{P})}\mathbf{n}(\mathbf{P})$$

By continuity, the set of $C_1(\mathbf{P})$ (resp. $C_2(\mathbf{P})$) defines a surface $E_1$ (resp. $E_2$) which is called the **focal surface** (or **surface of centers** or **evolute surface**) of the principal curvature $k_1$ (resp. $k_2$).

There is a specific relationship between $\Sigma$ and its focal surfaces $E_1$ and $E_2$. The following demonstration is adapted from [YU 07] (see also section 10.4 of [POR 01]).

We can estimate the variation of a point $\mathbf{C_1}$ over $E_1$ when $\mathbf{P}$ moves. For this, we use the principal frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ and we can write:

$$\frac{\partial \mathbf{C_1}}{\partial u} = \frac{\partial \mathbf{P}}{\partial u} - \frac{1}{k_1}\frac{\partial \mathbf{n}}{\partial u} + \frac{1}{k_1^2}\frac{\partial k_1}{\partial u}\,\mathbf{n}$$

We have seen in equation [1.20] that $\dfrac{\partial \mathbf{n}}{\partial u} = k_1 \dfrac{\partial \mathbf{P}}{\partial u}$. Thus:

$$\frac{\partial \mathbf{C_1}}{\partial u} = \frac{1}{k_1^2}\frac{\partial k_1}{\partial u}\,\mathbf{n}$$

$\frac{\partial \mathbf{C_1}}{\partial u}$ defines a tangent vector on $E_1$. Thus, the above equation shows that the normal vector $\mathbf{n}$ is tangent to $E_1$.

Now, we can estimate the variation of a point $\mathbf{C_1}$ over $E_1$ when $\mathbf{P}$ moves along the line of curvature defined by the principal curvature $\mathbf{t_2}$:

$$\frac{\partial \mathbf{C_1}}{\partial v} = \frac{\partial \mathbf{P}}{\partial v} - \frac{1}{k_1}\frac{\partial \mathbf{n}}{\partial v} + \frac{1}{k_1^2}\frac{\partial k_1}{\partial v}\,\mathbf{n}$$

We have seen in equation [1.20] that $\dfrac{\partial \mathbf{n}}{\partial v} = k_2 \dfrac{\partial \mathbf{P}}{\partial v}$. Thus:

$$\frac{\partial \mathbf{C_1}}{\partial v} = \left(1 - \frac{k_2}{k_1}\right)\frac{\partial \mathbf{P}}{\partial v} + \frac{1}{k_1^2}\frac{\partial k_1}{\partial v}\,\mathbf{n}$$

$\frac{\partial \mathbf{C_1}}{\partial v}$ defines a tangent vector on $E_1$. As we have seen that $\mathbf{n}$ is also tangent to $E_1$, it results from the above equation that $\mathbf{t_2}$ is tangent to $E_1$. As $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ forms an orthonormal frame, we can deduce that $\mathbf{t_1}$ is normal to $E_1$.

Similarly, we can show that $\mathbf{n}$ is tangent to $E_2$ and that $\mathbf{t_2}$ is normal to $E_2$. This shows that $E_1$ and $E_2$ can be defined as the envelopes of the normal vectors to $\Sigma$ and that the normal vectors of $E_1$ and $E_2$ correspond to the principal directions of $\Sigma$.

Note that in [HAG 92], the authors propose a generalization of the definition of the focal surfaces. Points of the "generalized focal surface" have the following form:

$$\mathbf{F}(\mathbf{P}) = \mathbf{P} + \alpha f(k_1(\mathbf{P}), k_2(\mathbf{P}))\mathbf{n}(\mathbf{P})$$

with $\alpha$ a real value called the scale factor. Different functions $f$ are proposed (like $f = k_1 k_2 = k_g$ or $f = k_1^2 + k_2^2$). By displaying this generalized focal surface, it is possible to emphasize some characteristics of the surface shape (see section 5.4 of [HAH 08]).

## 1.3. Computation of differential parameters on a discrete 3D mesh

### 1.3.1. *Introduction*

We have seen in the previous section the formulas to compute differential parameters on a continuous surface. In the case of a discrete 3D mesh, the problem becomes much more complex as we lose the analytic definition of the surface. We then need to either introduce an approximation scheme based on sparse measurements, or make some hypotheses in order to fit a continuous representation.

Many methods have been proposed to compute differential parameters on a 3D mesh and we can find quite complete surveys in [MAG 07] or [GAT 06]. In the following sections, for each type of feature, we list the main categories of methods and present with more details one algorithm (very often, one of the first published). Note that sometimes, the original implementation will be simplified for a better comprehension.

### 1.3.2. *Some notations*

We will assume that the 3D mesh $\mathcal{M}$ is given by its set of *vertices* $\mathbf{P_i}(x_i, y_i, z_i)$, the set of *edges* $e_j$ linking two vertices and the set of *facets* $F_k$

composed of edges (3 in the standard case of triangular facets). We will denote the oriented edge linking $\mathbf{P_i}$ and $\mathbf{P_j}$ as $\mathbf{e_i^j} = \mathbf{P_i P_j}$.

For each vertex $\mathbf{P_i}$, we can define its *neighborhood* $N(\mathbf{P_i})$ that contains all the vertices which are connected to $\mathbf{P_i}$. This direct neighborhood is also called the *1-ring* of the vertex $\mathbf{P_i}$. We can then compute the neighborhood of all the vertices belonging to $N(\mathbf{P_i})$ and fuse them which results in a more general neighborhood called the 2-ring of the vertex $\mathbf{P_i}$. Recursively, we can define the $n$-ring of the vertex $\mathbf{P_i}$.

We assume that the 3D mesh $\mathcal{M}$ is orientable, which means that the ordering of vertices in a facet is consistent between two adjacent facets: when two vertices of a common edge are ordered in one direction in a facet, they must be ordered in the opposite direction in the other facet. This allows us to define consistently the two sides of $\mathcal{M}$.

At each vertex $\mathbf{P_i}$, we can define the *normal vector* $\mathbf{n_i}$ to the mesh $\mathcal{M}$. This is the vector which is locally "orthogonal" to the surface. Note that on a discrete mesh, this notion of orthogonality is not straightforward to define at a vertex where many planar faces intersect. When normal vectors are considered on a closed 3D mesh, they are generally defined as outward-pointing (i.e. pointing towards the exterior of the mesh).

### 1.3.3. *Computing normal vectors*

To compute differential parameters, most of the methods require first the computation of the discrete normal vector $\mathbf{n_i}$ at each vertex $\mathbf{P_i}$ of a 3D mesh. This is a critical step as it influences the rest of the process a great deal.

Many algorithms have been proposed and we can find a review of them in [JIN 05]. In the following, we focus on an algorithm where the normal vector is computed by averaging the normal vectors to the adjacent facets weighted by their areas. This is a particular case of the class of "mean weighted methods" (see, in particular, [MAX 99]), which are efficient and easy to implement. Note that we assume that all the triangles of the mesh are oriented consistently.

It is generally accepted that the computed normal vector should point "outside" the mesh. This depends, in fact, on the initial choice of facet orientation, which will define the direction of the cross-product in the

algorithm. In most cases, the process of acquisition and design of the 3D mesh results in a correct orientation over all its facets. Nevertheless, some methods have been proposed (see, for example, [TAK 14]) to correct inconsistencies in facet orientation or in normal direction.

**Input:** Vertex $\mathbf{P_i}$
**Output:** Normal vector $\mathbf{n_i}$
**begin**
  Take all the neighbor vertex $\mathbf{P_j} \in N(\mathbf{P_i})$.
  All these vertices form a set of facets $F_i^j$ which shares the vertex $\mathbf{P_i}$.
  **forall** *facet* $F_i^j$ *(which can be triangular or not)* **do**
    Take the vectors $\mathbf{e}$ and $\mathbf{e}'$ corresponding to the two edges of the facet $F_i^j$ which are incident to $\mathbf{P_i}$
    Define the triangle $(\mathbf{P_i}, \mathbf{e}, \mathbf{e}')$ and compute its area $A_i^j$ and its normal vector $\mathbf{n_i^j}$ by using the cross product: $A_i^j = \frac{1}{2}\|\mathbf{e} \times \mathbf{e}'\|$ and $\mathbf{n_i^j} = (\mathbf{e} \times \mathbf{e}')/2A_i^j$
  **end**
  Compute the normal vector at $\mathbf{P_i}$ by averaging all the normal vectors of the neighborhood weighted by their areas:

$$\mathbf{n_i} = \frac{1}{\Sigma_j A_i^j} \sum_j A_i^j \mathbf{n_i^j}$$

**end**

**Algorithm 1:** Computing the normal vector $\mathbf{n_i}$ at vertex $\mathbf{P_i}$.

Note that the problem is much complex if we have an unstructured 3D point cloud instead of a 3D mesh. As we have no connection information anymore, we define the neighborhood of $\mathbf{P_i}$ as the set of points $N_\sigma(\mathbf{P_i})$, which are at a distance less than a given threshold $\sigma$. We can estimate the plane which best fits the set of points $N_\sigma(\mathbf{P_i})$ using the least-square criterion. The normal vector at point $\mathbf{P_i}$ is then given by a normalized vector orthogonal to this plane. Note that this normal vector is non-oriented and that some post-processing is required to associate a consistent direction (see the above paragraph). A major concern is that $\sigma$ has a large influence on the result quality and we can find in [MIT 04] a method to find the optimal neighborhood size using local information.

### 1.3.4. *Locally fitting a parametric surface*

In order to compute the differential parameters, a first idea is to fit locally a continuous parametric surface $f(u, v)$ on the 3D mesh around $\mathbf{P_i}$, which allows us to use equations of section 1.2.5. One of the first algorithms was proposed in [HAM 93] (in fact, we can find a similar method in an older paper [STO 92] but it was applied to depth images). The algorithm aims at fitting a quadratic surface around each vertex in order to get a formal computation of the differential parameters (see Algorithm 2).

In [PET 02], it is emphasized that this class of methods relies heavily on the accuracy of the surface normal $\mathbf{n_i}$. The authors propose then a more sophisticated method based on an extended quadric surface, which allows us to refine iteratively the computation of the normal vector.

In [CAZ 05a], the authors propose to use polynomial fitting with an order higher than 2. By increasing the order, they can compute derivatives of the differential parameters which can be used to define some feature lines. This method has been made available in the open-source C++ Computational Geometry Algorithms Library CGAL[2] (see also [CAZ 08b]).

### 1.3.5. *Discrete differential geometry operators*

The objective of *discrete differential geometry* (DDG) is to define geometry differential operators directly from measurements performed on the discrete elements of the 3D mesh (as the area of a facet or the angle between two vertices), in other words, without any approximation by a continuous function (see [CRA 13] for more details).

In the following, we illustrate the classical formulas of DDG by presenting some approximations. The idea is to model the local shape of the 3D mesh (which is assumed to be triangulated) at vertex $\mathbf{P_i}$ by a simple geometric primitive, which allows the computation of either the Gaussian (based on a sphere [MES 07]) or the mean curvature (based on cylinders [MES 12, DYN 01]).

2 https://doc.cgal.org/latest/Jet_fitting_3/classCGAL_1_1Monge__via__jet__fitting.html.

**Input:** Vertex $\mathbf{P_i}$, normal $\mathbf{n_i}$
**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ and the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$
**begin**
Define the plane $\Pi$ passing through $\mathbf{P_i}$ and orthogonal to $\mathbf{n_i}$. This corresponds to the tangent plane to the 3D mesh at $\mathbf{P_i}$.
Build an orthogonal coordinate system centered in $\mathbf{P_i}$ composed of the normal $\mathbf{n_i}$ and two orthogonal vectors $\mathbf{i'}$ and $\mathbf{j'}$ taken at random in $\Pi$. For example, we can take in the canonical frame $(\mathbf{i}, \mathbf{j}, \mathbf{k})$, after normalization, $\mathbf{i'} = \mathbf{n_i} \times \mathbf{i}$ and $\mathbf{t_2} = \mathbf{n_i} \times \mathbf{t_1}$.
**forall** *neighbor vertex* $\mathbf{P_j} \in N(\mathbf{P_i})$ **do**
Compute the coordinates $(u_j, v_j, w_j)$ of $\mathbf{P_j}$ in this system. $w_j$ corresponds then to the distance between $\mathbf{P_j}$ and the plane $\Pi$.
**end**
Approximate locally the surface of the 3D mesh by using a bivariate polynomial of order 2 (which corresponds also to the Taylor expansion of order 2 around $\mathbf{P_i}$):

$$w(u, v) = au^2 + 2buv + cv^2$$

We want that $w(0, 0) = 0$ as the origin of the coordinate system is $\mathbf{P_i}$ and that $w(u_j, v_j)$ is as close as possible to $w_j$. If we introduce the squared distance:

$$E = \sum_{N(\mathbf{P_i})} [(au_j^2 + 2bu_jv_j + cv_j^2) - w_j]^2$$

the objective is to minimize $E$ in order to find the values $a$, $b$ and $c$. This can be done by least-square minimization (see for example section 15.1 of [PRE 07]).
We have now a surface around $\mathbf{P_i}$ given by $\mathbf{P}(u, v) = (u, v, w(u, v))$ and by using equations [1.21] and [1.11], we can easily compute the principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ and the principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$.
**end**

**Algorithm 2:** Computing differential parameters by locally fitting a continuous parametric function.

## Gaussian curvature

Let us define a cone $\Gamma$ defined by its apex $\mathbf{P}$, its angle $\alpha$ and its slant height $l$ (see Figure 1.3, left). The radius of the cone base is then $R = l \sin \alpha$ which gives a perimeter of $2\pi l \sin \alpha$. If we cut $\Gamma$ along a generatrix, unfold the cone surface and develop it on a plane, we have a sector of disk of radius $l$ and of angle $\theta$ (see Figure 1.3, right). Note that $2\pi - \theta$ is called the *deficit angle* or the *defect angle* at $\mathbf{P}$.

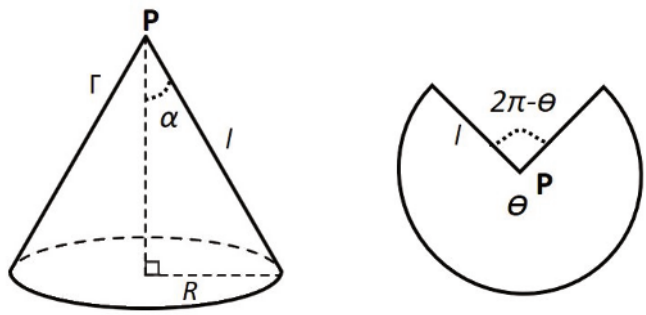



**Figure 1.3.** *Left: the cone $\Gamma$ is defined by its apex $\mathbf{P}$, its angle $\alpha$ and its slant height $l$. Right: the cone is now developed on a plane and becomes a sector of disk of angle $\theta$*

The perimeter of the disk sector is $\theta l$ and by construction, it is equal to the perimeter of the cone base which leads to $2\pi l \sin \alpha = \theta l$ or $\sin \alpha = \frac{\theta}{2\pi}$.

Now, let us introduce a sphere of radius $r$ into the cone, it stabilizes along a tangent circle of radius $R$. Let $A$ be the area of the spherical cap $\Sigma$ which is above this tangent circle. If $h$ is the height of the spherical cap (see Figure 1.4, left), we get $A = 2\pi r h$ or $h = A/2\pi r$. By using the relationships in a squared triangle, we get $h = r(1 - \sin \alpha) = r(1 - \theta/2\pi)$. Based on the two previous expressions, we infer that $1/r^2 = (1/A)(2\pi - \theta)$.

If we approximate the Gaussian curvature at $\mathbf{P}$ by the Gaussian curvature of the sphere which is equal to $1/r^2$, we get:

$$k_g(\mathbf{P}) = \frac{1}{A}(2\pi - \theta)$$

In a discrete 3D mesh, the set of facets $F_i^j$ which share the vertex $\mathbf{P_i}$ forms a generalized cone (with a regular shape if the mesh is locally convex or concave). By using the Gauss–Bonnet theorem, we can demonstrate (see, for example, [MEY 03], [ALB 05] or [MEE 00]) a similar discrete formula:

$$k_g(\mathbf{P_i}) = \frac{1}{A_i}(2\pi - \sum_j \theta_i^j)$$

where $\theta_i^j$ is the angle of the facet $F_i^j$ at the vertex $\mathbf{P_i}$ and $A_i$ is an "area" defined around the vertex $\mathbf{P_i}$ (see Figure 1.4, right).

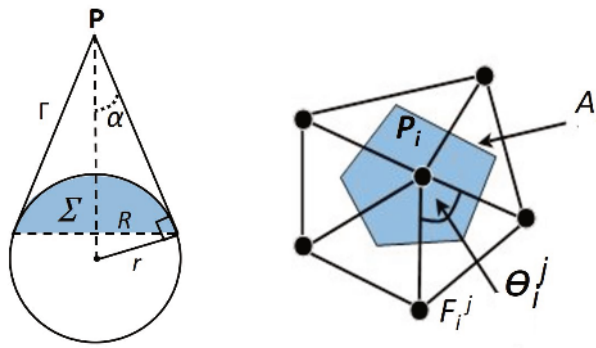



**Figure 1.4.** *Left: if we introduce a sphere of radius $r$ into the cone, it stabilizes along a tangent circle of radius $R$; the spherical cap $\Sigma$ of height $h$ and area $A$ will be above this circle. Right: the discrete Gaussian curvature at vertex $\mathbf{P_i}$ is based on the angles of the adjacent facets $F_i^j$ at $\mathbf{P_i}$ which defines the discrete "deficit angle" and on the "area" defined around $\mathbf{P_i}$*

As emphasized in [DYN 01], there are, in fact, different ways of defining the area $A_i$, which result in different curvature values. The most commonly used area definitions are:

– *barycentric area* which considers that a facet is equally shared by three vertices. We then get $A_i = \frac{1}{3}\sum_j A_i^j$ where $A_i^j$ is the area of facet $F_i^j$;

– *mixed area* (sometimes also called *Voronoï area*) defined in [MEY 03]. In this case, for each facet $F_i^j$, we perform a Voronoï tessellation based on the three vertices and we take into account only the area of the Voronoï cell including $\mathbf{P_i}$.

We can find a discussion about these two definitions in [XU 06].

## Mean curvature

Suppose that we replace each edge $\mathbf{e_i^j}$ between $\mathbf{P_i}$ and $\mathbf{P_j}$ with a small cylinder portion $\Sigma_i^j$ of radius $r_i^j$ that joins the adjacent faces tangentially (see Figure 1.5, left). We have seen in section 1.2.5 that for a cylinder of radius $r$, we have one principal curvature (along a generatrix) equal to 0 and the other (along a circle) equal to $1/r$. So for each point of $\Sigma_i^j$, we have $k_1 = 1/r_i^j$, $k_2 = 0$ and the mean curvature is $k_m^j = 1/2r_i^j$.

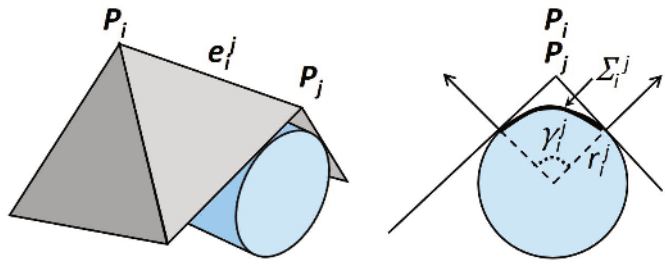



**Figure 1.5.** *The edge $\mathbf{e_i^j}$ between the vertex $\mathbf{P_i}$ and a neighbor vertex $\mathbf{P_j}$ is approximated by a tangent cylinder portion $\Sigma_i^j$ of radius $r_i^j$*

Let us define $\gamma_i^j$ the angle between the normal vectors of the two adjacent facets (see Figure 1.5, right). The area of the cylinder portion $\Sigma_i^j$ is equal to $A_i^j = \gamma_i^j \; r_i^j \; ||\mathbf{e_i^j}||$. This leads to $k_1 = \gamma_i^j ||\mathbf{e_i^j}||/A_i^j$ and $k_m^j = \gamma_i^j ||\mathbf{e_i^j}||/2A_i^j$.

Now, the idea is to approximate the mean curvature of $\mathbf{P_i}$ by averaging the mean curvature of each $\Sigma_i^j$ weighted by their areas. As each area concerns two facets, we have to divide the result by 2. We then get:

$$k_m(\mathbf{P_i}) = \frac{1}{A_i} \sum_j \frac{1}{2} A_i^j k_m^j = \frac{1}{A_i} \sum_j \frac{1}{2} A_i^j \frac{\gamma_i^j ||\mathbf{e_i^j}||}{2A_i^j} = \frac{1}{4A_i} \sum_j \gamma_i^j \; ||\mathbf{e_i^j}||$$

where $A_i$ is the area around $\mathbf{P_i}$. As for the Gaussian curvature, it is proposed to take for $A$ the aforementioned barycentric area or mixed area.

Another discrete method to compute the mean curvature was proposed in [MEY 03]. It is based on the discrete Laplace–Beltrami operator applied to the

mean curvature flow and results also in a weighted sum of the length of the incident edges at $\mathbf{P_i}$, but with different angles:

$$k_m(\mathbf{P_i}) = \frac{1}{2A}\sum_j (\cot\alpha_i^j + \cot\beta_i^j)||\mathbf{e_i^j}||$$

where $\alpha_i^j$ and $\beta_i^j$ are the measurements of the angles opposite to the edge $\mathbf{e_i^j}$ and $A$ is the *mixed area*. A discussion between the two formulas can be found in [MES 12, SIM 13].

