Curvature in Triangle Meshes

8

Algorithms for effective manipulation and visualization of triangle meshes often require some knowledge about the differential properties of the mesh. A trivial example is the surface normal, which is dealt with in Sect. 8.1. The surface normal is only defined for triangle faces, but when a mesh is drawn as a smooth surface, one generally defines the normal at each vertex and interpolates it across faces. To do so, some technique for normal estimation is required. For many other applications such as shape smoothing, shape analysis, artistic rendering, etc. information about curvature is needed.

This chapter explains how the notion of curvature can be extended to triangle meshes. We will assume that the mesh is a triangle mesh and not a general polygonal mesh. This is not a limitation since any polygonal mesh can be triangulated in order to produce a triangle mesh.

As regards notation: perhaps, it would be more stringent to use different symbols for the discrete versions of differential geometry, say, mean curvature, for which we use H both in the discrete and continuous case. Going further, we could have used various notational devices to distinguish between mean curvature computed using different methods. We have done neither and use H for mean curvature both in the continuous and discrete setting. We believe that it can be seen from the context to what precisely the symbol refers.

Since curvature is not, strictly speaking, defined on meshes, computing curvature from meshes must be approached by one of the following approaches:

- Compute the integral curvature of a small area instead of the pointwise curvature. This is the basic strategy used in the formulas for the computation of the mean curvature normal in Sect. 8.2 and the Gaußian curvature in Sect. 8.3.
- Find a smooth surface that is very close to the triangle mesh: if every edge in a triangle mesh is replaced by a cylindrical patch and every vertex is replaced by a spherical cap, we can compute the curvature of this smooth approximation. This basic strategy is employed in the methods discussed in Sect. 8.4.
- Compute a smooth approximation. This is the classic way of estimating curvature from triangle meshes is based on almost the opposite outlook. The triangle vertices are assumed to be sampled from some smooth surface, and we fit a smooth

J.A. Bærentzen et al., *Guide to Computational Geometry Processing*,
DOI 10.1007/978-1-4471-4075-7_8, © Springer-Verlag London 2012

surface to the vertices in the neighborhood of the point where we wish to find the curvature. This method is discussed in Sect. 8.5.

In Sect. 8.6 we discuss how to compute the directions of principal curvature from the shape operator, which has been discussed in the preceding sections.

Finally, it should be noted that there are many other strategies for estimating curvature than those mentioned here. Section 8.7 provides some pointers to additional literature. The Appendix contains a derivation of the Cotan formula [1] for the mean curvature normal [2].

For references to books on differential geometry as well as an introduction to the fundamental notions, the reader is referred to Chap. 3.

8.1 Estimating the Surface Normal

All curvature measures are basically functions of the derivatives of the normal, and information about the surface normal in itself is used for many purposes, e.g. for shading when rendering meshes. However, we frequently want to compute the surface normal at vertices where the normal is, strictly speaking, not defined. This section explains a sound method for computing what is termed a *pseudo normal* at mesh vertices. By pseudo normal we understand a vector, associated with a point on a triangle mesh, that is imbued with some properties analogous to those of the normal of a smooth surface.

In Chap. 3, we introduced the notion of a parametrization of a smooth surface. A parametrization $S : \mathbb{R}^2 \supseteq U \to \mathbb{R}^3 : (u, v) \mapsto S(u, v)$ is a map from a 2D domain to a smooth surface in 3D. As discussed in Sect. 3.3, the normal to a smooth surface is the cross product of the derivative of S with respect to u and v,

$$\mathbf{n} = \frac{S_u \times S_v}{\|S_u \times S_v\|}.$$

For a triangular mesh, the normal is clearly defined on the faces, where it is just a vector perpendicular to the face, but what do we do at vertices and edges? A starting point is to ask which properties we should require of a vertex normal. An obvious requirement is that the normal should be similar to the normal of a smooth surface, if the mesh is very close to a smooth surface. In other words, we should require that the normal computed for a mesh converges to the normal of a smooth surface, if we refine the mesh (divide the faces into smaller and smaller faces) until it converges to the smooth surface.

If the faces are refined in a sound fashion such that the radii of their circumcircles tend to zero, all the normals around a vertex will tend to the same normal as we refine the mesh, and the average of the face normals of faces incident on a given vertex would satisfy this property. Indeed, a common solution for computing the normal at a vertex is to simply average the normals of the incident faces. However, this is not the best solution. To see why, we consider two additional properties that we need to require of a *good vertex normal*.

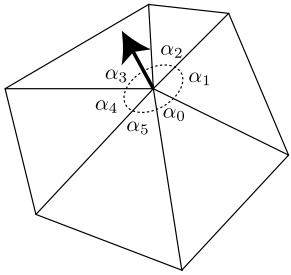

Fig. 8.1 The angle weighted normal is the average of the face normals weighted by the angle α_i which the face makes at the vertex

(1) The normal should be independent of the tessellation and depend only on the geometry of the mesh. Put differently, if one were to add an outgoing edge to a vertex by splitting an incident triangle into two triangles, this should not affect the normal estimate at that vertex.
(2) The normal should allow us to determine whether a point is inside or outside the mesh. This is more subtle, but for an arbitrary point near but *not* on a smooth surface