In its simplest form, the algorithm to compute Discrete Differential Geometry operators is then:

**Input:** Vertex $\mathbf{P_i}$
**Output:** Gaussian and mean curvatures $k_g$ and $k_m$, principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ with the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$
**begin**
  **forall** *facet* $F_i^j$ *(which are assumed triangular) of* $\mathcal{F}_i$ **do**
    Compute the area $A_i^j$.
    Compute the angle $\theta_i^j$ at $\mathbf{P_i}$.
  **end**
  **forall** *edge* $\mathbf{e_i^j}$ **do**
    Compute the angle $\gamma_i^j$ between the two adjacent facets.
  **end**
  Compute $A_i = \frac{1}{3}\sum_j A_i^j$
  Compute:

$$k_g(\mathbf{P_i}) = \frac{1}{A_i}(2\pi - \sum_j \theta_i^j)$$

$$k_m(\mathbf{P_i}) = \frac{1}{4A_i}\sum_j \gamma_i^j \, ||\mathbf{e_i^j}||$$

  Compute $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ by solving the quadatric equation given by the relationships $k_1 k_2 = k_g$ and $(k_1 + k_2)/2 = k_m$.
  Compute the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$ by the least-square fitting method presented in [MEY 03], section 5.3.
**end**

**Algorithm 3:** Computing differential parameters by using discrete differential geometry operators.

### 1.3.6. *Integrating 2D curvatures*

One of the first algorithms to compute differential parameters on a 3D mesh was described in [CHE 92]. It is based on the computation of the radii of circles going through the considered vertex and two neighbor vertices. For each circle, the inverse of the radius gives an estimation of the curvature in a tangent direction thanks to Meusnier's theorem. Based on the estimation for several circles, it becomes possible to estimate principal curvatures thanks to Euler's theorem. A version of the algorithm is given below (see Algorithm 4).

Note that, in the original method, the normal $\mathbf{n_i}$ is computed based on the vectors $\mathbf{n_{jk}}$.

In [WAT 01] and [DON 05], we find other methods which are also based on Euler's theorem. In both cases, approximation of the normal curvatures is based on only one neighbor $\mathbf{P_j}$, but it involves the normal vector $\mathbf{n_j}$.

### 1.3.7. *Tensor of curvature: Taubin's formula*

In this method, the idea is to compute the tensor of curvature, which is the map that associates with each point $\mathbf{P}$ of a surface $\Sigma$, a $3 \times 3$ matrix that measures the directional curvature $k_\mathbf{t}$ for any tangent vector $\mathbf{t}$. In [TAU 95], the author proposed to use the following formula:

$$M(\mathbf{P}) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} k_\mathbf{t}(\theta)\, \mathbf{t}(\theta) \cdot \mathbf{t}^t(\theta) d\theta \qquad [1.27]$$

with the notations $k_\mathbf{t}(\theta)$ and $\mathbf{t}(\theta)$ introduced in section 1.2.6.

If we write the above equation in the frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ defined by the principal directions and the normal at $\mathbf{P}$ and if we use Euler's theorem, we get:

$$M(\mathbf{P}) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} (k_1 \cos^2\theta + k_2 \sin^2\theta) \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \end{pmatrix} d\theta$$

**Input:** Vertex $\mathbf{P_i}$, normal $\mathbf{n_i}$
**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ and the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$
**begin**
  **forall** *couples of neighbor vertices* $(\mathbf{P_j}, \mathbf{P_k}) \in N(\mathbf{P_i})$ **do**
    Compute the "oppositeness" value $M = \mathbf{P_j P_i}.\mathbf{P_i P_k}$. The greater is $M$, the more opposite are $\mathbf{P_j}$ and $\mathbf{P_k}$ with respect to $\mathbf{P_i}$.
    Sort the couples in the decreasing order of oppositeness and keep the best couples in a list $\Lambda$.
    **foreach** *couple* $(\mathbf{P_j}, \mathbf{P_k})$ *of* $\Lambda$ **do**
      Compute the circle $\mathcal{C}_{jk}$ passing through $(\mathbf{P_j}, \mathbf{P_i}, \mathbf{P_k})$
      Compute its center $\mathbf{C_{jk}}$.
      $\mathbf{n_{jk}} = \mathbf{C_{jk}P_i} / \|\mathbf{C_{jk}P_i}\|$ is the normal vector to $\mathcal{C}_{jk}$ at $\mathbf{P_i}$.
      $k_{jk} = 1 / \|\mathbf{C_{jk}P_i}\|$ is the curvature of $\mathcal{C}_{jk}$ at $\mathbf{P_i}$
      Compute $\mathbf{t_{jk}} = \mathbf{n_i} \times \mathbf{n_{jk}}$. By definition, $\mathbf{t_{jk}}$ is a vector orthogonal to $\mathbf{n_i}$ and is then a tangent vector.
      Let us define $\Pi_{jk}$ the normal plane at $\mathbf{P_i}$ in the direction of the tangent vector $\mathbf{t_{jk}}$. The plane containing $\mathcal{C}_{jk}$ can be considered as a rotation of $\Pi_{jk}$ around the axis $(\mathbf{P}, \mathbf{t_{jk}})$ by an angle $\alpha_{jk}$ given by $\cos\alpha_{jk} = \mathbf{n_i} \cdot \mathbf{n_{jk}}$
      We can then apply Meusnier's theorem (see equation [1.26]) to get an approximation of the normal curvature in the direction $\mathbf{t_{jk}}$: $k_{\mathbf{t}}^{jk} = k_{jk} \cos\alpha_{jk}$
    **end**
    Now, we can use Euler's theorem to estimate the principal curvatures. We have (see equation [1.24]):
    $k_1(\mathbf{P_i})\cos^2\theta_{jk} + k_2(\mathbf{P_i})\sin^2\theta_{jk} = k_{\mathbf{t}}^{jk}$ where $\theta_{jk} = \angle(\mathbf{t_1}, \mathbf{t}_{jk})$
    In any other tangent frame shifted by an angle $\theta_0$ with respect to $\mathbf{t_1}$, we can write:
    $k_1(\mathbf{P_i})\cos^2(\theta_{jk} - \theta_0) + k_2(\mathbf{P_i})\sin^2(\theta_{jk} - \theta_0) = k_{\mathbf{t}}^{jk}$
    If we develop, we obtain:
    $a\cos^2\theta_{jk} + b\cos\theta_{jk}\ \sin\theta_{jk} + c\sin^2\theta_{jk} = k_{\mathbf{t}}^{jk}$
    If we have $n$ couples $(\mathbf{P_j}, \mathbf{P_k})$, we get $n$ equations. By using a least square method, we can compute $a$, $b$ and $c$ and then $\theta_0$. It gives directly the frame of the principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$ and the principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$.
  **end**
**end**

**Algorithm 4:** Computing differential parameters by integrating 2D curvatures.

which results in:

$$M(\mathbf{P}) = \begin{pmatrix} \frac{3}{8}k_1 + \frac{1}{8}k_2 & 0 & 0 \\ 0 & \frac{1}{8}k_1 + \frac{3}{8}k_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

In this diagonal matrix, we have two non-zero eigenvalues $\lambda_1 = \frac{3}{8}k_1 + \frac{1}{8}k_2$ and $\lambda_2 = \frac{1}{8}k_1 + \frac{3}{8}k_2$, which are associated with the eigenvectors $\mathbf{t_1}$ and $\mathbf{t_2}$. We then get:

$$k_1 = 3\lambda_1 - \lambda_2 \qquad k_2 = 3\lambda_2 - \lambda_1$$

with the normal vector $\mathbf{n}$ being the third eigenvector associated with the eigenvalue 0.

Taubin then proposed to approximate, for any vertex $\mathbf{P_i}$ of a 3D mesh, the matrix $M(\mathbf{P_i})$. For this, he discretized the circular area around $\mathbf{P_i}$ by a polygon linking the neighbor vertices $\mathbf{P_j}$ of $\mathbf{P_i}$. This leads to a discrete formulation of equation [1.27], which allows us to compute the differential parameters by the following algorithm (see Algorithm 5).

### 1.3.8. *Tensor of curvature based on the normal cycle theory*

In [COH 03], the authors proposed another definition of the curvature tensor, which has theoretical foundations as well as some convergence properties.

For a vertex $\mathbf{P_i}$ of the 3D mesh that we assume triangular, we can take all the vertices $\mathbf{P_j}$ of the neighborhood $N(\mathbf{P_i})$. This defines the set of connecting edges $\mathbf{e_i^j}$, which are defined by their unit vectors $\mathbf{t_i^j}$ (so with the same direction as $\mathbf{e_i^j}$). For an edge $\mathbf{e_i^j}$, we can define the following $3 \times 3$ matrix $M(\mathbf{e_i^j})$ as:

$$M(\mathbf{e_i^j}) = \frac{1}{B_i^j} ||\mathbf{e_i^j}|| \gamma_i^j \, \mathbf{t_i^j} \cdot \mathbf{t_i^{j^t}}$$

where $B_i^j$ is the half area of the two adjacent triangles incident to the edge $\mathbf{e_i^j}$ and $\gamma_i^j$ is the angle between the normal vectors to these triangles. The sign of $\gamma_i^j$ is chosen to be positive if the two triangles form a convex dihedron and negative if it is concave.

**Input:** Vertex $\mathbf{P_i}$, normal vector $\mathbf{n_i}$
**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ with the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$.
**begin**
  **forall** *neighbor vertex* $\mathbf{P_j} \in N(P_i)$ **do**
    Compute the unit tangent vector $\mathbf{t_i^j}$ at $\mathbf{P_i}$ in the direction of $\mathbf{P_j}$. For this, project and normalize the vector $\mathbf{P_iP_j}$ on the tangent plane at $\mathbf{P_i}$ which is defined by $\mathbf{n_i}$.
    If we work in the normal plane at $\mathbf{P_i}$ containing $\mathbf{P_j}$, we can write the coordinates of $\mathbf{P_j}$ as $y = \mathbf{n_i}(\mathbf{P_j} - \mathbf{P_i})$ and $x = \mathbf{t_i^j}(\mathbf{P_j} - \mathbf{P_i})$. By applying an approximation of equation [1.25], we can write:

$$k(\mathbf{P_j}) = \frac{2\ \mathbf{n_i}(\mathbf{P_j} - \mathbf{P_i})}{||\mathbf{t_i^j}(\mathbf{P_j} - \mathbf{P_i})||^2}$$

    Note that in Taubin's paper, the approximation is slightly different as it is based on a circle interpolation.
    Compute the area $a_i^j$ of the two adjacent facets.
  **end**
  The discrete formulation of the tensor of curvature at $\mathbf{P_i}$ is:

$$M(\mathbf{P_i}) = \frac{1}{\sum_j a_i^j} \sum_{\mathbf{P_j}} a_i^j\, k(\mathbf{P_j})\ \mathbf{t_i^j} \cdot \mathbf{t_i^{jt}}$$

  Extract the non-null eigenvectors with the corresponding eigenvalues $\lambda_1$, $\lambda_2$ of the $3 \times 3$ tensor $M(\mathbf{P_i})$ (see [TAU 95] for details on the numerical algorithm).
  The principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$ are inferred from the eigenvectors and the principal curvatures are given by:
  $k_1(\mathbf{P_i}) = 3\lambda_1 - \lambda_2$
  $k_2(\mathbf{P_i}) = 3\lambda_2 - \lambda_1$
**end**

**Algorithm 5:** Computing differential parameters by using Taubin's tensor of curvature.

By taking the approximation of the curvature of the edge $\mathbf{e_i^j}$ by a cylinder portion (see section 1.3.5 and Figure 1.5), we have $k_1 = \gamma_i^j ||\mathbf{e_i^j}||/A_i^j$ and $k_2 = 0$ where $A_i^j$ is the area of the cylinder portion. The principal directions

corresponding to $k_1$ and $k_2$ are respectively given by a vector $\mathbf{v_i^j}$ orthogonal to $\mathbf{t_i^j}$ and tangent to the cylinder and by $\mathbf{t_i^j}$.

Now, if we write $M(\mathbf{e_i^j})$ in the orthonormal frame $(\mathbf{t_i^j}, \mathbf{v_i^j}, \mathbf{n_i^j})$ where $\mathbf{n_i^j} = \mathbf{t_i^j} \times \mathbf{v_i^j}$, we get:

$$M(\mathbf{e_i^j}) = \frac{1}{B_i^j} ||\mathbf{e_i^j}|| \gamma_i^j \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

If we assimilate $B_i^k$ and $A_i^k$, it results in:

$$M(\mathbf{e_i^j}) = \begin{pmatrix} \frac{A_i^j}{B_i^j} k_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \approx \begin{pmatrix} k_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

As the matrix is written under a diagonal form, this means that we have an eigenvalue equal to $k_1$, associated with the eigenvector $\mathbf{t_i^j}$ (which is the principal direction corresponding to $k_2$), a null eigenvalue which can be related to $k_2$, associated with $\mathbf{v_i^j}$ (which is the principal direction corresponding to $k_1$) and another null eigenvalue associated with $\mathbf{n_i^j}$.

This means that the principal curvatures $(k_1, k_2)$ and their associated principal directions can be defined as eigenvalues and eigenvectors of $M(\mathbf{e_i^j})$. Note that the principal directions are switched with respect to the principal curvatures.

Now, the idea is to integrate all tensors $M(\mathbf{e_i^j})$ in a unique tensor centered in $\mathbf{P_i}$:

$$M(\mathbf{P_i}) = \frac{1}{B_i} \sum_{\mathbf{P_j} \in N(\mathbf{P_i})} ||\mathbf{e_i^j}||\ \gamma_i^j\ \mathbf{t_i^j} \cdot \mathbf{t_i^j}^t$$

where $B_i = \sum_j B_i^j$ if the area of the sub-mesh formed by all the vertices belonging to $N(\mathbf{P_i})$.

If we generalize the formulas obtained with $M(\mathbf{e_i^j})$, we may assume that we can compute the differential parameters at vertex $\mathbf{P_i}$ based on the

eigenvalues and eigenvectors of $M(\mathbf{P_i})$. More precisely, there will always be a null eigenvalue corresponding to the normal vector $\mathbf{n_i}$, the non-null eigenvalues will correspond to the principal curvatures and the associated eigenvectors will be the principal directions. In fact, all these results can be formally demonstrated by using a mathematical theory called *normal cycle* [COH 03], which provides a unified way to define curvature in both smooth and polyhedral surfaces.

A simplified version of the method can be implemented by the following algorithm:

**Input:** Vertex $\mathbf{P_i}$, normal vector $\mathbf{n_i}$

**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ with the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$.

**begin**

  **forall** *neighbor vertex* $\mathbf{P_i^j} \in N(P_i)$ **do**

    Compute the unit tangent vector $\mathbf{t_i^j}$ at $\mathbf{P_i}$ in the direction of $\mathbf{P_j}$. For this, project and normalize the vector $\mathbf{P_iP_j}$ on the tangent plane at $\mathbf{P_i}$ defined by $\mathbf{n_i}$.

    Compute the half-area $a_i^j$ of the two adjacent facets.

  **end**

  The discrete formulation of the tensor of curvature at $\mathbf{P_i}$ is:

$$M(\mathbf{P_i}) = \frac{1}{\sum_j a_i^j} \sum_{\mathbf{P_j} \in N(P_i)} ||\mathbf{e_i^j}||\ \gamma_i^j\ \mathbf{t_i^j} \cdot \mathbf{t_i^j}^t$$

  Extract the two eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$ corresponding to the two maximum eigenvalues $\lambda_1$ and $\lambda_2$.

  The couples $(\lambda_2, \mathbf{v_1})$ and $(\lambda_1, \mathbf{v_2})$ correspond to the principal directions and curvatures $(k_1(\mathbf{P_i}), t_1(\mathbf{P_i}))$ and $(k_2(\mathbf{P_i}), t_2(\mathbf{P_i}))$ (beware that the eigenvectors are switched with respect to the eigenvalues).

**end**

**Algorithm 6:** Computing differential parameters by using the normal cycle-based tensor of curvature.

A demonstration application of this method, with the C++ source, is available on the Web[3].

3 http://www-sop.inria.fr/members/Pierre.Alliez/demos/curvature/.

### 1.3.9. *Integral estimators*

At a point $\mathbf{P}$ of a closed surface $\Sigma$, let us define the ball $B(\mathbf{P}, r)$ of radius $r$. If we assume that, at any point, the normal $\mathbf{n}$ will be outward-pointing (i.e. pointing towards the exterior of the volume enclosed by the surface), we can define $B^+_\Sigma(\mathbf{P}, r)$ as the sub-volume of $B(\mathbf{P}, r)$, which is above the surface $\Sigma$ (i.e. on the side containing the normal vector $\mathbf{n}$) (see Figure 1.6).

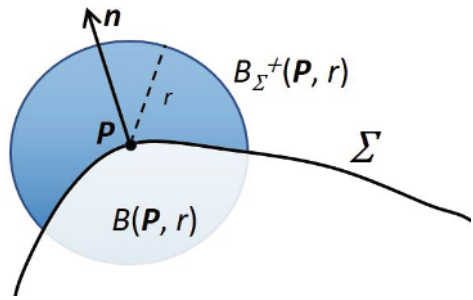



**Figure 1.6.** *$B(\mathbf{P}, r)$ is the ball centered at $\mathbf{P}$ of radius $r$. $B^+_\Sigma(\mathbf{P}, r)$ (in dark blue) is the sub-volume of $B(\mathbf{P}, r)$ which is above the surface $\Sigma$, i.e. on the side containing the normal vector $\mathbf{n}$ which points outside the surface*

In [POT 07], the authors propose to define the differential parameters with respect to the volume integral of some functions. For this purpose, the authors work in the frame $(\mathbf{P}, \mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$, from which we can write the quadratic approximation of the surface $\Sigma$ around $\mathbf{P}$ (see section 1.2.8):

$$z(x, y) = \frac{1}{2}(k_1 x^2 + k_2 y^2)$$

For any function $f$, its integral value $I(f, r)$ over the sub-volume $B^+_\Sigma(\mathbf{P}, r)$ can be computed as (see Figure 1.7):

$$I(f, r) = \int_{B(\mathbf{P},r)^+} f(x, y, z)\ \ dx\, dy\, dz - I_1(f, r) + I_2(f, r)$$

where:

– $B(\mathbf{P}, r)^+$ is the half-ball above the tangent plane;

– $I_1$ is the integral of $f$ in the volume between the tangent plane and the surface $\Sigma$. Note that the sign minus before $I_1$ is due to the sign of $z$, which is directly related to the orientation of $\mathbf{n}$. For example, in Figure 1.7, $z$ and then $I_1$ will be negative;

– $I_2$ is the small volume between the tangent plane and the surface $\Sigma$, which does not belong to $B(\mathbf{P}, r)^+$, so that it is outside the sphere $x^2 + y^2 + z^2 = r^2$ but inside the cylinder $x^2 + y^2 = r^2$. The plus sign before $I_2$ is also related to the predefined orientation of $\mathbf{n}$.

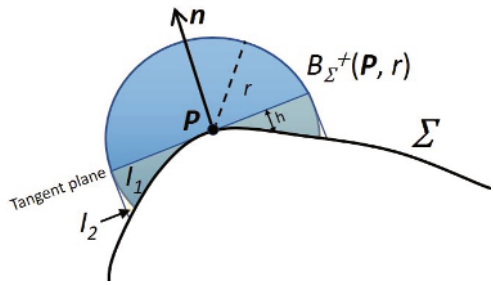



**Figure 1.7.** *The volume $B^+_\Sigma(\mathbf{P}, r)$ is the sum of the half ball which is above the tangent plane (in dark blue), of the volume $I_1$ between the tangent plane and the surface $\Sigma$ minus the small volume $I_2$ between the tangent plane and the surface $\Sigma$ which does not belong to $B(\mathbf{P}, r)^+$ that is outside the sphere $x^2+y^2+z^2 = r^2$ but inside the cylinder $x^2 + y^2 = r^2$. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

This formula is valid whatever the point $\mathbf{P}$ is, elliptical, hyperbolic or parabolic. We can show that $I_2$ is negligible in the computation of $I(f, r)$ and we can approximate $I_1$ as:

$$I_1(f,r) \approx \int_{x^2+y^2\leq r^2} \left( \int_{z=0}^{\frac{1}{2}(k_1x^2+k_2y^2)} f(x,y,z)\, dz \right) dx\, dy$$

This leads to the approximation of the integral estimator $I(f, r)$:

$$I(f,r) \approx \int_{B(\mathbf{P},r)^+} f(x,y,z)\ \ dx\, dy\, dz - \int_{x^2+y^2\leq r^2} \left( \int_{z=0}^{\frac{1}{2}(k_1x^2+k_2y^2)} f(x,y,z)\, dz \right) dx\, dy \quad [1.28]$$

We can use this formula to measure approximately the volume $V(\mathbf{P}, r)$ of $B_{\Sigma}^{+}(\mathbf{P}, r)$ by using the function $f(x, y, z) = 1$. A conversion to polar coordinates leads to (with $k_m = 1/2(k_1 + k_2)$):

$$V(\mathbf{P}, r) = \frac{2\pi}{3} r^3 - \frac{\pi}{4} k_m r^4$$

We can also get an approximation of the coordinates of the barycenter $\mathbf{G}$ of the sub-volume $B_{\Sigma}^{+}(\mathbf{P})$ by using the functions $f(x, y, z) = x$, $y$ or $z$ and we get:

$$\mathbf{G} \left(0, 0, \frac{3}{8} r + \frac{9}{64} k_m r^2\right)$$

We can now define the covariance matrix:

$$J(\mathbf{P}, r) = \int_{B_{\Sigma}^{+}(\mathbf{P}, r)} (\mathbf{X} - \mathbf{G})(\mathbf{X} - \mathbf{G})^t \, d\mathbf{X}$$

As there is symmetry with respect to the planes $(\mathbf{G}, x, z)$ and $(\mathbf{G}, y, z)$, the non-diagonal terms are null and we can simplify the expression in:

$$J(\mathbf{P}, r) = \int_{B_{\Sigma}^{+}(\mathbf{P}, r)} \mathbf{X}\mathbf{X}^t \, d\mathbf{X} - V(\mathbf{P}, r)\mathbf{G}\mathbf{G}^t$$

that we can approximate by:

$$\begin{aligned} J(\mathbf{P}, r) \approx & \, diag\left(I(x^2, r), I(y^2, r), I(z^2, r)\right) \\ & - diag\left(0, 0, \frac{\pi}{4} r^4 \left(\frac{3}{8} r + \frac{9}{64} k_m r^2\right)\right) \end{aligned}$$

By converting in polar coordinates, we can easily compute $I(x^2, r)$, $I(y^2, r)$ and $I(z^2, r)$. Then by developing the expression of $J(\mathbf{P}, r)$, we find the three diagonal terms. The first two correspond to the two eigenvalues associated with the eigenvectors which are in the plane $(\mathbf{B}, x, y)$. They are given by:

$$m_1 = \frac{2\pi}{15} r^5 - \frac{\pi}{48} (3k_1 + k_2) r^6$$

$$m_2 = \frac{2\pi}{15} r^5 - \frac{\pi}{48} (k_1 + 3k_2) r^6$$

These formulas then allow us to compute the principal curvatures $k_1$ and $k_2$ by using the following algorithm:

**Input:** Vertex $\mathbf{P_i}$, radius $r$
**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$
**begin**
Transform the 3D mesh into a binary 3D image by computing an occupancy voxel grid via a scan conversion algorithm (for fundamentals of such algorithm, see the seminal paper of [KAU 87]). Voxel value is 1 if it is inside the surface and 0 if outside.
The voxels belonging to $B_\Sigma^+(\mathbf{P_i}, r)$ are those which are labeled 1 and whose distance of their center to the vertex $\mathbf{P_i}$ is inferior to $r$.
Compute $J(\mathbf{P_i}, r)$ based on the voxels of $B_\Sigma^+(\mathbf{P_i}, r)$. This is equivalent to perform a Principal Component Analysis of the centers of these voxels.
This step can be transformed into a convolution expression, which means than Fast Fourier Transform can be employed for efficiency [POT 07].
Note that in [POT 09], it is proposed to use an octree structure instead of a binary image which allows us to perform computations based on the exact coordinates of the mesh vertices and not on the centers of discrete voxels.
Extract the two eigenvalues $m_1$ and $m_2$ of $J(\mathbf{P_i}, r)$ which correspond to the two eigenvectors close to the tangent surface (i.e. orthogonal to $\mathbf{n}$).
$k_1(\mathbf{P_i}) = \frac{6}{\pi r^6}(m_2 - 3m_1) + \frac{8}{5r} \quad k_2(\mathbf{P_i}) = \frac{6}{\pi r^6}(m_1 - 3m_2) + \frac{8}{5r}$
The principal directions can be estimated by projecting the two corresponding eigenvectors on the tangent plane. Notice that they are not necessarily orthogonal.
**end**

**Algorithm 7:** Computing differential parameters by local integral estimators.

An implementation of another algorithm [COE 14], which is also based on integral invariants can be found in the collaborative project DGtal[4] developed as an open-source C++ library.

4 http://dgtal.org/doc/0.9.2/moduleIntegralInvariant.html.

### 1.3.10. *Processing unstructured 3D point clouds*

If we now have only an unstructured 3D cloud of points $\mathbf{P_i}$ without any edge or facet as in a mesh, how can we compute differential parameters?

As we have no more connection information between the points $\mathbf{P_i}$, the definition of the normal vector $\mathbf{n_i}$ or of the point neighborhood $N(\mathbf{P_i})$ is no more valid, and the methods based on local fitting, integration of 2D curvatures or tensor of curvatures (see respectively sections 1.3.4, 1.3.6, 1.3.7 and 1.3.8) cannot be used. Moreover, we have no facet so the method based on discrete differential geometry operators (see section 1.3.5) becomes inapplicable. At last, we have no mesh surface to define outside and inside which prevents using a method based on integral estimators (see section 1.3.9).

Of course, we could reconstruct a 3D mesh based on the points $\mathbf{P_i}$ (for a review of reconstruction methods, see [BER 14]) and then apply one of the algorithms seen in the previous sections. But some specific methods have been developed to compute differential parameters in the case of an unstructured 3D point cloud. In Algorithm 8, we describe the algorithm proposed in [KAL 09] (see also some details in [KAL 07b]).

Note that this method can also be used for 3D meshes [KAL 07b]. An implementation is publicly available as a Windows executable[5].

We can also find in the Point Cloud Library[6] an implementation of another method to estimate principal directions and curvatures of a 3D point cloud based on the principal component analysis of the normal vectors.

### 1.3.11. *Discussion of the methods*

#### Convergence towards theoretical values

The 3D mesh $\mathcal{M}$ is a discrete representation of an unknown continuous surface $\mathcal{S}$. Thus, a natural question is to assess whether the differential parameters computed by the methods described in the previous sections

5 http://people.cs.umass.edu/kalo/papers/curvature/index.html.

6 http://docs.pointclouds.org/trunk/classpcl_1_1_principal_curvatures_estimation.html.

converge towards the theoretical values obtained by explicit formulas (see section 1.2.5) when the discretization becomes "finer".

**Input:** Vertex $\mathbf{P_i}$
**Output:** Principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$ with the associated principal directions $\mathbf{t_1}(\mathbf{P_i})$ and $\mathbf{t_2}(\mathbf{P_i})$.
**begin**
Estimate the normal vector $\mathbf{n_i}$ at point $\mathbf{P_i}$ (we can find a reference to a method at the end of section 1.3.3). The plane passing by $\mathbf{P_i}$ and orthogonal to $\mathbf{n_i}$ can be considered as the tangent plane to $\mathbf{P_i}$. We can then define an orthogonal frame $(\mathbf{u}, \mathbf{v})$ in this tangent plane.
Decompose this tangent plane into six $60°$ slices around $\mathbf{P_i}$.
For each slice $j$, find the point of the cloud $\mathbf{P_i^j}$ whose projection on the slice is the closest to $\mathbf{P_i}$. If this projected point is not within a predefined distance, $\mathbf{P_i^j}$ will not be taken into account in the following steps.
Estimate the normal vectors $\mathbf{n_i^j}$ at points $\mathbf{P_i^j}$
Compute for all pairs of points the positional variation $\Delta\mathbf{P_i^{jj'}} = \mathbf{P_i^{j'}} - \mathbf{P_i^j}$ and the normal variation $\Delta\mathbf{n_i^{jj'}} = \mathbf{n_i^{j'}} - \mathbf{n_i^j}$.
After projection in the tangent plane frame $(\mathbf{u}, \mathbf{v})$, we can estimate the shape operator by (see equations [1.18]):

$$\begin{pmatrix} \Delta\mathbf{n_i^{jj'}} \cdot \mathbf{u} \\ \Delta\mathbf{n_i^{jj'}} \cdot \mathbf{v} \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \Delta\mathbf{P_i^{jj'}} \cdot \mathbf{u} \\ \Delta\mathbf{P_i^{jj'}} \cdot \mathbf{v} \end{pmatrix} \quad [1.29]$$

Given more than three pairs of points, we obtain an over-constrained system which can be solved by a least-squares method.
Once we get the coefficients $(a, b, c, d)$ of the shape operator matrix, we can compute its eigenvalues and eigenvectors to obtain the principal curvatures and the associated principal directions (see section 1.2.4).
An iteration process coupled with robust estimators can refine the estimation of normal variation and then the principal curvature values.
**end**

**Algorithm 8:** Computing differential parameters in the case of an unstructured 3D point cloud with the method from [KAL 09].

We could think that the convergence is only related to the local vertex density. But, in [BOR 03], it is shown that the angle defect, used in the DDG operator to compute the Gaussian curvature, depends also on the vertex valence. Moreover, in [RUS 04], the author describes some vertex arrangements that produce large errors in the normal cycle-based tensor method, whatever the density vertex is (note that the author also proposed a method to solve this problem with an implementation in C++ which is publicly available in the trimesh2 library[7]).

More generally, in [XU 05] the authors describe a counterexample mesh that prevents any Gaussian curvature and mean curvature method to converge. Based on the vertices of the mesh, they define different continuous surfaces by a quadratic approximation. This leads to different potential continuous curvature values which never correspond to the unique result given by a discrete method.

Nevertheless, it is proved in [LIU 07] that when $\mathcal{M}$ is a quadrilateral mesh, the Gaussian curvature computed with the DDG method has a quadratic convergence rate under a "parallelogram criterion". Moreover, when the quadrilateral mesh is based on a discrete net of curvature lines [BAU 10], the discrete estimation (by a specific described method) of the principal curvatures converges to the exact values when the distances between vertices become smaller. However, in [XU 09], it is emphasized that, in the general case, there is no convergence for a regular vertex with valence 4.

When $\mathcal{M}$ is a Delaunay triangulation restricted to the surface and under a local uniformity condition on the sampling, the authors of [COH 03] proved that the normal cycle-based curvature estimator converges linearly with respect to the sampling density. This case is important in practice as several surface reconstruction algorithms are based on the Delaunay triangulation of the input points.

A general convergence property can be found in [CAZ 05a], where it is proved that a polynomial fitting of degree $d$ estimates the principal curvatures and directions (away from umbilics) to accuracy $O(h^{d-1})$, where $h$ is the sampling density. However, in fact, this property depends under some conditions on the positions of points. They are not explicitly given in the

7 http://gfx.cs.princeton.edu/proj/trimesh2/.

paper but as in the counterexample described before, they are linked to the unicity of the approximation of the vertices by the polynomial function. In [XU 13], we can find a well-posedness condition (which is considered by the authors as a very mild one) which ensures that the Gaussian and mean curvatures computed by a quadratic local fitting have a linear convergence rate towards the continuous values.

### Robustness to noise

In many cases, the 3D mesh $\mathcal{M}$ is obtained by scanning a physical object. This requires at least three steps: acquisition, where the sensor takes measurements of points with a limited resolution (see, for example, [SAN 09] for a review of the available techniques), surface reconstruction which builds a 3D mesh by connecting or interpolating points based on regularity constraints [BER 14] and post-processing in order to smooth or "repair" the 3D mesh [WEY 04]. This processing pipeline introduces some errors or inaccuracies in the coordinates of vertices or in the mesh connectivity, which both form the so-called "*noise*". In methods to compute differential parameters, the problem is then to cope with these distortions, i.e. to be robust to noise.

The first idea is to optimize the algorithm parameters as the neighborhood size for the local fitting and the tensor of curvature methods or the radius of the ball for the integral estimators. In general, by increasing these values, we average the measures and then we diminish the influence of the noise.

The second idea consists of not taking into account all the measurements computed around $\mathbf{P_i}$ with the same reliability. For example, if we consider that the further a vertex $\mathbf{P_j}$ is from $\mathbf{P_i}$, the more inconsistent are its associated parameters (as the normal vector), we can introduce weights based on the distance $||\mathbf{P_i}\mathbf{P_j}||$. This is particularly useful when we sum some values over the neighbor vertices $\mathbf{P_j}$ as in the local fitting and tensor of curvature methods. Another way is to use some robust estimators [ROU 87] as the least trimmed squares in the minimization of $E$ in the local fitting method.

In fact, both ideas are often combined. For example, in [KAL 09] (see also the Windows executable[8]), the authors propose to use a neighborhood radius

8 http://people.cs.umass.edu/kalo/papers/curvature/index.html.

of three times the average distance of $\mathbf{P_i}$ to its direct neighbors, and at the same time, they implement an Iteratively Reweighted Least Squares approach to weight the contribution of each neighbor.

A last idea is to apply some smoothing process on the 3D mesh before computing the differential parameters. But the risk is then to make some geometric features disappear like, for example, the ridges. Indeed, comparing the values of the curvatures before and after smoothing appears to be a good method to measure locally the roughness of the 3D mesh (see [LAV 09] for more details).

### Multiscale approach

Several methods propose a multiscale approach. It consists of defining a series of $n$ values between a low and a high bound. Each value corresponds to a *scale* and is used to compute differential parameters. We then get a series of results for principal curvatures and directions, which are fused in definitive results by taking into account their scales.

These $n$ values can be used directly in a method. For example, they can represent neighborhood sizes for local fitting methods (see, for example, [PAN 10b]) or radii of the ball for integral estimators (as in [YAN 06]). Another way is to use these $n$ values to smooth the mesh $\mathcal{M}$ in order to build different versions, which will present different levels of detail according to the scale. We can then apply a method with fixed parameters to all these smoothed versions [MOK 01] and select the result given by the most adapted scale or combine the results obtained at different scales.

The main problem is to define the bound values. For example, if the value is too low, the neighborhood or the ball may be too small, too few vertices will be taken into account and the computed curvatures will be inaccurate. If the value is too high, the ball may be too large and may incorporate some parts of the mesh which are no more connected to the considered vertex (see [YAN 06], Figure 4) or the smoothing process will be so important that important geometric details of the mesh will be definitively lost. Very often, the bound values are defined proportionally to the average edge length in the mesh. But this solution is not adequate when the triangle size varies a lot and this is frequent when the mesh was reconstructed from a scan of a real object.

In [SEE 16], the authors propose an algorithm to find an optimal scale (in this case, the scale corresponds to the radius of the ball for the integral

estimator method) in an automatic way. The scales are defined independently for each vertex $\mathbf{P_i}$ in order to cope with local variation of the edge length. A first scale is defined by averaging the lengths of all the edges connected to $\mathbf{P_i}$. This value is then iteratively modified by taking into account the values computed for the neighboring vertices and this gives the lower scale $s_0(\mathbf{P_i})$. The upper scale is computed by $s_n(\mathbf{P_i}) = s_0(\mathbf{P_i}).f^i$, where $f$ is a float value greater than $1.0$ and $i$ an integer. $i$ is set in order that $s_n(\mathbf{P_i})$ corresponds to the radius of a ball of the same magnitude as the details of the surface we want to distinguish. For each scale $s \in [s_0, s_n]$, given the formulas of section 1.3.9, we compute the mean curvature $k_m^s(\mathbf{P_i})$ which is a function of $s$. If $\mathbf{P_i}$ is in a planar region, we have $k_m^s(\mathbf{P_i}) \approx 0$ and we define the optimal scale as the largest scale, which is below a given threshold. In the other cases, we draw the curve $k_m^s(\mathbf{P_i})$ w.r.t. $s$, which presents in general one or more extrema. The optimal scale is then computed based on this (or these) extrema. If there is no extremum (except planar regions), the middle scale is selected. The authors show some examples where the local selection of the optimal scale allows us to take into account finer details of the mesh than with a fixed scale.

### Comparison of the different methods

We have seen that many methods exist to compute differential parameters on a 3D mesh. Some of them have more theoretical foundations which give them some interesting properties of convergence or robustness; others are particularly adapted for a multi-scale approach. But is it possible to assess their practical efficiency?

In [GAT 06], the authors used different parametric surfaces for which they know the exact differential parameters (a sphere, a cylinder, a cubic polynomial or a sum of cosine and sine functions). Then, they introduce some distortions by tuning seven parameters in order to simulate different mesh resolutions, regularities, noises or valences. Methods of three classes – locally fitting parametric surfaces, DDG operators and tensor of curvatures – give results which are compared to the theoretical values. Among the conclusions, the authors emphasize that:

– some fitting methods (in particular cubic fitting) have better overall performance but at a quite important computational cost. The results are better with a large neighborhood;

– DDG methods are fast but they are more sensitive to valence, noise and mesh regularity;

– tensor of curvature method is based only on the vertex neighborhood and suffers from a severe sensitivity to noise normal to the surface.

In a similar manner, in [ANG 11], four methods belonging to the same three classes are tested on some parts of geometric primitives (a plane, a cylinder, a cone, a sphere and a torus). The authors introduce some noise in point coordinates and use different tessellations in order to analyze the sensitivity to mesh regularity and resolution. Their main conclusions are:

– all the methods have high sensitivity to noise if the neighborhood is restricted to the *1-ring*, i.e. the direct neighbors of the vertex;

– the local fitting method shows a quite general good capability whatever the vertex is, elliptical, hyperbolic, parabolic or an umbilic.

In [MAG 07], methods are tested on parametric NURBS surfaces and on scanned objects corresponding to geometric primitives (a plane, a sphere, a cylinder and a cone), the parameters of which were accurately measured. For each case, several meshes were produced corresponding to different resolutions, from about one hundred to several thousand triangles. Four different approaches – locally fitting a parametric surface, DDG, integrating 2D curvatures, tensor of curvatures – were selected and for each of them, a representative algorithm was implemented. By comparing the computed and theoretical values of the Gaussian and mean curvatures, the authors draw the following conclusions:

– DDG and surface fitting methods give best results on parametric and scanned data;

– on parametric data, DDG and surface fitting methods converge as the fineness of the mesh is improved;

– when resolution is very high in scanned objects, the relative error of the digitization process perturbs the accuracy of the computed curvatures.

In a recent paper [VÁŠ 16] (see also the companion paper [VÁŠ 17]), nine different methods belonging to locally fitting, DDG, normal cycle tensor of curvature and integral estimators classes are compared. They have all

been implemented in C# in a framework which is publicly available[9]. The framework is modular and data, distortion sources, curvature estimators or evaluation routines can be added easily. Parametric surfaces, implicit functions and NURBS surfaces have been discretized with different sampling schemes (e.g. rectangular, equilateral or random triangles) and different densities, and some noise was added. The computed and the theoretical curvatures were compared in order to assess the accuracy of the methods and the influence of their parameters. The main conclusions for noiseless meshes are that:

– DDG methods provide the best results but only if the mesh sampling is regular;

– local fitting methods provide comparably good results even when the mesh regularity is lower;

– the normal cycle tensor method and the integral estimator provide, in general, results of considerably lower accuracy as they tend to smooth the results.

Whereas for noised meshes, we can conclude that:

– all the methods which are based on a small neighborhood (e.g. only the vertex neighbors) fail to provide any reasonable estimation of the curvatures. Even when a larger neighborhood is used, methods like local fitting give quite bad results;

– the best results are obtained by integral estimators but only when an appropriate and quite large radius is used.

In fact, all these results demonstrate that no single estimator is efficient for all meshes. Thus, in [VÁŠ 16], the authors propose to construct a meta-estimator based on statistics collected during the evaluation of the different methods. This allows the meta-estimator to select one or several of them according to some properties of the input mesh (in particular, the ”smoothness” defined by the discrete Laplacian operator applied on vertex positions). In fact, even a very simple meta-estimator which chooses between only two or three estimators considerably improves the average accuracy.

9 http://graphics.zcu.cz/curvature.html.

Note also that the efficiency of all methods depends of course on mesh preprocessing (in particular of surface smoothing) as well as on implementation details. For example, we can see in [WAN 09] that the accuracy and stability of the local fitting method (by polynomials) may be significantly changed by the choice of the numerical solver.

## 1.4. Feature line extraction

### 1.4.1. *Introduction*

We can mathematically define some lines on the surface to characterize its shape. Detecting such feature lines provides robust shape descriptors, relevant from a geometric and a topological point of view. This is essential in many applications such as remeshing [ALL 03], shape indexing [MAE 96], shape interrogation [HAH 08] or non-realistic rendering [RÖS 00a].

Similarly to [HOS 92], we can separate feature lines into three classes: the lines which are defined on the surface independently of its shape, the lines which exist only around some specific shape configurations and the lines which depend on some external parameters such as the view point or the direction of the incident light.

The first class of feature lines is very useful to parameterize the entire surface in a way which is intrinsically adapted to its shape. It then becomes possible to analyze locally the surface and then to detect some patterns that define a particular shape feature. In section 1.4.2, we will focus on the concept of "lines of curvature".

In the second class of feature lines, we find in particular the lines which emphasize the salient parts of the surface which are very strong visual features of the shape. In section 1.4.3, we will describe the concept of "crest/ridge lines".

In [HOS 92], we can find several examples of feature lines belonging to the third class. In particular, we find the *contour curves* which represent the intersection of the surface with equidistance parallel planes or the *highlight curves* which are the locus of points where the brightness (which is computed according to the ray source direction and the surface normal) to the observer's eye is constant. In both cases, it requires the definition of a specific direction

to determine the plane orientation or the view point. But in most of 3D modeling applications, we are reluctant to introduce such external parameters. As a result, we choose not to study this class of feature lines in this book.

### 1.4.2. *Lines of curvature*

#### Application to shape description

We have seen in section 1.2.3 the definition of the lines of curvature and how they form a network of orthogonal curves all over the surface, except at umbilics. This network can be used to define an orthogonal mesh composed of quadrilateral *principal patches* [MAR 83, SIN 90], which parameterize a surface in an intuitive or a useful manner for several applications.

Moreover, equation [1.15] shows that the lines of curvature characterize the maximum variation of the normal vector. As most of the lighting effects (shading, specularity) that give the perception of a surface are based on the variation of the normal vector, it is then possible to visualize the global shape of an object just by displaying the principal directions (see, for example, [GIR 00]) or the lines of curvature.

Thus, when drawing a concept sketch of an object, a 3D designer intuitively uses lines of curvature to emphasize the surface bending. Sometimes, this is accurate enough to get a very realistic view of the object by automatic shading and texturing, as proposed in [IAR 15]. More precisely, a skilled 3D designer aligns the edges of the objects with lines of curvature, whereas in spherical areas which correspond to umbilics, points will be sampled isotropically. By using a remeshing or quadrangulation algorithm driven by lines of curvature as proposed in [ALL 03, KÄL 07a, LI 11], we can reconstruct a very ”efficient” 3D mesh, in the sense that it keeps the main geometric features of the initial mesh while it minimizes the number of faces. Lines of curvature can also be used to align strokes in automatic hatching techniques in order to display complex surfaces in a perceptually convincing way (see, for example, [RÖS 00a]).

Lines of curvature can also be integrated in a computer-aided design process. For instance, in [JOO 14], the authors show how the lines of curvatures of a curved surface can be used to define plates which can be locally adjusted in order to reconstruct this surface for shipbuilding or architectural free-form building applications. In [TSU 17], the authors

describe a method to reconstruct from scanned 3D data the surface of objects which have been created by sweeping a specified 3D section curve along a 3D spine curve (as with for example, the hood of a car). The idea is to estimate a first surface and then compute its lines of curvature. The lines of curvature with less torsion locally estimate the 3D spine curve. This makes it possible to then reconstruct a definitive parametric surface which looks visually acceptable for designers. The method described in [TAK 16] maps the four sides of each principal patch onto a plane. It then connects these patches one by one by aligning the equilength adjacent edges using translations and rotations. These strips can then be assembled to form a 3D surface from which it is possible to build 3D objects with sheets of light material as paper or plastic. It is even possible to build complex and large objects with sheets of metal or carbon fiber-reinforced plastics by adding stiffeners or frames along the lines of curvature. Note also an application to architecture in [MES 18] where the authors propose a methodology to generate surfaces with planar lines of curvature. These surfaces called super-canals can then be easily built by using flat panels.

### Computation of lines of curvature on parametric surfaces

Lines of curvature are not directly defined by an equation. They are in fact *integral curves* of the principal direction field which means that, at each point except umbilics, they are tangent to a principal direction.

Let us define the point $\mathbf{P_0}(u_0, v_0)$ on the surface $\Sigma$. We assume that it is not an umbilic and let $\mathcal{C}_i(u, v)$ $(i \in 1, 2)$ be one of the two lines of curvature going through $\mathbf{P_0}$ and tangent to the principal direction corresponding to the principal curvature $k_i$. We now have to find all the points $\mathbf{P}(u, v)$ that define $\mathcal{C}_i$.

For this, we can follow the method proposed in [MAE 96] (see also [FAR 98] or section 9.4 of [PAT 10]). We will express the line of curvature as $u = u(s)$ and $v = v(s)$ where $s$ is the arclength of $\mathcal{C}_i$ from $\mathbf{P_0}$. At each point $\mathbf{P}$, we can compute the coefficient $\gamma = dv/du$ by one of the two equations [1.9] or [1.10]:

$$\begin{cases} (M - k_i F) + \gamma(N - k_i G) = 0 \\ (L - k_i E) + \gamma(M - k_i F) = 0 \end{cases} \qquad [1.30]$$

The first equation can be written as $\gamma = (k_i F - M)/(N - k_i G)$. Then, at a point $\mathbf{P}(u, v)$, we can introduce the scalar function $\alpha(\mathbf{P})$ which is non-null and write:

$$\frac{du}{ds} = \alpha(N - k_i G) \quad [1.31]$$

$$\frac{dv}{ds} = \gamma \frac{du}{ds} = \alpha(k_i F - M) \quad [1.32]$$

As we work with arclength parameterization, we have:

$$||\frac{d\mathbf{P}}{ds}||^2 = 1$$

which can be rewritten by using the first fundamental form (see equation [1.1]):

$$E\left(\frac{du}{ds}\right)^2 + 2F\left(\frac{du}{ds}\frac{dv}{ds}\right) + G\left(\frac{dv}{ds}\right)^2 = 1$$

By using equations [1.31] and [1.32], we find:

$$\alpha = \pm \frac{1}{\sqrt{E(N - k_i G)^2 + 2F(N - k_i G)(k_i F - M) + G(k_i F - M)^2}} \quad [1.33]$$

But if we use the second equation of 1.30, we have $\gamma = (k_i E - L)/(M - k_i F)$ and we get another formulation of $\alpha$:

$$\alpha = \pm \frac{1}{\sqrt{E(M - k_i F)^2 + 2F(M - k_i F)(k_i E - L) + G(k_i E - L)^2}} \quad [1.34]$$

In conclusion, if we compute the coefficients $E$, $F$, $G$, $L$, $M$, $N$ and the principal curvatures $k_i$ at each point $\mathbf{P}$ of the surface, it is possible to compute $\alpha(\mathbf{P})$ by one of the two above equations. We can then integrate equation [1.31] or [1.32] to define the two lines of curvature $\mathcal{C}_i(u, v)$.

In practice, the integration can be performed by standard numerical techniques such as Runge–Kutta (fixed step solver) or Adams (variable step solver). This allows us to compute quite precisely the lines of curvature of parametric surfaces such as an ellipsoid $P(\theta, \phi) = (a \cos\theta \cos\phi, b \cos\theta$

$\sin\phi, c\sin\phi)$ [FAR 98], an elliptic paraboloid $P(u,v) = (u, v, au^2 + v^2)$ [JOO 14] or more generally a Bézier surface [JOO 14].

The accuracy of the lines of curvature depends on the number of integrated steps, but several issues arise:

– What formulation of $\alpha$ to use, either equation [1.33] or [1.34]? The idea proposed by [MAE 96] is to select the largest coefficients in equations [1.30] in order to avoid numerical inaccuracies. Since $(M - k_i F)$ is a common coefficient, equation [1.33] is used if $|N - kG| \leq |L + k_i E|$ and equation [1.34] otherwise. This also prevents the use of an equation with null coefficients.

– How can we choose the sign of $\alpha$? This sign determines the direction along which the line of curvature is built. As emphasized in [MAE 96], choosing a fixed sign does not guarantee that the tangent vector:

$$\frac{d\mathbf{P}}{ds} = \frac{d\mathbf{P}}{du}\frac{du}{ds} + \frac{d\mathbf{P}}{dv}\frac{dv}{ds}$$

does not change direction along the computation of the line of curvature. One idea is then to integrate in the direction which is the closest to the direction of the tangent vector computed at the previous integration step. This leads to the following condition:

$$||\frac{d\mathbf{P}}{ds} - \frac{d\mathbf{P^p}}{ds}|| < ||-\frac{d\mathbf{P}}{ds} - \frac{d\mathbf{P^p}}{ds}||$$

where $\frac{d\mathbf{P^p}}{ds}$ is the tangent vector at the previous integration step. If this condition is not true, the sign of $\alpha$ has to be inverted for the integration.

– In some points which are not umbilics, the two coefficients of one of the equations [1.30] could be null, preventing the use of this equation to compute $\gamma$. But [FAR 98] emphasizes that this occurs if and only if the principal directions are tangent to the surface parameter lines. For general free-form surfaces, this occurs only at very isolated points and then has little influence on the integration process.

– We have seen in section 1.2.3 that the lines of curvature are not defined at umbilic points. A problem then arises when a line of curvature passes through an umbilic. This corresponds to the fact that the point $\mathbf{P}$ obtained after the integration step is such that $k_1(\mathbf{P}) = k_2(\mathbf{P})$. In this case, [MAE 96] proposes

to slightly shift the position of $\mathbf{P}$. As for general free-form surfaces, umbilics are isolated points, this being a convenient way to avoid umbilic singularity and to draw long lines of curvature.

### Computation of lines of curvature on a discrete 3D mesh

To compute the lines of curvature on a discrete 3D mesh, the authors of [RÖS 00a] propose to use a simplified version of the method described in the previous section. First, they compute the principal direction vectors $\mathbf{t_1}$ and $\mathbf{t_2}$ for all the vertices of the 3D mesh. This can be done by any of the methods described in section 1.3. They then propose to define the principal direction vectors at any point on a facet (assumed triangular) using a barycentric interpolation of the principal direction vectors computed at the vertices of the facet. It is then possible to integrate, facet after facet, the line of curvature from a vertex $\mathbf{P_0}$ along one of the two principal direction $\mathbf{t}$ ($\mathbf{t}$ corresponds either to $\mathbf{t_1}$ or $\mathbf{t_2}$) (see Algorithm 9).

Nevertheless, this algorithm gives only a part of the line of curvature, $L^+$. We then have to run the algorithm again from $\mathbf{P_0}$ but along the opposite vector $-\mathbf{t}$ to get the second part $L^-$. The line of curvature $L$ going through $\mathbf{P_0}$ is then obtained by merging $L^+$ and $L^-$.

We also face some difficulties which are similar to the continuous case. First, we must be sure that all the principal direction vectors of the facet are oriented consistently in order that the barycentric interpolation gives a significant result. For this, it is proposed in [RÖS 00a] that when a segment $\mathbf{P_iP_{i+1}}$ enters a new facet $F_{i+1}$, the principal direction vectors of the three vertices of the facet are oriented in order that their dot product with $\mathbf{P_iP_{i+1}}$ is maximum.

The step value $h$ has a large influence on the integration of the line of curvature. In particular, if it is too large, we can easily deviate from the actual line. The idea is then to adjust it adaptively at each step, in particular with respect to the shape of the current facet $F_i$. We can find in [RÖS 00a] the following formula which assumes that the facets are triangles. It is heuristic but works well according to many experiments and is very fast to compute:

$$h_i = \frac{c}{2(1 + \sum_{j=1}^{3} \alpha_j)^2}$$

where $c$ is the circumference of the triangular facet $F_i$ and $\alpha_j$ the smallest angle between two principal direction vectors computed at vertices of the facet.

**Input:** Vertex $\mathbf{P_0}$ and its corresponding principal direction vector $\mathbf{t_0}$
Principal direction vector $\mathbf{t_i}$ at each vertex $\mathbf{P_i}$
Maximum length of the line of curvature $max\_length$
**Output:** List of points $\mathbf{L_k}$ defining the line of curvature.

$\mathbf{L_0} = \mathbf{P_0}$
$k = 0$
$F_0$ is the facet adjacent to $\mathbf{P_0}$ which is pointed by $\mathbf{t_0}$
**while** *($i < max\_length$) AND ($\mathbf{L_k}$ not on a boundary edge of the 3D mesh)*
**do**
  **begin**
    Let $F_{k+1}$ the facet containing $\mathbf{L_k}$ and opposite to $F_k$.
    Compute the principal direction vector $\mathbf{T_k}$ at point $\mathbf{L_k}$ by barycentric interpolation of the principal direction vectors $\mathbf{t_i}$ associated with the three vertices $\mathbf{P_i}$ of the facet $F_{k+1}$.
    Project $\mathbf{T_k}$ on $F_{k+1}$ and normalize the resulting vector. We get $\mathbf{T_k^P}$.
    Define the next point of the line of curvature by $\mathbf{L_{k+1}} = \mathbf{L_k} + h\mathbf{T_k^P}$, where $h$ is a given value corresponding to the integration step.
    if $\mathbf{L_{k+1}}$ is outside the facet $F_{k+1}$, compute the intersection point between $\mathbf{L_k L_{k+1}}$ and the traversed edge of $F_{k+1}$. $\mathbf{L_{k+1}}$ will then be relocated at this intersection.
    $k = k + 1$
  **end**
**end**

**Algorithm 9:** Computing lines of curvature on a 3D mesh.

This discrete integration scheme is improved in [ALL 03]. The authors compute $\mathbf{T_k^P}$ by using a 2D curvature tensor which is obtained by a discrete conformal parameterization which locally flattens the surface. They distinguish two sets of lines of curvature: one corresponds to the minimum principal curvatures and the other to the maximum principal curvatures. They also state that a line of curvature either starts from an umbilic and ends at another one, or is closed, or finishes on a mesh boundary. They first detect

umbilics in order to have starting points to compute the lines of curvatures. Then, they apply an accurate numerical scheme based on a fourth-order Runge–Kutta with an adaptive step based on a so-called "deviator" tensor.

In [MOR 10], the authors detail a method to detect the closeness of a line of curvature. It consists of monitoring the distances between $\mathbf{P_0}$ and the points of the line part $L^+$ and the ones of $L^-$. If the minimum distances to $L^+$ and $L^-$ both become small, it means that $L^+$ and $L^-$ go along the 3D mesh and then approach very close to $\mathbf{P_0}$; the line of curvature can then be considered as closed. Nevertheless, in most cases, the two parts $L^+$ and $L^-$ are not really connected. Two signed distance fields $d^+$ and $d^-$ are locally built on the 3D mesh from the points of $L^+$ and $L^-$. The sign is given by a projection on the principal direction vector orthogonal to the line of curvature, assuming that the frame of principal directions is direct. In fact, this separates locally the 3D mesh into a "left" side and a "right" side with respect to the line part. As the direction of integration of $L^+$ and $L^-$ are opposite, the isoparametric line $d^+(\mathbf{P})+d^-(\mathbf{P}) = 0$ corresponds to an average line between the two part lines. By construction, it is closed so it is a good approximation of the closed line of curvature $L$ going through $\mathbf{P_0}$.

In [KAL 09], the authors generalize Algorithm 9 in cases in which we no longer have a 3D mesh but an unorganized 3D point cloud (a simpler method based on the same framework can be found in [PAN 10a]). They first compute the differential parameters with Algorithm 8. Then they apply the same idea as in Algorithm 9. Suppose that we begin from point $\mathbf{P_0}$, we compute the principal direction vector $\mathbf{t_0}$ and we estimate the next point of the line of curvature by: $\mathbf{P_1} = \mathbf{P_0} + h\mathbf{t_0}$. Nevertheless, $\mathbf{P_1}$ lies on the estimated tangent plane of $\mathbf{P_0}$ but not on the surface sampled by the point cloud which is unknown. So, a projection step has to be added. For this, we take the points of the cloud which are in the neighborhood of $\mathbf{P_0}$ and we estimate the surface locally (see [KAL 09] for details). $\mathbf{P_1}$ is then projected on this surface, giving the next point $\mathbf{P'_1}$ of the line of curvature. Differential parameters at $\mathbf{P'_1}$ are then interpolated from the ones computed at points of the neighborhood of $\mathbf{P_0}$. The process is iterated by $\mathbf{P'_{i+1}} = projection[\mathbf{P'_i} + h\mathbf{t_i}]$ and gives points on the line of curvature. Note that at each step $i$, the value $h$ is adapted according to the potential error on the principal direction. For this, another point is computed as: $\mathbf{P''_{i+1}} = projection[\mathbf{P'_i} + h\mathbf{t_{i+1}}]$ (so with the principal direction estimated at $P'_{i+1}$). The step size is then modified as:

$h_{i+1} = h_i\sqrt{\tau/\Delta}$ where $\Delta = ||P'_{i+1} - P''_{i+1}||/||P'_{i+1} - P'_i||$ where $\tau$ is a predefined tolerance value.

### How to improve the computation of lines of curvature?

In all these methods, the main difficulty is to reduce the accumulation of local errors when building the line of curvature. These errors are due to:

– the inaccuracy of the vertex position, especially in the case of a scanned object;

– the inaccuracy in the computation of differential parameters due to the discretization of the 3D mesh;

– the drift during the discrete integration scheme.

A solution is to get some information about the pattern of lines of curvature on the 3D mesh in order to infer some global constraints. In [SOT 08], the authors characterize the set of lines of curvatures over a surface with the list of umbilic points, a first list of lines of curvature (called *separatrices*) which terminate or converge to an umbilic and a second list of closed lines of curvatures (called *cycles*) with some of their properties. This characterization allows us to state some theorems about the stability of the pattern of lines of curvature with respect to some classes of perturbations of the surface. In [ZHA 09], the authors extract on different surfaces what they call a $\mathcal{P}$ graph composed of the umbilics and the separatrices. They compute all the differential parameters on parametric surfaces defined by implicit equations (see [CHE 07] for formulas), which allows us to get exact values. Nevertheless, they emphasize the difficulty of isolating precisely the locations of umbilics which are, by definition, singular points on the surface.

Note that we can find some similarities in 3D flow visualization where we have to compute streamlines based on a local vector field. In this case, several techniques to extract topological features as singular points (sinks, sources, vortices) or limit cycles have been proposed (see, for example, [ARM 11]). But they are applied in a 3D isotropically discretized space, which allows us to make many computations with a higher accuracy than for a 3D mesh representing a surface where the sampling is locally 2D and irregular. Moreover, flows can be modeled by physical equations (as Navier–Stokes) and many assumptions can be made, in particular to characterize singular

points or the local shape of streamlines. Nevertheless, some techniques like the detection of closed streamlines proposed in [WIS 06] could be interesting to analyze the pattern of lines of curvature.

If we are able to find some global information about the pattern of lines of curvature, we can use it to improve the computation of these lines. In [VEM 93a] (see also [VEM 93b]), the authors propose to use a reference continuous surface $\mathcal{S}$ (e.g. a tube) gridded by its lines of curvature which are explicitly computed (e.g. the meridians and the parallels in the case of a tube). This surface $\mathcal{S}$ is then deformed in order to fit the vertices of the 3D mesh. At each step of the deformation, any point $\mathbf{M}$ of the surface $\mathcal{S}$ is associated with a point $\mathbf{P}$ of the 3D mesh. The tangent vectors of the isoparametric curves at point $\mathbf{M}$ are then aligned with the principal directions computed at $\mathbf{P}$, which deforms the surface $\mathcal{S}$. At the end, not only the deformable surface $\mathcal{S}$ is fitted to the mesh, but it is possible to emphasize the lines of curvatures of the mesh by tracing the isoparametric lines of $\mathcal{S}$. Then, by using a template grid of lines, it is possible to map a complete set of lines of curvature and to enforce the orthogonality of the principal directions. Unfortunately, results remain limited to quite simple and smooth scanned surfaces such as a light bulb. Moreover, this method can work only if there is no umbilic on the 3D mesh, otherwise we have to select a template with the appropriate pattern of lines of curvature around the umbilic(s) (for a classification of these patterns, see [SOT 08]).

Note that if we were able to accurately compute a discrete net of lines of curvature over the surface mesh, i.e. a net composed of polygonal approximation of lines of curvature, we could get the value of the principal curvatures at any point of the mesh with a bounded error [BAU 10].

### 1.4.3. *Crest/ridge lines*

#### A concept used in many applications

*Crest* or *ridge lines* are intuitively defined as topographic features, which describe the valley or the hill profiles of a landscape. Nevertheless, as emphasized in [KOE 93], the mathematical definition of these lines is not straightforward and it was heavily discussed in particular at the beginning of the 20th Century. Equations of crest/ridge lines are then proposed according to a natural frame where the $z$-axis is along the gravity direction and gives the

elevation. For example, in [KWE 94], the authors define these lines as the contour lines "forming modified V's" pointing upstream (for the valleys or ravines) or downstream (for the ridges) which correspond to local extrema of curvatures along a contour line (see also [HOS 92], section 7.4.1). But, as explained in [BRU 96], if we want to generalize the concept of crest/ridge lines to a 3D surface, we must drop the notion of a predefined direction and work in a local frame which is intrinsic to the surface. Note that two applications of crest/ridge lines to analyze the topography of a landscape are presented in sections 3.4 and 3.5.

Crest/ridge lines can also be seen as the edges of a manufactured object. In particular, if this object is modeled as a 3D mesh by a computer-aided design procedure, we can just threshold the value of the dihedron angle between two contiguous facets and keep only the most salient edges. Nevertheless, when the 3D mesh is given by a 3D scan of the object, the potential low resolution and/or noise may result in unstable or inaccurate lines. We can then use other measurements as the defect angle (see section 1.3.5) or use a large set of facets and smooth the measurements (see, for example, [HUB 01] or [JIA 08]). But, to get a robust method for noisy 3D meshes, it appears that it is necessary to take into account principal curvatures and directions [PAG 02, VID 11], especially if we want to get continuous lines and not a set of disconnected sharp edges.

Crest/ridge lines are also anatomical terms. They define the prominent borders of some anatomical (sub)structures such as, for example, the "cusp ridges" of a tooth, the "alveolar ridges" of the mouth, the "sagittal crest" of the skull or the "illiac crest" of the pelvis. We can also find equivalent terms like the "gyri", which are the ridges on the cerebral cortex or the "orbital rim" which is the crest line around the orbital opening. The huge development of 3D medical imaging systems allows radiologists to easily have access to 3D meshes of the anatomical structures of their patients [LEV 12]. Automatically detecting crest/ridge lines could assist medical doctors in precisely localizing and visualizing anatomical structures (see, for example, some applications to the brain in [STY 04], craniofacial anatomy in [ZHE 17] and paleo-anthropology in section 3.3), comparing them by registration (see, for example, applications to the brain in [SUB 99], the skull in [GUÉ 94, JAN 12] or the mandible in [AND 01]), analyzing their shape (e.g. to identify some anatomical facial features [KEN 96] or to make a diagnosis in craniofacial diseases [SUB 97]) or planing a surgical procedure [SUB 98].

## A survey of mathematical definitions

Around the middle of the 1990s, several applications to describe the shape of a 3D mesh based on *crest* or *ridge* lines were presented and there was a convergence of different mathematical definitions.

A first definition was proposed in [POR 71] (for a more detailed and extended version, see also [POR 01]) which is based on the focal surfaces (i.e. the sets of centers of principal curvatures; see section 1.2.9). Let us call $E$ the focal surface defined by one principal direction over the surface $\Sigma$. Now, if we follow the focal points $\mathbf{C}(\mathbf{P})$ corresponding to the points $\mathbf{P}$ of a given line of curvature of $\Sigma$, we define on $E$ a *focal curve*. If the focal curve fails to be regular at one point of $E$ (i.e. its derivative vanishes, which means the curve stops and backtracks on itself forming a cusp), then the corresponding point on $\Sigma$ is said to be a *ridge point*. As it can be shown that the set of non-regular points on $E$ is a smooth curve, we can deduce that ridge points form ridge lines on $\Sigma$.

A second definition can be inferred from a classification of the focal points $\mathbf{C}(\mathbf{P})$ and was proposed in the first edition of [POR 01]. In [BRU 96] (see also [BRU 99] for details and [CAZ 05b] for an overview), we can find a sketch of the demonstration. Let us write a local approximation of the surface $\Sigma$ at order 3 around the point $\mathbf{P}$, which extends the one described in section 1.2.8. We then get in a frame defined by the principal directions and the normal vector:

$$z(x,y) = \frac{1}{2}(k_1x^2 + k_2y^2) + b_0x^3 + b_1x^2y + b_2xy^2 + b_3y^3 + O(x^4, y^4) \quad [1.35]$$

where $k_1$ and $k_2$ are the principal curvatures at point $\mathbf{P}$ and $O(x^4, y^4)$ means terms in $x$ or $y$ with an order higher or equal to 4. In this frame, we can define the *focal spheres* $S_1$ and $S_2$ which are centered at focal points respectively located at positions $(0, 0, 1/k_1)$ and $(0, 0, 1/k_2)$ and which go through $\mathbf{P}$. Now, if we want to analyze the intersection between the focal spheres and the surface $\Sigma$, we have to solve (for the focal sphere $S_2$):

$$x^2 + y^2 + (z(x,y) - 1/k_2)^2 = 1/k_2^2$$

Bu using equation [1.35], we can write the equation of the projection in the plane $(x, y)$ of the intersection curve between $\Sigma$ and $S_2$. If we limit to order 3, we get:

$$x^2\left(1-\frac{k_1}{k_2}\right)-\frac{2}{k_2}(b_0x^3+b_1x^2y+b_2xy^2+b_3y^3)+O(x^4,y^4)=0$$

If $\mathbf{P}$ is not an umbilic, we get $k_1 \neq k_2$ and we have a term in $x^2$. Then, if $b_3 \neq 0$, the intersection curve can be approximated as something like $x^2 \pm y^3 = 0$ which is a single cusp. But if $b_3 = 0$, the intersection curve becomes something like $x^2 \pm y^4 = 0$ which is a tacnode or a double cusp. We can consider that, in this case, the focal sphere $S_2$ has a closer contact with the surface $\Sigma$ and that $\mathbf{P}$ is a *ridge point*.

In fact, this geometric definition can be transformed in a simple equation relative to $k_2$ and its associated principal direction $\mathbf{t_2}$. For that (for some mathematical details, see [BEL 97, CAZ 05b]), we again use equation [1.35]. As the frame $(x, y)$ is along the principal directions, the normal section $S_{\mathbf{t_2}}$ along $\mathbf{t_2}$ can be defined by:

$$s_{\mathbf{t_2}}(y)=\frac{1}{2}k_2y^2+b_3y^3+O(x^4,y^4)$$

We can compute the curvature of this 2D curve by derivation and if we approximate the formula to order 1 in $y$, this leads to an approximation of $k_2$ around $\mathbf{P}$ along its associated principal direction $\mathbf{t_2}$:

$$k_2(y)=k_2+6b_3y+O(y^2)$$

We have seen before that $\mathbf{P}$ is defined as a ridge point when $b_3 = 0$. In this case, the above equation shows that $k_2$ reaches a local extremum at point $\mathbf{P}$ along the principal direction $\mathbf{t_2}$. We can do the same process for $k_1$ so we can write the following general definition for $j \in \{1, 2\}$

$$\mathbf{P} \text{ is a } \textit{ridge point} \text{ when } \nabla k_j(\mathbf{P}) \cdot \mathbf{t_j}(\mathbf{P}) = 0 \qquad [1.36]$$

In [POR 01], it is shown that this definition of ridge points which is based on extrema of principal curvatures is equivalent to the first definition which was based on focal surfaces. This implies that the ridge points defined by equation [1.36] form *crest/ridge lines* on the surface $\Sigma$.

Equation [1.36] can also be found in [HOS 92] where it defines the *principal curvature extremum curves*.

Now, if we set $k_{amax} = max(|k_1|, |k_2|)$ (note that we use absolute values) and $\mathbf{t_{amax}}$ its associated principal direction, the equation:

$$\nabla k_{amax}(\mathbf{P}) \cdot \mathbf{t_{amax}}(\mathbf{P}) = 0 \qquad [1.37]$$

defines a subset of ridge points called *crest points* in [BRU 96]. These points form the *crest lines* which are then a subset of the ridge lines. In fact, we can find a similar definition in a previous paper [GUÉ 93] even though the definition of $k_{amax}$ was not clearly presented. In this chapter, it is shown that crest lines characterize the salient curves on a surface.

Note that the lines, defined by the equation $\nabla k_i(\mathbf{P}) \cdot \mathbf{t_j}(\mathbf{P}) = 0$ where $j \neq i$ (that is $k_1$ reaches an extremum along the principal direction $\mathbf{t_2}$ or $k_2$ reaches an extremum along the principal direction $\mathbf{t_1}$) are called *sub-parabolic lines* (as there is a connection with the parabolic points (see section 1.2.8) of the focal surface) and are thoroughly analyzed in [MOR 96].

In [BEL 96], the authors propose five different definitions of ridge lines based on extrema of curvatures. These extrema are not only along the associated principal direction but may also be along the normal or the vertical section curve passing through $\mathbf{P}$ in the direction of the maximum principal curvature. The authors then show some connections between all these different definitions.

A third definition is based on the concept of symmetry. In [NAC 85], the Symmetric Axis Transform is generalized to a 3D surface by using spheres which touch the surface at two points; the symmetric axis then becomes a symmetric surface. In [YUI 90], the authors prove that the intersection of these symmetry surfaces with the surface are generated by and only by lines on the surface corresponding to the extrema of principal curvature. Nevertheless, they do not explicitly use the terms crest or ridge line. Moreover, the authors of [ANO 94a] emphasize that the symmetric axis transform can be used only to find the ridge lines, which are along the convexities of the surface and not those which are along the concavities. They propose then in [ANO 94b] a new definition of the skeletonization based on singularity theory in order to define all ridge lines. In this chapter, they also

make a connection between the definitions based on the focal surfaces, the differential singularities and symmetry (see also [BEL 96]).

A fourth definition is based on an implicit definition of the surface in a 3D image. Let a 3D image (i.e. a regular grid) $I$ where each element (called a *voxel*) of coordinates $(x, y, z)$ is associated with a scalar value $I(x, y, z)$. As for a 2D image, we can compute the differentials of intensity, at any order, along the three axes (as, for example, $\frac{\partial I}{\partial x}$ or $\frac{\partial I^2}{\partial x \partial y}$). The implicit equation $I(x, y, z) = I_0$ defines then a surface $\Sigma$ called an *isosurface* of the 3D image $I$. In [THI 95], we can find the formulas to compute differential parameters at any point $\mathbf{P}(x, y, z)$ of $\Sigma$ based on the intensity and its derivatives at the corresponding voxel. For example, the first coefficient of the first fundamental form is given by:

$$E(x, y, z) = \left( \frac{\partial I(x, y, z)}{\partial x}^2 + \frac{\partial I(x, y, z)}{\partial z}^2 \right) / \frac{\partial I(x, y, z)}{\partial z}^2$$

Note that [SAN 90] and [MON 95] also propose some formulas based on image differentials, but they are based on a local parametric representation of the isosurface (obtained by fitting a surface patch in the case of the first reference) which is less general than using the implicit representation.

In [MON 92] (for a more detailed version, see also [MON 95]), the authors use such formulas to detect what they called the ridge and valley points on $\Sigma$. These points correspond to the zero-crossing of what they call the *extremality criterion* $e$ which is defined as:

$$e = \nabla k_{max} \cdot \mathbf{t_{max}} = 0$$

In fact, this is exactly the same as equation [1.37]. In [THI 96a], this extremality criterion is generalized to the two principal curvatures and their associated principal directions resulting in four different extremality zero-crossings which define *extremal lines* (which include the ridge lines). In [EBE 94], the authors extend the definition of the ridge and valley points to $n$-dimensional images by generalizing the concept of isosurface to a specific level set of dimension $n - 1$. They also point out the connection between this image-based definition and the symmetry-based definition.

In the following, we will call the *crest/ridge function* the generalized version of equation [1.36]:

$$e_j(\mathbf{P}) = \nabla k_j(\mathbf{P}) \cdot \mathbf{t_j}(\mathbf{P}) \qquad [1.38]$$

where $j$ may be either one of the two principal curvatures (and associated principal directions) ($j \in \{1, 2\}$) or the maximum ($j = max$) or minimum ($j = min$), or the maximum in absolute value ($j = amax$). The equation $e_j = 0$ defines then crest/ridge points and crest/ridge lines.

### Computation of crest/ridge lines on parametric surfaces

If we assume that we have a parametric form $f(u, v)$ of the surface $\Sigma$, we are able to compute formally, and then exactly, without any numerical approximation, all the differential parameters. Nevertheless, some problems must be investigated before proposing a method to compute the crest/ridge lines.

At any point $\mathbf{P}$ except umbilics, we have two principal directions that we can order with respect to the value of their associated principal curvatures. We then define $k_{min} = min(k_1, k_2)$ and $k_{max} = max(k_1, k_2)$ which are respectively associated with their corresponding principal directions $\mathbf{t_{min}}$ and $\mathbf{t_{max}}$. For clarity in the following, we will call, according to [POR 01], *blue* crest/ridge lines the lines defined by the equation $e_{max} = \nabla k_{max}(\mathbf{P}) \cdot \mathbf{t_{max}}(\mathbf{P}) = 0$ and *red* crest/ridge lines those defined by the equation $e_{min} = \nabla k_{min}(\mathbf{P}) \cdot \mathbf{t_{min}}(\mathbf{P}) = 0$. Note that some researchers, such as [BRU 96], or [THI 96a] order the principal directions according to their absolute values. Blue and red ridge lines may then change colors but keep the same structure.

Blue and red ridge lines do not auto-intersect but may cross in so-called *purple points*, which were first described in [POR 01] and studied in [CAZ 06]. Umbilic points can also be considered as crest/ridge points since they are in the closure of crest/ridge lines. More precisely, it can be shown (see, in particular, [CAZ 05b]) that generic umbilics are of two types: either three crest/ridge lines cross or only one crest/ridge line crosses at an umbilic. This property allows us to define a graph of crest/ridge lines over all of a closed surface where the nodes are umbilics or purple points (see, in particular, the ellipsoid example in [CAZ 05b]). Such a graph description was also proposed under the name of *extremal mesh* in [THI 96a].

A crucial problem is that the principal directions are not intrinsically oriented along their corresponding line of curvature (see section 1.2.3). If we invert the orientation of the principal direction frame $(\mathbf{t_1}, \mathbf{t_2})$ in $(-\mathbf{t_1}, -\mathbf{t_2})$, $e_{min}$ and $e_{max}$ turn respectively into $-e_{min}$ and $-e_{max}$. But then, how to compute the zero-crossings of an expression whose sign is not clearly defined?

One solution is to impose a continuity of the orientation of the principal direction $\mathbf{t_i}$ in a neighborhood which can be defined in space or along the corresponding line of curvature. This means that if $\mathbf{P}$ and $\mathbf{P'}$ are close, the two principal directions make an acute angle, i.e. $\mathbf{t_i}(\mathbf{P}) \cdot \mathbf{t_i'}(\mathbf{P'}) > 0$. This constraint is described in [MOR 96] and called the *Acute Rule* in [CAZ 06].

Another solution is to use the Gaussian extremality [THI 96a] which is defined by: $e_g(\mathbf{P}) = e_{min}(\mathbf{P}).e_{max}(\mathbf{P})$. In this case, if we assume that the frame $(\mathbf{t_{min}}, \mathbf{t_{max}})$ is direct, changing the orientation of $\mathbf{t_{min}}$ results in changing the orientation of $\mathbf{t_{max}}$ and that does not change the sign of $e_g$. But then we get all the crest/ridge lines, without being able to distinguish between the blue and red ones.

We can classify the methods to compute crest/ridge lines on parametric surfaces into five categories: formal computation of zero-sets, numerical computation of zero-sets, incremental tracing, marching by interpolation or integration of a direction field.

In the first category, we formally solve the crest/ridge equation $e_i = 0$ by explicitly computing the formulas of partial derivatives to order 3 of $f(u, v)$. But this is computationally very expensive; for example, it is shown in [CAZ 08a] that in the case of a Bézier surface of degree 4, it requires solving a bivariate polynomial of total degree 84, with 1,907 terms! This makes methods of this category impossible to use on complex parametric surfaces such as NURBS.

In the second category of methods, we numerically evaluate the crest/ridge function $e_i$ at only some points of the surface. A first method was described in [HOS 92], section 7.4, where it is proposed that we find the extremum points of curvature along lines of curvature. As it is too complex to formally solve the mathematical equations, the idea is to determine these extremum points numerically by interpolating points which are sampled along the line of

curvature. Nevertheless, in this chapter, the computed points are not linked to forming crest/ridges lines. In [GUÉ 93, GUÉ 95], the parametric surface is defined by a tensor product of spline functions. By regularly sampling the parameter space it becomes possible to compute the value of the curvature $k_i$ on the surface and detect the local maximum points. These points are then linked based on their proximity. The author compares this method with the numerical resolution of the crest/ridge line equation $e_i = 0$ and concludes that results are similar.

The third category of methods proposes to trace incrementally small parts of the crest/ridge lines. For example, the algorithm described in [MUS 11] is based on the property of crest/ridge lines that they transversely intersect the corresponding curvature lines (i.e. the ones which are defined with the same differential parameters $k_i$ and $\mathbf{t_i}$), except at a few isolated points (called *turning points*) where they are tangent.

Let us assume that we want to compute the crest/ridge lines based on $e_1 = 0$. The idea is to begin at a seed crest/ridge point $\mathbf{R}$. Then we are going to take a neighbor point $\mathbf{P}$ on the surface. Then, if we slide $\mathbf{P}$ along the curvature line defined by $\mathbf{t_1}$, it will cross the crest/ridge line. Once we have determined the intersection point, we can link it to $\mathbf{R}$ in order to get a local linear approximation of the crest/ridge line. Starting from $\mathbf{R}$, the algorithm is then composed of the following steps:

– define a neighbor point on the surface $\mathbf{R_{adv}}$. This is practically performed by advancing a step $\epsilon_{adv}$ along the line of curvature defined by $\mathbf{t_2}(\mathbf{R})$ (thus orthogonally to $\mathbf{t_1}(\mathbf{R})$) and by projecting the point $\mathbf{R} + \epsilon_{adv}\mathbf{t_2}(\mathbf{R})$ on the surface;

– now, define the point $\mathbf{R_{slide}}$ on the surface by sliding a step $\epsilon_{slide}$ from $\mathbf{R_{adv}}$ along the line of curvature defined by $\mathbf{t_1}(\mathbf{R_{adv}})$ and projecting the point $\mathbf{R_{adv}} + \epsilon_{slide}\mathbf{t_1}(\mathbf{R_{adv}})$ on the surface;

– test whether $\mathbf{R_{slide}}$ is a crest/ridge point using the equation $e_1(\mathbf{R_{slide}}) = 0$:

- if not, iterate again the sliding step from $\mathbf{R_{slide}}$,

- if a crest/ridge point is reached, use $\mathbf{R_{slide}}$ as a starting point to continue the tracing of the crest/ridge line and go back to step 1 of the algorithm.

Note that the Acute Rule is used to orient consistently principal directions. The set of seed crest/ridge points is defined by finding the critical points of curvature (i.e. where the curvature gradient is equal to zero) and the umbilics. In fact, umbilics are included as seed points since crest/ridge lines may pass through these points. This algorithm has been successfully tested on rational tensor product B-spline surface representations.

An example of the fourth category of methods is given in [CLÉ 08]. This is called the marching method as it is based on a regular discretization of the parameter space and a tracing by interpolation from one discrete element to a neighbor one. More precisely, we can use a regular grid where each node $(u_j, v_k)$ corresponds to a point $\mathbf{P_{j,k}}$ on the parametric surface. We then compute the crest/ridge function at this node by $e_i(j,k) = e_i(\mathbf{P_{j,k}})$. Now, for each square of the grid which is defined by four nodes, if at least one node has a negative crest/ridge value and at least one node has a positive crest/ridge value, we can assume that a line segment corresponding to $e_i = 0$ crosses the square and we can draw it based on interpolation of values $e_i(j,k)$ of the square. At the end, we link all the line segments to get a continuous line going through the parameter space. This line corresponds, in the 3D space, to a crest/ridge line. Of course, the geometric accuracy is directly related to the discretization step. This implies that it may be difficult to detect some intersection, loop or tangency problems if the step is not small enough.

Note that the "Marching Lines" term is also used in [THI 96b]. In this case, the parametric surface is defined by an implicit equation. If we discretize the space into a 3D regular grid of cubes (also called voxels), we can approximate each intersection between the surface and a cube by a patch composed of one or several polygons. If we now compute, at each vertex of the patch, the crest/ridge function, we can interpolate line segments which will form crest/ridge lines.

The fifth category of methods is based on the construction of a vector field in the parameter space. Our goal is to compute the crest/ridge lines which correspond to the isolines $e(u,v) = 0$. By definition, when a point $\mathbf{P}(u,v)$ moves along this isoline, we have $\frac{\partial e}{\partial u}du + \frac{\partial e}{\partial v}dv = 0$. This implies that the tangent vectors of isolines form, in the parameter space, a vector field defined by: $(\frac{\partial e}{\partial v}, -\frac{\partial e}{\partial u})$.

Now, if we start from a crest/ridge point $\mathbf{P_0}(u_0, v_0)$, we can integrate from $(u_0, v_0)$ the vector field and we build a line in the parameter space which corresponds to a crest/ridge line in the 3D space.

In [CLÉ 08], we find example of implementation of such an integration method. The algorithm is composed of the following steps:

– a uniform random distribution of points is generated over all of the parameter space. We select the points with a small value of the crest/ridge function. Then, by applying a conjugate gradient-based algorithm, we can find the local minima of this function. This will define crest/ridge points (including umbilics) as $\mathbf{P_i}$;

– numerical instability often appears when computing the partial derivatives of $e$ around the points $\mathbf{P_i}$. So, we define a circle centered in $\mathbf{P_i}$ with a radius so that we are far from the instability area. On this circle, we can find one or several points $\mathbf{Q_i^k}$ which have small values of the crest/ridge function. The set of $\mathbf{Q_i^k}$ and the vectors $\mathbf{P_iQ_i^k}$ will be, respectively, the initial points and the initial directions for the integration process starting from $\mathbf{P_i}$;

– as there may be some error accumulation during integration, the crest/ridge line starting from a point $\mathbf{Q_i^k}$ (which is linked to the point $\mathbf{P_i}$) may diverge and not reach another crest/ridge point $\mathbf{P_j}$. The idea is then to build a Voronoï tessellation of the parameter space based on the set of all $\mathbf{Q_i^k}$ (then for all $i$ and $k$). We will integrate the crest/ridge line inside each Voronoï cell until it reaches its border. Then we are sure to have line parts which are linked to points $\mathbf{P_i}$. A fifth-order Runge–Kutta method is used for numerical integration;

– now, when the extremities of two line parts coming from $\mathbf{Q_i^k}$ and $\mathbf{Q_j^l}$ are close on both side of a Voronoï cell, we connect them and we get a longer crest/ridge line part which links $\mathbf{P_i}$ and $\mathbf{P_j}$. We update the Voronoï tesselation after removing $\mathbf{Q_i^k}$ and $\mathbf{Q_j^l}$, we apply the integration process and link another couple of points if it possible.

The authors tested the method on a fourth-order polynomial function and emphasize that its efficiency is strongly related to the accuracy of the numerical integration scheme.

Another integration method is described in [CHE 11]. It is also based on a vector field computed in the parameter space. In order to deal with the problem of error accumulation, the authors propose projecting the estimation

of a new crest/ridge point on its corresponding line of curvature in order to find the optimal integration step. This method can deal with some singular points such as turning points and was tested on several bicubic Bézier patches which correspond, in particular, to manufactured objects.

### Computation of crest/ridge lines on a discrete 3D mesh

One of the first methods to compute crest/ridge lines on a discrete 3D mesh was proposed in [WAT 01]. In fact, it does not belong to any category of methods seen in the previous section. This method is based on the detection of particular configurations of the focal surfaces by using the following property demonstrated in [LUK 98] and [WEA 55], item 75.

Let us work in the frame of the lines of curvature $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$. We assume that none of the principal curvatures are null which allows us to define the focal surfaces $E_1$ and $E_2$ (see section 1.2.9). If $\mathbf{P}$ moves a little over $\Sigma$, it will sweep the area:

$$A = ||\frac{\partial \mathbf{P}}{\partial u} \times \frac{\partial \mathbf{P}}{\partial v}||$$

At the same time, the corresponding focal point $\mathbf{C_1}$ will move over $E_1$ and sweep an area given by:

$$A' = \|\frac{\partial \mathbf{C_1}}{\partial u} \times \frac{\partial \mathbf{C_1}}{\partial v}\|$$

We have seen at the end of section 1.2.9 that for a point $\mathbf{P}$:

$$\frac{\partial \mathbf{C_1}}{\partial u} = \frac{1}{k_1^2}\frac{\partial k_1}{\partial u}\,\mathbf{n}$$

$$\frac{\partial \mathbf{C_1}}{\partial v} = \left(1 - \frac{k_2}{k_1}\right)\frac{\partial \mathbf{P}}{\partial v} + \frac{1}{k_1^2}\frac{\partial k_1}{\partial v}\,\mathbf{n}$$

We then get:

$$A' = |\left(1 - \frac{k_2}{k_1}\right)\frac{1}{k_1^2}\frac{\partial k_1}{\partial u}|\ \|\frac{\partial \mathbf{P}}{\partial v}\|$$

which allows us to compute the area-ratio $r_1$:

$$r_1 = \frac{A'}{A} = \frac{|\left(1 - \frac{k_2}{k_1}\right) \frac{1}{k_1^2} \frac{\partial k_1}{\partial u}| \, \|\frac{\partial \mathbf{P}}{\partial v}\|}{\|\frac{\partial \mathbf{P}}{\partial u}\| \, \|\frac{\partial \mathbf{P}}{\partial v}\|}$$

$$r_1 = \frac{|\left(1 - \frac{k_2}{k_1}\right) \frac{1}{k_1^2} \frac{\partial k_1}{\partial u}|}{\|\frac{\partial \mathbf{P}}{\partial u}\|}$$

This means that $r_1$ tends to 0 if and only if:

– $k_1 = k_2$, which means that the point $\mathbf{P}$ is an umbilic.

– $\frac{\partial \mathbf{P}}{\partial u} = 0$ when $\mathbf{P}$ moves along the line of curvature defined by $\mathbf{t_1}$. This corresponds exactly to the definition of the crest/ridge point given by equation [1.36].

Of course, we can do exactly the same for the other focal point $\mathbf{C_2}$ moving on $E_2$, and we get an area-ratio $r_2$ with the same property along the line of curvature defined by $\mathbf{t_2}$.

The authors of [WAT 01] then apply this "area degenerating" property in the discrete case of a 3D mesh. Note that they explain that they use this property instead of directly detecting non-regular points of a focal curve (see the first definition in the section *A survey of mathematical definitions* in section 1.4.3) as this latter method is too unstable for complex 3D meshes.

Let us assume that we have computed the normal vector $\mathbf{n}(\mathbf{P_i})$ and the principal curvatures $k_1(\mathbf{P_i})$ and $k_2(\mathbf{P_i})$, or all vertices of the 3D mesh $\mathcal{M}$ by any of the methods described in section 1.3. For each facet $F_i$ of the 3D mesh (which is assumed to be triangular), we can define the three focal points corresponding to the three vertices of the face, $\mathbf{P_i^0}$, $\mathbf{P_i^1}$, $\mathbf{P_i^2}$:

$$\mathbf{C_i^0} = \mathbf{P_i^0} + \frac{1}{k_j(\mathbf{P_i^0})}\mathbf{n}(\mathbf{P_i^0})$$

$$\mathbf{C_i^1} = \mathbf{P_i^1} + \frac{1}{k_j(\mathbf{P_i^1})}\mathbf{n}(\mathbf{P_i^1}) \quad \mathbf{C_i^2} = \mathbf{P_i^2} + \frac{1}{k_j(\mathbf{P_i^2})}\mathbf{n}(\mathbf{P_i^2})$$

where $k_j$ corresponds either to $k_1$ or $k_2$.

We can compute then two area-ratios $r_1$ and $r_2$ for the facet $F$ by:

$$r_j(F) = \frac{A(\mathbf{C_i^0 C_i^1 C_i^2})}{A(\mathbf{P_i^0 P_i^1 P_i^2})}$$

where $j \in \{1, 2\}$ depending if we choose $k_1$ or $k_2$.

According to the above property, a facet $F$ where one of the area-ratios $r_1(F)$ or $r_2(F)$ is high (a threshold value 30% of all the area-ratios of the mesh facets is proposed in the paper) potentially contains a crest/ridge point. Note that, to have good results on complex meshes, it is necessary to smooth the normal vector coordinates and curvature values at a vertex by a nonlinear averaging in its neighborhood.

Nevertheless, a crest/ridge facet with a small area-ratio may, in fact, be a flat umbilic. If we compute for this facet $F$ a *curvedness* value $c(\mathbf{P_i^j}) = |k_1(\mathbf{P_i^j})| + |k_2(\mathbf{P_i^j})|$ at its three vertices and apply a threshold (such as, for example, the 50% percentile of all the values computed on the entire 3D mesh), we can discard the potential crest/ridge facets which are not salient enough.

Nevertheless, we have a set of crest/ridge facets but not lines. For this, the authors propose to use a thinning technique which generalizes the skeletonization process, well-known in image processing. The idea is to "peel-off" the set of crest/ridge facets in order to only keep a one-triangle width undisconnected band of facets (for more details, see the next section).

In [YU 08], the authors have developed both CPU and GPU-based algorithms in order to efficiently approximate the focal meshes. They could then be used to apply the above method to very large 3D meshes.

All the other methods which have been proposed to compute crest/ridge lines on a discrete 3D mesh belong only to the category "numerical computation of zero-sets" defined in the previous section. In the following, we describe some of the most referenced methods.

In the method proposed in [STY 04], the first step consists of detecting the crest/ridge points on the 3D mesh and is described in the following algorithm:

**Input:** 3D mesh composed of vertices $\mathbf{P_i}$
**Output:** Flag $f_i \in \{false, true\}$ ($f_i = true$ if $\mathbf{P_i}$ is a crest/ridge point)
**begin**
  **forall** *vertex* $\mathbf{P_i}$ **do**
    Compute the differential parameters by fitting locally a quadric to the mesh (see section 1.3.4).
    We have then the principal curvatures $(k_1^i, k_2^i)$ and the associated principal directions $(\mathbf{t_1^i}, \mathbf{t_2^i})$.
  **end**
  **forall** *vertex* $\mathbf{P_i}$ **do**
    Select the maximum principal curvature $k_{max} = max(k_1^i, k_2^i)$ and the associated principal direction $\mathbf{t_{max}}$.
    Project $\mathbf{t_{max}}$ on the facets connected to $\mathbf{P_i}$.
    Find the two facets $F_i'$ and $F_i''$ which contain the projection of $\mathbf{t_{max}}$. These two facets are on opposite sides with respect to $\mathbf{P_i}$.
    The projection of $\mathbf{t_{max}}$ intersects the edges of $F_i'$ and $F_i'$, opposite to $\mathbf{P_i}$, respectively at the points $\mathbf{P_i'}$ and $\mathbf{P_i''}$.
    By linear interpolation of the maximum curvature of the two vertices of the edge, it is possible to compute the maximum principal curvature $k'_{max}$ at $\mathbf{P_i'}$ and $k''_{max}$ at $\mathbf{P_i''}$.
    If $((k_{max})^2 - (k'_{max})^2 > \epsilon$ AND $(k_{max})^2 - (k''_{max})^2 > \epsilon)$, this means that the maximum curvature of $\mathbf{P_i}$ is locally maximum in its corresponding principal direction, i.e. $\mathbf{P_i}$ is a crest/ridge point. In this case $f_i = true$ else $f_i = false$.
  **end**
**end**

**Algorithm 10:** Detect crest/ridge points in a 3D mesh by the method [STY 04].

The problem is then to link crest/ridge points in order to build lines over the 3D mesh. First, a region growing process is applied in order to create, for each crest/ridge point, a crest/ridge region corresponding to its 2-neighborhood (i.e. the points which belong to the neighborhood of the neighborhood). Then, as for the focal method described before, we use a skeletonization process in order to obtain a set of connected lines along the edges of the 3D mesh.

Another method was proposed in [OHT 04] and is described in the following algorithm:

**Input:** 3D mesh composed of vertices $\mathbf{P_i}$
**Output:** Crest/ridge lines as a set of segments
**begin**
  **forall** *vertex* $\mathbf{P_i}$ **do**
    Construct an implicit surface which approximates locally the vertices of the neighborhood $N(\mathbf{P_i})$ as their normal vectors.
    Based on this parametric representation, it is possible to compute the principal curvatures $(k_1^i, k_2^i)$ and the associated principal directions $(\mathbf{t_1^i}, \mathbf{t_2^i})$ at vertex $\mathbf{P_i}$ (see the end of section 1.2.5).
    Select the maximum principal curvature $k_{max}^i = max(k_1^i, k_2^i)$ and the associated principal direction $\mathbf{t_{max}^i}$.
    As we have a parametric representation, we can compute the crest/ridge function $e_{max}^i = \nabla k_{max}^i \cdot \mathbf{t_{max}^i}$ at the vertex $\mathbf{P_i}$ (see equation [1.38]).
  **end**
  **forall** *edge* $\mathbf{P_iP_j}$ **do**
    If the angle between $\mathbf{t_{max}^i}$ and $\mathbf{t_{max}^j}$ is obtuse, flip $\mathbf{t_{max}^i}$ which inverts the sign of $e_{max}^i$. This corresponds to the Acute Rule constraint (see section 1.4.3).
    If $e_{max}^i \,.\, e_{max}^j < 0$ then there is a zero-crossing of $e_{max}$ which corresponds to a crest/ridge point on the edge.
    If we want to recover only the maxima of $k_{max}$, this means that when you go along the edge from $\mathbf{P_i}$ to $\mathbf{P_j}$, the derivative of $k_{max}$ along the direction $t_{max}$ should be positive, then negative. This can be approximated by one of the conditions:

$$(\mathbf{P_iP_j} \cdot \mathbf{t_{max}^i})\, e_{max}^i > 0$$

$$(\mathbf{P_jP_i} \cdot \mathbf{t_{max}^j})\, e_{max}^j > 0$$

    If one condition is true, approximate the position of the crest/ridge point $\mathbf{P_{ij}}$ by linear interpolation of the positions of $\mathbf{P_i}$ and $\mathbf{P_j}$, respectively weighted by $e_{max}^i$ and $e_{max}^j$.
  **end**
  **forall** *facet of the mesh* **do**
    If 2 crest/ridge points belong to the edges of the facet (which is assumed triangular), connect them by a segment.
    If all the three edges of the facet contain crest/ridge points, compute the centroïd of these three points, connect it to the three points by three segments.
  **end**
**end**

**Algorithm 11:** Extract crest/ridge lines on a 3D mesh by the method [OHT 04].

In [KIM 05], the method to connect crest/ridges points is modified in order to make it more robust. There is no more linear interpolation: either the vertex $\mathbf{P_i}$ or $\mathbf{P_j}$ is selected as the crest/ridge point depending on the maximum value of $k_{max}$. Then, for each crest/ridge vertex $\mathbf{P_k}$, we examine all the neighboring vertices. If two or more crest/ridge vertices exist, we select the vertex $\mathbf{P_l}$, such as the angle $\angle(\mathbf{P_k P_l}, \mathbf{t^k_{max}})$ is minimum. Note that the authors use a modified Moving Least-Squares approximation technique in order to reduce the computation time of the differential parameters.

In [HIL 05], the authors describe two methods to add a smoothing step in Algorithm 11 to get more visually pleasing crest/ridge lines.

In the first method, values of the crest/ridge point function $e^i_{max}$ are regularized before approximating the crest/ridge point $\mathbf{P_{ij}}$, by applying a Laplacian smoothing (i.e. the value at a vertex is modified with respect to the values at the neighbor vertices). But the difficulty is being independent of the sign of $e^i_{max}$ which can be inverted just by flipping the corresponding $\mathbf{t^i_{max}}$ vector. The modified Laplacian smoothing is then defined by:

$$e^i_{max} \leftarrow e^i_{max} + \sum_{\mathbf{P_j} \in N(\mathbf{P_i})} (w_{ij} e^j_{max} - e^i_{max})$$

where $w_{ij} = sgn(t^i_{max} \cdot t^j_{max})$, which, in fact, corresponds to the Acute Rule.

The second method consists of smoothing the crest/ridge lines after their extraction. We can use any smoothing scheme such as, for example, the Laplace one (see, for example, [JIN 06]). The problem is then to ensure that the smoothed line does not deviate too much from the original one. The idea is to introduce a constraint based on the Hausdorff distance w.r.t. to the original line.

In [YOS 08], the authors propose to add a step to reduce the fragmentation of the crest/ridge lines to Algorithm 11. The idea is to look in the neighborhood of each crest/ridge line endpoint. If we can find another endpoint belonging to another crest/ridge line, we connect both of them, if the angle between their end segments is relatively small. An implementation of their algorithm which

includes the estimation of differential parameters and the tracing of crest/ridge lines is publicly available as C++ code associated with a Java3D viewer[10].

Another method to compute crest/ridge lines is proposed in [CAZ 05c] and is publicly available in the open-source C++ Computational Geometry Algorithms Library CGAL[11]. The algorithm is based on the topological analysis of the crest/ridge line network described in [CAZ 06]. In particular, in [CAZ 05b], it is emphasized that only either three or one crest/ridge lines can cross a generic umbilic point. It then becomes possible, by assuming some general hypotheses about the triangulation of the 3D discrete mesh, and by detecting umbilic points, to extract the crest/ridge lines in a topologically consistent way. This method can be decomposed into the following steps.

The first step consists of computing the differential parameters for each vertex of the 3D mesh. Even though the proposed method is the one described in [CAZ 05a], we could use any of the algorithms described in section 1.3. The second step aims to detect umbilic areas over the 3D mesh. The idea is to define the patch of neighborhood facets over the considered face. A vertex $\mathbf{P}$ of this patch is an umbilic if $k_1(\mathbf{P}) = k_2(\mathbf{P})$ (see section 1.2.2). As we have differential parameters at some discretized vertices, we never have a strict equality and it requires introducing a threshold $\epsilon$ to decide if $\mathbf{P}$ is a potential umbilic vertex by $|k_1(\mathbf{P}) - k_2(\mathbf{P})| < \epsilon$. Nevertheless, as it is difficult to precisely locate an umbilic which is an isolated singularity point, it is better to detect potential umbilic facets, which are the facets where all vertices are potential umbilic vertices. However, this may be not sufficient to detect umbilic areas with a good level of reliability. In [SAN 92], the authors propose to locally analyze the field of principal directions by using the *index function* defined at a point $\mathbf{P}$ by:

$$index(\mathbf{P}) = \frac{1}{2\pi} \int_0^{2\pi} < \mathbf{t_1}(\mathbf{M}(\theta)), \mathbf{u} > \ \mathbf{d}\theta$$

where $\mathbf{u}$ is a given direction vector, $\mathbf{M}(\theta)$ is a point, parameterized by the angle $\theta$ with respect to $\mathbf{u}$, which follows a small closed counterclockwise trajectory around the point $\mathbf{P}$. The index function is then the integral all around $\mathbf{P}$ of the angle between a principal direction and a given direction. In fact, this function describes the way the lines of curvature turn around the point $\mathbf{P}$, and it is

10 http://www.riken.jp/brict/Yoshizawa/Research/Crest.html.

11 http://doc.cgal.org/latest/Ridges_3/index.html.

proven (see, for example, [CAZ 05b]) that for an umbilic point, the index is either $+\frac{1}{2}$ (the umbilic is then classified as *lemon* or *monstar*) or $-\frac{1}{2}$ (the umbilic is then classified as *star*). Then, if we think that the considered facet is a potential umbilic facet, we approximate its index function by moving $\mathbf{M}$ around the boundary of the patch of its neighborhood facets. We confirm that this facet is an umbilic one if its index is equal to $\pm\frac{1}{2}$ and the neighborhood patch of this umbilic facet becomes an umbilic area. More details about the index function and its computation around a vertex of a 3D discrete mesh can be found in [PAT 10, p. 236].

The authors then propose to compute some third order differential parameters to define if 1 or 3 crest/ridge lines go through the umbilic area, even though there is no detail about how to approximate these parameters on a 3D discrete mesh.

The third step focuses on finding crest/ridge points on edges outside the umbilic areas. It uses the same idea as Algorithm 11: after applying the Acute Rule constraint, test if the crest/ridge function has an opposite sign at the two sides of an edge. If this is the case, compute the position of the crest/ridge point by linear interpolation. In a triangular facet, two crest/ridge points on consecutive edges are linked to form a crest/ridge segment.

The fourth step consists of tagging the crest/ridge segment as elliptic or hyperbolic. In fact, an elliptic crest/ridge point corresponds to a maximum of $k_1$ or a minimum of $k_2$. This is equivalent to the detection of the maxima of $k_{max}$ in the second loop of Algorithm 11, but the authors propose to use not only the two signs of the crest/ridge function on an edge but rather the three signs computed on the vertices of the facet crossed by the segment. This prevents some ambiguities when an edge is almost parallel to a crest/ridge line. At the end, we get a set of tagged crest/ridge segments.

In the last step, by linking all the crest/ridge segments, we get crest/ridge lines. When these lines end at the boundary of an umbilic area, they are connected to the center of the umbilic area in order that they do not cross each other. According to theoretical topological constraints, there must be only one or three connections to perform, but it may be more complex in the case of a discrete 3D mesh.

All these steps lead to the following algorithm.

**Input:** 3D mesh composed of vertices $\mathbf{P_i}$ which is assumed compliant w.r.t. to some general hypotheses
**Output:** Crest/ridge lines as a set of segments
**begin**
  **forall** *vertex* $\mathbf{P_i}$ **do**
    Compute the principal curvatures $k_1(\mathbf{P_i}), k_2(\mathbf{P_i})$ and the associated principal directions $\mathbf{t_1}(\mathbf{P_i}), \mathbf{t_2}(\mathbf{P_i})$.
  **end**
  **forall** *facet* $F_i$ *of the mesh* **do**
    Take iteratively the neighbor facets which centers are within a given radius around the centrod of $F_i$. They define a patch $\mathcal{P}_i$ of facets which has the topology of a disk around $F_i$.
    **forall** *facet* $F_i^j$ *of the patch* $\mathcal{P}_i$ **do**
      Compute $\Delta k(F_i^j)$ as the arithmetic average of the values $k_1(\mathbf{P}) - k_2(\mathbf{P})$ computed at each vertex $\mathbf{P}$ of the facet $F_i^j$.
    **end**
    **if** $F_i = \text{argmin}_{F_i^j} \Delta k(F_i^j)$
    **then**
      Apply the Acute Rule to orient consistently the principal directions of the vertices located on the boundary of the patch $\mathcal{P}_i$.
      By adding the angle deviation between one of the principal direction w.r.t. and a fixed direction, for all the vertices $\mathbf{P_i}$ on the boundary of the patch $\mathcal{P}_i$, estimate the index function $index(F_i)$.
      **if** $index(F_i) = \pm\frac{1}{2}$
      **then**
        By computing some third order differential parameters, detect if 1 or 3 crest/ridge lines cross the facet $F_i$.
        Tag all the facets of the patch $\mathcal{P}_i$ as "umbilic" facets.
      **end**
    **end**
  **end**
  **forall** *edge* $\mathbf{P_i P_j}$ *which does not belong to facet tagged "umbilic"* **do**
    Compute the crest/ridge function $e_i$ and $e_j$ respectively at $\mathbf{P_i}$ and $\mathbf{P_j}$.
    If $e^i \,.\, e^j < 0$ then there is a zero-crossing of the crest/ridge function which corresponds to a crest/ridge point on the edge.
    Approximate the position of the crest/ridge point $\mathbf{P_{ij}}$ by linear interpolation of the positions of $\mathbf{P_i}$ and $\mathbf{P_j}$ weighted by $e^i$ and $e^j$.
  **end**
  **forall** *facet* $F_i$ *of the mesh* **do**
    If 2 crest/ridge points belong to two consecutive edges of $F_i$ (which is assumed triangular), connect them by a straight crest/ridge segment.
  **end**
  **forall** *crest/ridge segment* $s$ **do**
    Find the facet $F_s$ which contains the segment $s$.
    Tag the segment $s$ as "elliptic" or "hyperbolic" according to the signs of the crest/ridge function at the vertices of $F$.
  **end**
  Link all the consecutive segments to form crest/ridge lines.
  When these lines end at the boundary of an umbilic patch, connect them to the center of the umbilic area in order that they do not cross each other.
**end**

**Algorithm 12:** Extract crest/ridge lines on a 3D mesh by the method [CAZ 05c].

All the different methods described above give a set of crest/ridge lines. But in real-word applications, 3D meshes may be distorted, incomplete, rounded at the edges or with a different sampling density. Some resulting crest/ridge lines may then be non-significant as they are too short or too irregular. It is then required to add a post-processing step in order to filter these crest/ridge lines and keep only the significant ones. Several algorithms have been proposed, based on thresholding a parameter computed on the list of the points $\mathbf{P_i}$ of a crest/ridge line:

In [OHT 04], the authors propose to use the *strength*:

$$Strength = \sum_j \|\mathbf{P_{j+1}} - \mathbf{P_j}\| \ \frac{k_{max}(\mathbf{P_j}) + k_{max}(\mathbf{P_{j+1}})}{2}$$

This parameter, which gives the average magnitude of the maximum principal curvature, is scale-independent. We can also find, in [KIM 06], a formula which gives a slightly different parameter:

$$T = \sum_j \|\mathbf{P_{j+1}} - \mathbf{P_j}\|^2 \ k_{max}(\mathbf{P_j})$$

In [CAZ 05c], the authors compute the Taylor expansion around a point $\mathbf{P}$ of $k_1$ in the direction of its associated principal direction $\mathbf{t_1}$. By starting with an extension of the equation [1.35] at order 4, which introduces in particular the term $c_0 x^4$, they obtain:

$$k_1(x) = k_1(\mathbf{P}) + b_0 x + \frac{P_1(\mathbf{P})}{2(k_1(\mathbf{P}) - k_2(\mathbf{P}))} x^2 + O(x^3)$$

with $P_1(\mathbf{P}) = 6b_1^2 + (k_1(\mathbf{P}) - k_2(\mathbf{P}))(c_0 - 3k_1^3(\mathbf{P}))$.

For a crest/ridge point, we have by definition $k'(0) = 0$. But in plane or spherical areas, the principal curvatures are constant and this equality is also true. So, the idea is to study the second derivative $k''(\mathbf{P})$. The higher its value is (in absolute value), the faster the variation of curvature is. This means that for a maximum or a minimum of $k(\mathbf{P})$, the crest/ridge line will be sharp if $\|k''(\mathbf{P})\|$ is high.

By differentiating the previous equation, we get:

$$k_1''(x) = \frac{P_1(\mathbf{P})}{k_1(\mathbf{P}) - k_2(\mathbf{P})}$$

And we can define the *sharpness* of a crest/ridge line as:

$$Sharpness = L^2 \sum_j \| \frac{P_1(\mathbf{P_j})}{k_1(\mathbf{P_j}) - k_2(\mathbf{P_j})} \|$$

By multiplying by $L^2$ where $L$ is the length of the crest/ridge line (or it could more generally be the mesh size as its diameter), we obtain a scale-independent parameter to threshold.

In [YOS 08], the authors compute what they call the *cyclidity* by using two different formulas which give similar results:

$$Cyclidity_1 = \sum_j \|\mathbf{P_j} - \mathbf{P_{j+1}}\| \ \sqrt{|e_{max}(\mathbf{P_i})| + |e_{min}(\mathbf{P_i})|}$$

$$Cyclidity_2 = L \sum_j \|\mathbf{P_j} - \mathbf{P_{j+1}} \| \sqrt{e_{max}^2(\mathbf{P_i}) + e_{min}^2(\mathbf{P_i})}$$

where $L$ is the length of the crest/ridge line and $e_{max}(\mathbf{P})$ (resp. $e_{max}(\mathbf{P})$) is the crest/ridge function for the blue (resp. red) crest/ridge lines (see section 1.4.3).

This parameter is also scale-dependent but as it involves third-order derivatives, it may be more complex to apply.

Some other methods to filter crest/ridge lines are based on a multi-scale analysis. In [SUB 99], the idea is to use two versions of the 3D mesh, the original one $\mathcal{M}$ and a smoothed version $\mathcal{M}_s$ (many smoothing methods exist, we can refer to [TAU 00] for some supplementary information). On the smoothed version $M_s$, we should extract longer and more reliable crest/ridge lines but the positions of their points may be shifted by the displacement of the 3D mesh vertices due to smoothing. Moreover, we can miss some crest/ridge lines emphasizing some small salient parts of the 3D mesh, which may have disappeared after the smoothing step. By registering the two sets of crest/ridge lines of $\mathcal{M}$ and $\mathcal{M}_s$ and keeping the common ones with the

position of points on $M$, we expect to keep the most reliable crest/ridge lines while keeping their original accuracy. Nevertheless, the presented algorithm is quite crude because it only uses two values of the scale parameter that are manually defined. A more sophisticated multi-scale method can be found in [FID 97] where crest/ridge lines are extracted and followed across many scales.

Another method to get more significant results is to link several close short crest/ridge lines in order to get a longer one. In particular, crest/ridge lines are often cut around umbilic areas whereas they should cross them according to theory (see [CAZ 05c]). The first algorithm proposed in [PAG 06] is based on path planning widely used in robotics. The idea to link two crest/ridge short lines $L_1$ and $L_2$ is to define two artificial potential fields. The first field $U_{att}(\mathbf{P})$ will be based on the distance to $L_2$, characterizing the proximity to a vertex of $L_2$. The second field $U_{rep}(\mathbf{P})$ aims to repel or push according to the differential elements of the considered vertex. In our case, we could use, for example, an increasing function of $|e(\mathbf{P})|$. Then, the more this value is, the more the vertex $\mathbf{P}$ can be considered as different from a crest/ridge point where $e$ should be null. Now, if we start from an extremity $\mathbf{P}$ of $L_1$, we compute $U_{att}(\mathbf{P})$ and $U_{rep}(\mathbf{P})$, and we can define the vector $\mathbf{g}$ corresponding to a weighted sum of the gradient vectors of the two fields. $\mathbf{P}+\alpha\mathbf{g}$ will then define a new point in the direction of the minimum path between $L_1$ and $L_2$ with respect to the potential fields. By iterating the process, we can link the two short crest/ridge lines. In [KHA 98], we can find the same idea of finding the optimal line linking two crest/ridge points by minimizing a criterion about the curvature of the 3D mesh. The method is based on dynamic programming but we could also use the very classic Dijkstra's algorithm to find the optimal path.

When detecting crest/ridge lines, the estimation of the differential parameters may be imprecise and may distort the positions of the points of the lines. It is then interesting to smooth these noisy crest/ridge lines. In [JIN 06], the authors propose to use an equivalent of the well-known Laplacian smoothing of 3D mesh: each point of the crest/ridge line will slide on the 3D mesh towards the middle of its previous and next point on the line. Compared to a direct Laplacian smoothing of the whole 3D mesh, this method concentrates more on the local shape of the crest/ridge lines and is more effective. Note that in [HIL 05], it is proposed to use the Hausdorff distance to

guarantee that the smoothed line does not deviate strongly from the initial line, but no detail is given.

In [JIN 06], the authors also propose to remove some crest/ridge branches which could be connected to long crest/ridge lines. For this purpose, they compute a Minimum Spanning Tree which allows them to identify the longest path. It then becomes possible to automatically define a length threshold and to disconnect small parts from the lines.

Finally, note that some methods to compute crest/ridge lines have been proposed when we have no more a 3D mesh but an unorganized 3D cloud of points $\mathbf{P_i}$.

In this case, the authors of [GUM 01] propose to connect all the segments between a 3D point of the cloud and its k-nearest neighbors in order to build a neighbor graph (a more complex method based on Delaunay tetrahedrization is also proposed). Then, for each 3D point, the $3 \times 3$ covariance matrix of the coordinates of the neighbor points is computed. The eigenvalues of this matrix give information about the local shape around the 3D point. In particular, it is possible to estimate a local fitting plane (and then a normal vector) and the curvatures in the direction of the neighbor points. Based on the eigenvalues of the endpoints, we can affect a weight to the edges of the neighbor graph. Then, by extracting a Minimum Spanning Tree (and eventually removing short branches), we can define some feature lines. If we choose a weight characterizing if a point is a crease locally, we can extract crest/ridge lines.

In [MÉR 11], the covariance matrix is based on a 3D Voronoï diagram. The $3 \times 3$ matrix called the Voronoï Covariance Measure is approximated either by a Monte-Carlo algorithm or by a 3D discrete tessellation. All the VCM in the neighborhood of a point can be summed in order to get the convolved Voronoï covariance measure which can provide information about the normal vectors and the differential parameters of the surface which locally underlies the 3D point cloud. It is then possible to extract the so-called sharp edges that correspond to segments of crest/ridge lines. Note that the extraction of the covariance matrix can be multi-scale as proposed in [PAU 03] or [PAR 12].

In [ALT 13], the covariance matrix of the coordinates of the neighbor points of the point $\mathbf{P_i}$ is used to build a local coordinate system where axes $Ox$ and $Oy$ correspond to the principal directions and the axis $Oz$ to the normal vector. By interpolation, it is then possible to compute the height $z$ of the local surface for the points $x_l$ which are regularly sampled along $Ox$. In order to get a robust and continuous representation of the curve $z(x_l)$, we can apply a discrete Fourier transform and remove the coefficients of higher frequency. By taking the curvature of the curve $z(x)$ at 0 (which can be directly derived from the Fourier transform), we get a robust value of one principal curvature (corresponding to the principal direction defining the axis $Ox$) at point $\mathbf{P_i}$. By doing the same for the axis $Oy$, we can get all the differential parameters. Now, by thresholding the Gaussian curvature of the points $\mathbf{P_i}$, we can select potential crest/ridge points. In a second step, an equivalent of Laplacian smoothing is applied to these potential points and when two of them are too close, they are merged. The smoothing process is iterated until the number of potential points remains constant; we can then consider obtaining a set of robust crest/ridge points. In the third step, crest/ridge lines are constructed by a line growing technique. To add a crest/ridge point to a current line, we take the two endpoints of the current line, we build two spheres centered at these endpoints and which radius is proportional to mean distance between a point $\mathbf{P_i}$ and it closest point and we search for the crest/ridge points inside the spheres. If one of these crest/ridge points is roughly aligned with the extremity part of the current line, it is added to the current line and the process can continue. The method has been implemented on GPU, which allows one to process large point clouds.

### 1.4.4. *Feature lines based on homotopic thinning*

In this section, we present a way to characterize feature lines on a mesh using a well-known notion coming from mathematical morphology: the *homotopic thinning* [STE 71, ROS 75, ROS 82]. The feature lines are materialized by a skeleton computed by thinning a vertex set lying on the 3D meshes. The key idea comes down from eroding a 2D set located on a discrete 2-manifold.

The skeleton is a popular and established shape descriptor. It is an entity that is globally centered in a 2D or a 3D object, and it characterizes its

topology and its geometry. This structure is widely used in various applications (video tracking [GAL 09], shape recognition [YU 08], surface sketching [MAR 09], etc.). Several techniques exist in order to extract the skeleton from binary 2D images [ZHA 84], 3D closed volume meshes [AU 08] or 3D cubic grids [LEE 94].

Nevertheless, very few approaches have been dedicated to the extraction of skeletons from binary information located on an arbitrary 3D mesh. It remains to compute the skeleton of a skew subset of a discrete surface embedded in $\mathbb{R}^3$. Rössl *et al.* [RÖS 00b] have presented a first method that uses the elementary *opening* mathematical morphology operator, ported to 3D triangulated meshes. However, the operator definition is not complete and the underlying algorithm presents some issues. Therefore, several drawbacks have been pointed out which mainly lead to unexpectedly disconnected skeletons [KUD 11]. Kudelski *et al.* have later proposed a modified algorithm that produces topologically robust skeletons, by generalizing the notion of *morphological erosion* to arbitrary 3D meshes [KUD 13]. This approach takes a subset lying on a triangulated surface meshes in 3D as an input, and outputs thin lines corresponding to the skeleton obtained by homotopic thinning. The main idea is to transpose the notion of neighborhood from the classical thinning algorithms where the adjacency is constant (e.g. 26-adjacency in digital volumes, 8-adjacency in 2D grids) to the mesh domain where the neighborhood is variable due to the adjacency of each vertex. The authors propose a thinning operator dedicated to irregular meshes in order to extract the skeleton of a vertex set.

The work of Kudelski *et al.* is interesting in the frame of feature line extraction because it carries the idea of homotopic thinning using a generalized adjacency. Non relevant vertices of the subset (topologically speaking, i.e. the *simple vertices*) are removed iteratively. It produces a lineal skeleton composed of initial vertices and edges. This skeletonization is general because it can deals with non-developable surfaces (operations are local, and there is no need to have an $[i, j]$ indexing like in 2D grids). Moreover, the resulting skeleton preserves the topology of the original shape lying on the surface: cycles and Y-junctions are completely preserved, and this is not the case of previously described approaches where umbilic points create discontinuities of the feature lines.

Let $\mathcal{M}$ be an unstructured mesh patch representing an arbitrary manifold surface $\mathcal{S}$, such as $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$. The sets $\mathcal{V}$, $\mathcal{E}$ and $\mathcal{T}$ correspond, respectively, to the vertices, the edges, and the triangles composing $\mathcal{M}$, a piece-wise linear approximation of $\mathcal{S}$. We denote the vertices as $p_i$, with $i \in [0; n[$ and $n = |\mathcal{V}|$ being the total number of vertices of $\mathcal{M}$. The neighborhood $\mathcal{N}$ of a vertex $p_i$ can be defined as follows:

$$\mathcal{N}(p_i) = \{q_j \mid \exists \text{ a pair } (p_i, q_j) \text{ or } (q_j, p_i) \in \mathcal{E}\}. \quad [1.39]$$

In such a case, $m_i = |\mathcal{N}(p_i)|$ represents the total number of neighbors of $p_i$.

### *Definition of the subset $R$*

As we consider obtaining a skeleton of a subset of $\mathcal{M}$, let us now define a binary attribute $F$ on each vertex of $\mathcal{V}$. The set $R \subseteq \mathcal{V}$ is then written as follows:

$$\forall p_i \in R \iff F(p_i) = 1. \quad [1.40]$$

The attribute $F$ may be defined from a previous process such as a manual selection, a thresholding based on geometric properties (triangle area, principal curvatures, etc.) or any binarization process. Then, an edge $e = (p, q)$ belongs to $R$ if and only if $p, q \in R$. Similarly, a triangle $t = (p, q, r)$ belongs to $R$ if and only if $p, q, r \in R$.

In [KUD 13], the main objective was to develop a technique to extract the skeleton of the set $R$ by using a topological thinning based on the mesh connectivity. This skeletonization algorithm consists of an iterative thinning, relying on a classification of each vertex of $R$. The authors proposed four vertex types based on $c(p_i)$, the *complexity* of the vertex $p_i$ defined as:

$$c(p_i) = \sum_{j=0}^{m_i - 1} |F(q_j) - F(q_k)|, \quad [1.41]$$

where $k = j + 1 \bmod m_i$ and $q_j, q_k \in \mathcal{N}(p_i)$.

A vertex $p_i$ is said to be *complex* if and only if $c(p_i) \geq 4$. The set of all *complex* vertices is named $C$. A *complex* vertex $p_i$ thus potentially corresponds

to a part of a skeleton branch if $c(p_i) = 4$, or a connection through several branches if $c(p_i) > 4$.

A vertex $p_i$ is said to be *center* if and only if $\mathcal{N}(p_i) \subseteq R$. The set of all *center* vertices is named $E$.

A vertex $p_i$ is called *disk* if and only if $\exists q_j \in \mathcal{N}(p_i), q_j \in E$ that is a *center*. The set of all *disk* vertices is named $D$. A *disk* vertex corresponds to a *simple* vertex: a point that does not modify the expected skeleton's topology if it is removed (by analogy with *simple points* [BER 96]).

A vertex $p_i$ is marked as *outer* if and only if $F(p_i) = 1$ and $p_i \notin (C \cup D \cup E)$. The set of *outer* vertices is named $O$ and is defined as follows:

$$O = R \setminus (C \cup D \cup E) \quad [1.42]$$

The algorithm then removes one by one all the *disk* vertices that are not converted to a different class after the thinning operator application. Indeed, at each iterative thinning step, a disk vertex may change from one class to another and, as a side-effect, this may lead to potential disconnections during the skeletonization. To counteract this issue, this requires explicitly specifying a verification stage in the algorithm: at each application of the thinning operator, the class of a vertex is recomputed before its deletion. For example, if a *disk* vertex becomes a *complex* vertex, the vertex is not removed.

After applying the skeleton operator until idempotency on $R$, the set of the remaining vertices, corresponding to the final *skeleton*, is called $Sk_R$. During each pass, the skeleton operator removes the boundary *disk* vertices (if they do not change class). The thinning approach can be summarized by Algorithm 13.

Figure 1.8 illustrates the execution of the algorithm. The extracted skeleton is fully connected and faithfully characterizes the topology of $R$.

After obtaining the skeleton $Sk_R$ of $R$, it is possible to remove the smallest branches by running a final and optional *pruning* operation (like in most of the skeleton approaches).

**Input:** Region $R$ defined from a subset of vertices on a mesh
**Output:** Skeleton $Sk_R$ of the region $R$
**begin**
  Repeat until idempotency:
  **forall** *vertex* $p_i \in R$ **do**
    **if** $p_i$ *is a disk vertex* **then**
      Compute the complexity of vertex $p_i$
      **if** *the class of* $p_i$ *does not change* **then**
        Remove $p_i$ from $R$
      **end**
    **end**
  **end**
**end**

**Algorithm 13:** Thinning algorithm.

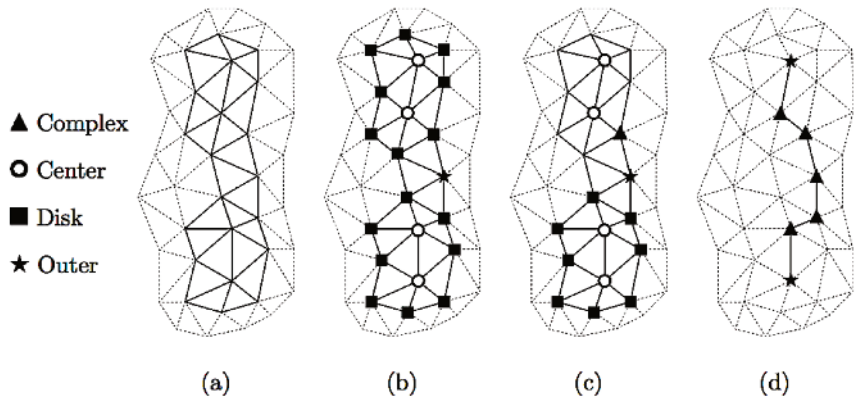



**Figure 1.8.** *Illustration of the homotopic thinning algorithm based on a generalized adjacency (Kudelski* et al. *[KUD 11, KUD 13]): (a) region $R$, (b) vertex classification, (c) execution of the thinning algorithm with update and (d) final skeleton fully connected*

The main asset of this approach is that the obtained skeletons describe the geometry and the topology of the original set $R$. For instance, in Figure 1.9, the algorithm has been tested on irregular meshes to show the robustness of the proposed approach. It can be noted that the resulting skeletons are the expected ones and correctly reflect the topology and geometry of the original set $R$ in a proper way.

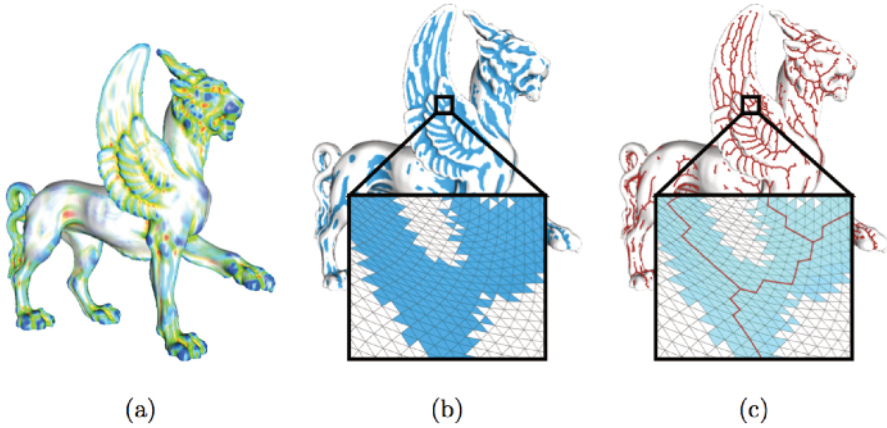



**Figure 1.9.** *Algorithm of feature line extraction: (a) curvature estimation, (b) definition of the set $R$ and (c) extraction of lines from $R$ by Kudelski* et al.*'s thinning approach (courtesy of [KUD 11, KUD 13]). For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Kudelski *et al.*'s approach produces topologically robust skeletons, by generalizing the notion of *morphological erosion* to arbitrary meshes. This method outperforms approaches based on the study of the variation of curvatures (see, for example, [YOS 08]), because there is no need to compute third-order estimators (the noise is thus handled in a better way) and the umbilic points can be processed without cutting the lines. No post-processing is needed to obtain X-junctions on surface meshes, which makes this approach robust, efficient and contrasting.

## 1.5. Region-based approaches

### 1.5.1. *Mesh segmentation*

Segmenting areas on a surface model consists of defining a partitioning of the surface into several regions, according to a specific criterion. Such a criterion can be related not only to the area, curvatures and flatness of a region, but also to semantic aspects of the shape (arm, leg, face of a human body, for example).

Segmenting and detecting feature lines are generally dual problems: once the regions of a segmentation are determined, their boundaries can be

considered as feature lines. Conversely, feature lines can sometimes be assembled so that they bound the regions of a segmentation.

The techniques dealing with mesh segmentation are numerous and varied. Many comparative studies have been published on this topic (see, for example, [ATT 06b, AGA 07, SHA 08a, THE 15, ROD 18]).

Among the different kinds of approaches, region growing is the easier way to handle segmentation on meshes. It consists of an iterative process starting from a given number of points called *seeds* defined on the surface. Vertices of the 3D mesh are then added to a seed region if they are adjacent to this region and if they match a particular criterion, such as curvature [LAV 05] (see Figure 1.10) or flatness [KAL 96].

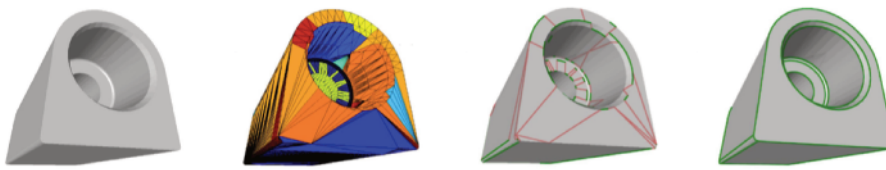


**Figure 1.10.** *Segmenting approach proposed by Lavoué* et al. *The segmentation is processed by a region growing technique constrained by a curvature criterion. The boundaries of the regions are first extracted from the segmented areas, then corrected and completed. Image from [LAV 05]. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Mesh segmentation by partitioning is similar to region growing. Approaches in this category aim at expressing the properties of the elements of the mesh within a common space of representation, in order to define subsets by using a classification algorithm. As always, the choice of the discriminating property is crucial. For instance, it is possible to use membership probabilities [KAT 03] (see Figure 1.11), an approximation with primitives [ATT 06a] or a distance function [SHL 02].

In the domain of image processing, *watersheds* are commonly used in segmentation. This notion has been extended to 3D mesh segmentation. However, the main difficulty remains in defining a relevant height function on a geometric structure, which is not an elevation model. In order to artificially add a scalar value on each vertex of the mesh and thus to use a watershed-like

approach based on heights, the curvature [DEL 06] or the dihedral angle [ZUC 02] can be used.

**Figure 1.11.** *Segmentation based on membership probabilities. Image from [KAT 03]. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

In addition to the aforementioned methods, one family of approaches derives from spectral analysis. They shift the mesh segmentation problem to a matter of planar graph partitioning. The mesh segmentation comes from the eigenvalues of an affinity matrix [LIU 04] (see Figure 1.12) or a matrix of geodesic distances between the vertices of the graph [ZHO 04].

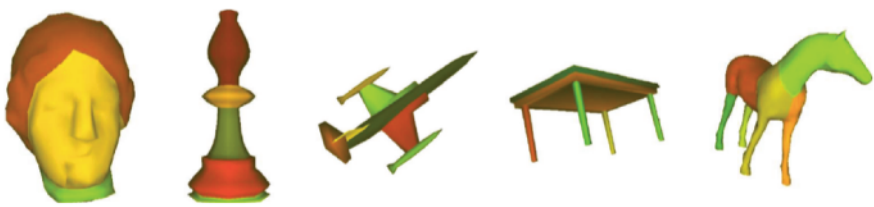


**Figure 1.12.** *Segmentation using spectral analysis. Image from [LIU 04]. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

A curve called *skeleton* can sometimes be used to guide the segmentation process: Shapira *et al.* define a *shape diameter function* that provides a relevant description of the volume of the model, in order to characterize the boundaries between the different parts of the object [SHA 08b].

Some approaches are based on statistics: their aim is to classify data, in order to emphasize the relations that could exist between them. [BEN 11] present a segmentation method based on a heat flow mapping. The starting points are initialized according to the curvature and the connectivity of the

vertices of the mesh. [GOL 08] developed a randomized segmentation approach: starting from the original mesh, several segmentations are performed to produce one final segmented object.

Machine learning techniques are becoming more and more popular, they can also be used to enhance the segmentation process, as soon as a large database of pre-segmented meshes is readily available. For instance, [KAL 10] propose a data-driven approach to simultaneous segmentation and labeling of parts in 3D meshes. An objective function is learned from a collection of pre-labeled training meshes. This function is formulated as a Conditional Random Field model, with terms assessing the consistency of faces with labels.

### 1.5.2. *Shape description based on graphs*

Graphs have been used in order to describe shapes and the layout of sub-shapes mostly within an image, i.e. in 2D. For instance, Damiand *et al.* use combinatorial maps to describe image partitions within a generic segmentation algorithm [DAM 12].

Some recent studies are based on the division of 3D shapes into overlapping regions determined by their regularity properties, such as symmetries [TEV 14]. Then, a graph is formed that connects the pieces with pairwise relations that capture geometric relations between rotation axes and reflection planes. Finally, the authors perform graph matching to establish correspondences.

Graph and sub-graph processing for the analysis of shapes is a known methodology used, for instance, to detect shape symmetries [BER 08], to extract features on points clouds using sub-graphs selection [GUM 01], and to perform mesh segmentation [ZHA 08a]. Approaches using "skeleton graphs" have been developed to achieve global matching of shapes [SUN 03] or to locally match the surface parts using discrete curvature as an invariant descriptor [GAL 06]. We will now describe one of them in detail to give an example.

Polette *et al.* have recently developed an approach that aims to segment a mesh according to several areas with the same "differential type" (using the same classification table with eight categories) [POL 15b, POL 15a, POL 17].

It consists of decomposing the surface into patches, and in constructing an underlying graph describing the shape. Each node is associated with an area of the same type.

The table (see Figure 1.13) described by Besl *et al.* [BES 88b] categorizes a vertex by eight different types according to the signs of the mean curvature $H$ and the Gaussian curvature $K$: *peak*, *ridge*, *saddle ridge*, *minimal*, *saddle valley*, *valley*, *pit* and *flat*. This table is used to tag the areas of a surface.

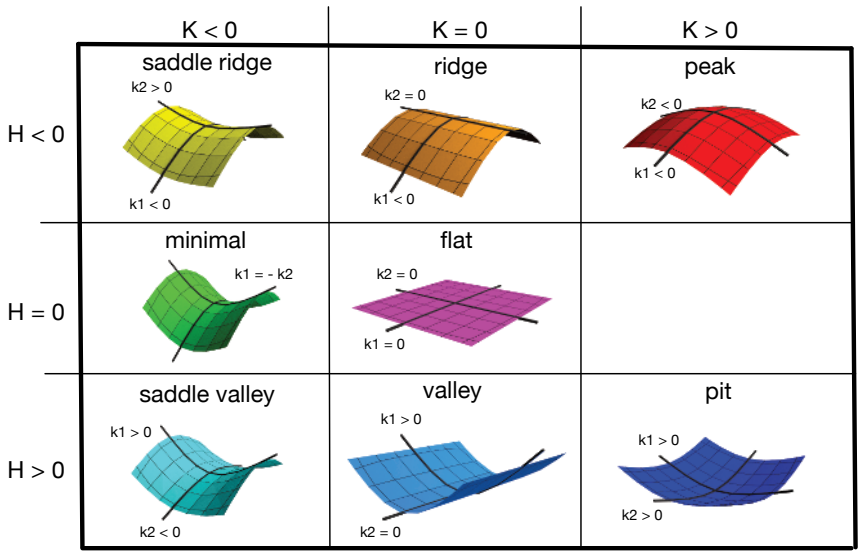



**Figure 1.13.** *Shape categories using mean and Gaussian curvatures. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

However, depending on how a surface is triangulated, the same shape can have several divisions. Thus, it can lead to several different graphs for one similar shape, due to the discrete aspect of the representation by meshes. To tackle this, the main idea is to proceed by analogy with the continuous world: when dealing with a continuous $C^2$ surface, it is not possible to go from a vertex of a bumped area (with Gaussian curvature $K > 0$ and mean curvature $H < 0$) to a saddle vertex (with Gaussian curvature $K < 0$) without passing a $K = 0$ vertex (like a ridge or a valley). Meshes are discrete surface representations, and they can host two adjacent vertices with two non-adjacent types: one peak vertex can be the immediate neighbor of a

saddle or pit vertex. Because this statement cannot apply to continuous surfaces, Polette *et al.* added *intermediary patches* to ensure consistency at the transition between areas.

### *Graph construction*

Figure 1.14 presents an overview of the graph construction method. Starting from a triangulated mesh on which each vertex has been labeled according to its class (among the eight possible categories based on Gaussian and mean curvatures, respectively $K$ and $H$), the shape is then decomposed into patches of the same type. Subsequently, transition areas are defined at sub-vertex level, and the graph is then constructed.

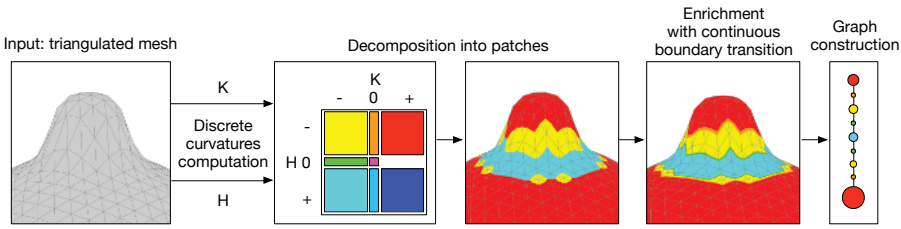



**Figure 1.14.** *Overview of the graph construction methodology. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Polette *et al.* use Meyer *et al.*'s discrete curvature estimator [MEY 03] to obtain the mean curvature $H$ and the Gaussian curvature $K$. It is a robust curvature estimation based on Voronoï cells and a finite-element method.

After this labeling, a mean filter is applied to the discrete surface, in order to smooth and remove irrelevant small areas that do not correspond to feature zones on the shape. A mathematical morphology filter with several distances is employed on the mean and Gaussian curvatures values (see Figure 1.15).

In order to regularize the patches computed on the mesh, because discrete curvatures estimators depend on the sampling, intermediate patches are added to the layout. If the mesh was derived from a continuous object, then it would be impossible to have a peak patch connected to a pit patch without having a transitional patch. Thus, continuity rules are defined to ensure consistency between all the different patches. Figure 1.16 shows, for each patch status,

the compatible patches that can be adjacent. If a yellow area (saddle ridge) is connected to a red area (peak), then there must be an orange area (ridge) in between.

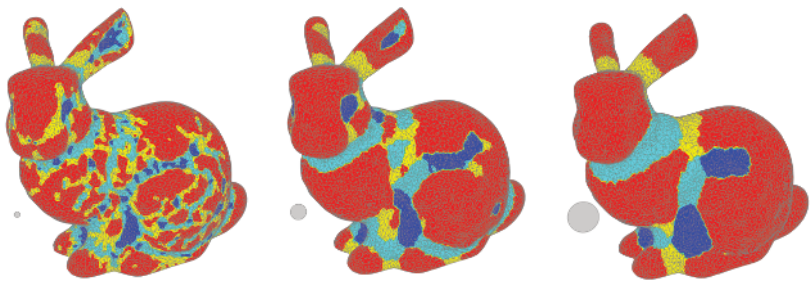


**Figure 1.15.** *Influence of the distance parameter on the bunny mesh (courtesy of the Stanford University Computer Graphics Laboratory); the radius distance is shown as a gray disk. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

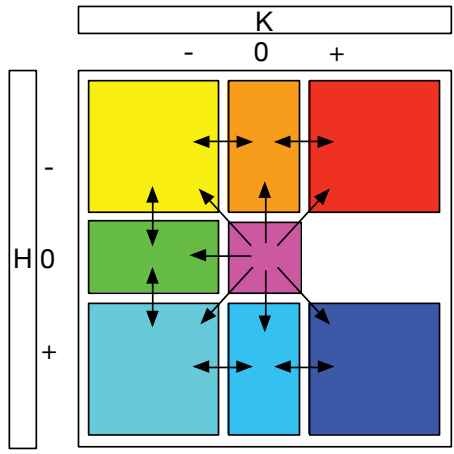



**Figure 1.16.** *Adjacency rules between patches. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

To induce these transition areas, Polette *et al.* use an implicit interpolation of the curvatures between patches to establish the tolerated adjacency for each category. Using these rules, only one shortest path exists between two

patches. Figure 1.17 shows a mesh with the addition of continuous transition boundaries. Each patch between two red patches forms a ring. The yellow parts are thus linked and considered to be a unique area.

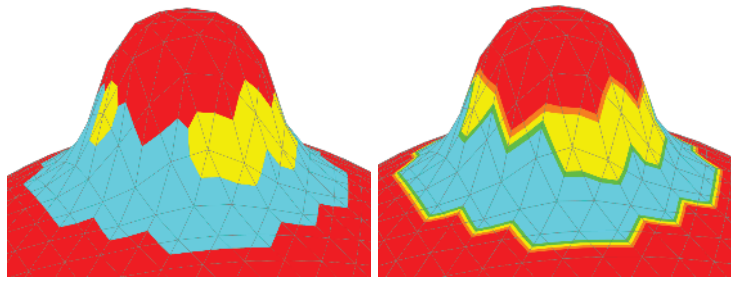


**Figure 1.17.** *Continuous boundary incorporation on a simple bumped mesh. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

The sub-vertex adding approach is illustrated in Figure 1.18. Each triangle of the mesh is checked to ensure that the adjacency rules are respected for all edges. If not, then new areas are added between them and linked to their neighbors. The first two columns describe the addition of missing nodes. Before adding a new node, its existence is checked. If the node does not exist, then a new one is built; otherwise, the existing one is used. Then, links between the additional nodes are created (see columns 2 and 3).

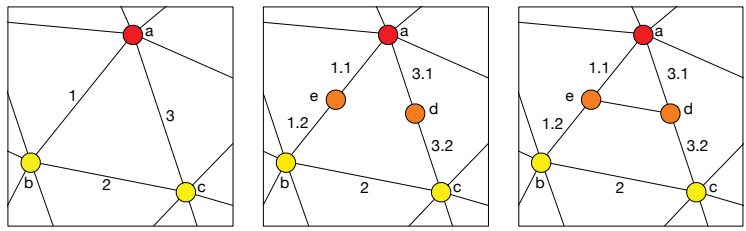
After the sub-vertex areas addition step, a graph is constructed using the patch neighborhood. A propagation algorithm is used to select all contiguous vertices and build a list of patches by category. A node of the graph is defined for each patch. Each node holds the category, the patch area and a link to each neighboring patch (see Figure 1.19).

In Figure 1.20, three meshes from the same shape with different samplings are presented, in order to show how adding the boundary affects the graph's consistency. The sub-vertex step helps to produce three similar graphs, even though the computed curvature areas have different layouts and sizes.

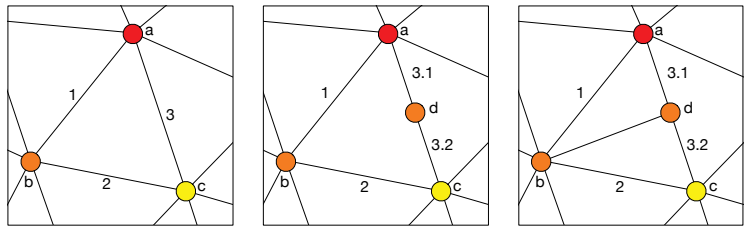
### *Feature extraction*

Polette *et al.* apply the shape description graph defined above to feature extraction. It can be integrated into several applications, like semantic feature

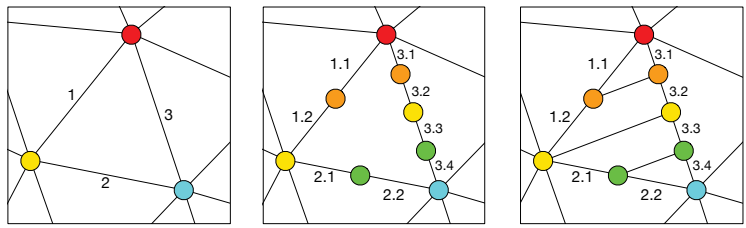
description, similarities extraction between meshes or self-similarities retrieval within one single mesh.




(a) Sub-vertex adding routine with only one additional boundary




(b) Sub-vertex adding routine with an additional boundary linked to an existing node




(c) Sub-vertex adding routine with multiple additional boundaries

**Figure 1.18.** *Continuous boundary adding. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

As a graph can describe a 3D shape, a specific feature within the shape can be described by a sub-graph. Three strategies are proposed by Polette *et al.* to extract features according to different purposes: using a hand-made pattern, using two input meshes or using one single mesh as input to extract self-similarities [POL 15a, POL 17].

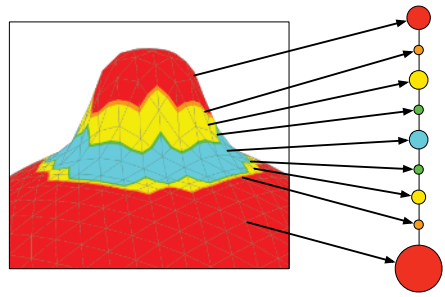


**Figure 1.19.** *Graph construction procedure. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Extracting patterns from a shape descriptor graph consists of a partial sub-graph matching problem. Because the graphs at stake have different node categories and sizes, with specific adjacent rules, Polette *et al.* have developed a specific matching approach in three steps: construction of a similarity matrix $S$, selection of the starting node pairs and recursive node pairing from starting pairs.

The similarity matrix is constructed based on the method described by Nikoli [NIK 12]. This approach is designed to compute the similarity between two entire graphs and Polette *et al.* have adapted it so it can locally compute the similarity between two nodes.

Let us consider two graphs $G_1$ and $G_2$ as inputs defined as $G_1 = (V_1, E_2)$ and $G_2 = (V_2, E_2)$, where $V$ is a set of nodes and $E$ is a set of edges. $|V_1|$ and $|V_2|$ represent the number of nodes in each graph. The *similarity matrix* $S$ is a $|V_1| \cdot |V_2|$ matrix, with $S_{ij}$ the similarity between the nodes $V_{1i}$ and $V_{2i}$, $S_{ij} \in \mathbb{R}$ and $0 \leq S_{ij} \leq 1$. Two identical nodes give a similarity value of 1, and two strictly different nodes give a value of 0. The criterion to construct the matrix is the following: "two nodes $i \in VA$ and $j \in VB$ are considered to be similar if neighbor nodes of $i$ can be matched to similar neighbor nodes of $j$" (see [NIK 12]).

To extract shared sub-graphs, recursive node pairing is operated. Starting pairs are found by testing the similarity value using a threshold $t$. An $ij$ pair is a starting pair if $S_{ij} > t$. Starting from each selected pair, each node of their neighborhood is recursively paired by the maximum similarity value. This pairing function is recursively called on each new pair $ij$ if $S_{ij} > t$. The maximum size of each $ij$ pair of paired sub-graphs is saved in a new $|V_1| * |V_2|$ matrix $M$.

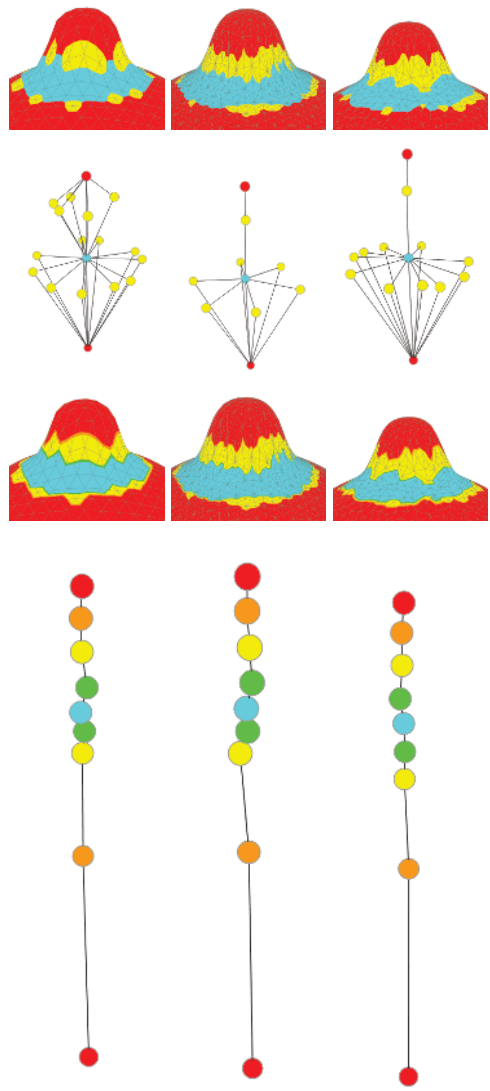



**Figure 1.20.** *Three examples of graph construction with (bottom) and without (top) continuous boundary adding on a similar shape with different sampling. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Maximum similar patterns are extracted using the matrix $M$, beginning with the maximum values of $M_{ij}$. Similar sub-graphs are detected using the same previous recursive method. Corresponding pairs are indexed as a new extracted pattern. Multiple similarities can be detected by propagating this index to all the same maximum values through the rows and columns of each indexed node.

### *Some examples of applications*

Some examples are presented in this section to illustrate the use of Polette *et al.*'s shape descriptor graphs for each scheme.

## Semantic description of a feature

A specific feature can be described semantically via a pattern. Figure 1.21 shows the characterization of a feature on the wing of the *gargoyle*[12] mesh. The sub-graph used in Figure 1.21(d) can semantically describe "a pit bounded by a saddle ridge that can contain one or more peaks, the whole area being bounded by a saddle valley".

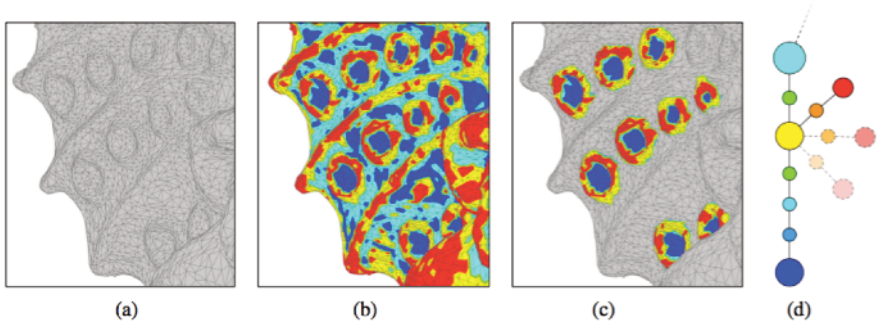



**Figure 1.21.** *Feature extraction by terminal sub-graph recognition: (a) input mesh, (b) mesh partition into patches, (c) extracted features and (d) terminal sub-graph used as a feature descriptor. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

12 Courtesy of the AIM@SHAPE consortium.

In Figure 1.22, Polette *et al.* use this approach to detect lotus flowers in the Buddha mesh[13] (this detection can also be found in [GAL 06]). The yellow part of the flowers and the branches belong naturally to the same node. By limiting the size of patches, the authors can extract only the flowers.

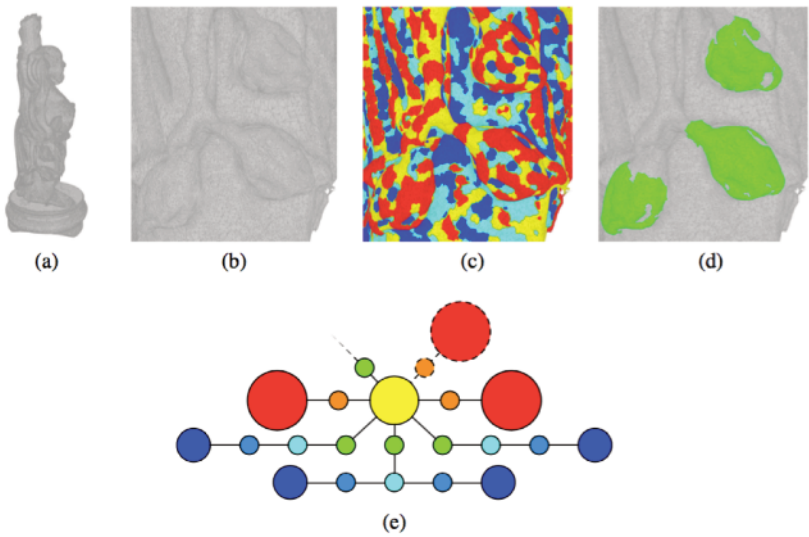



**Figure 1.22.** *Extraction of flowers on (b) the Buddha mesh, (c) categories, (d) extracted parts and (e) the extraction pattern used. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

## Similarity between two meshes

The authors illustrate that this approach can be used to detect similarities between two or more meshes. Figure 1.23 presents two extracted features, a peak and a pit, that can be found on two different meshes.

## Self-similarity within a mesh

Self-similarity within a single mesh can be performed with the shape descriptor graphs by adding an additional constraint in order to avoid the trivial pairing of all nodes to themselves: a node cannot be paired to itself in the final pairing procedure.

13 Courtesy of the Stanford University Computer Graphics Laboratory.


**Figure 1.23.** *Feature detection between two meshes: (a) input meshes and (b) extracted features. For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Figure 1.24 presents an example of self-similarity detection on a mesh. Four maximum sub-graphs are found multiple times; as an additional output, the method gives the sub-graphs that characterize (a) a peak, (b) a cross, (c) a pit and (d) a crater.

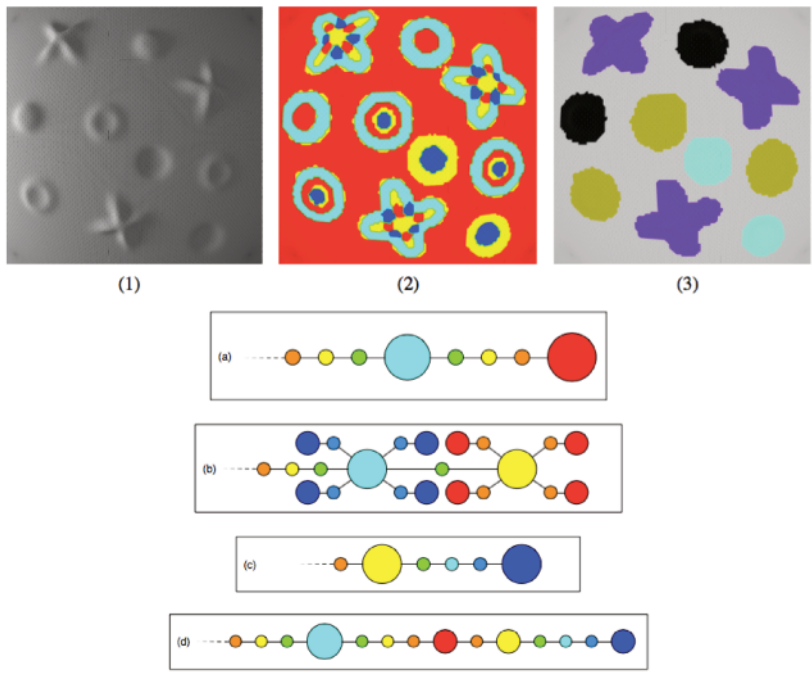



**Figure 1.24.** *Self-similarity extraction on a mesh: (1), (2) and (3) show the input mesh, the computed list of patches and the output classification of sub-graphs. Extracted sub-graphs produced by the approach are presented in (a), (b), (c) and (d). For a color version of this figure, see www.iste.co.uk/mari/analysis.zip*

Along with mesh segmentation and the feature zones detection approach, the method recently developed by Polette *et al.* is interesting for several reasons. It provides a simple graph formalism to describe 3D shapes, based on a curvature map and using a connecting graph. Features can be detected by extracting relevant sub-graphs that correspond to specific patterns. It can be integrated into several applications, such as semantic feature description, similarities extraction or self-similarities detection.

## 1.6. Conclusion

In this chapter, we have presented some geometric features which characterize the shape of a 3D mesh. We focused specifically on features based on differential geometry parameters as they allow us to define not only particular points (umbilics) but also lines (parabolic lines, curvature lines or crest/ridge lines) as well as regions. Moreover, all these features are mathematically related: curvature lines or crest lines end on umbilics, and regions are delimited by parabolic lines. This allows us to define a schematic modeling of the shape of the 3D mesh as it was proposed more than 35 years ago in [HAR 83]. We have seen that this modeling could be represented by a graph. Nevertheless, the discretization of the 3D mesh, and then the approximation of differential parameters, in general makes this graph quite unstable with respect to some small variation of the coordinates of the vertices. It is then interesting to define some global features of the 3D mesh which could stabilize this geometric modeling. In the next chapter, we present some features which are based on topology, which by definition does not change when the shape is continuously deformed.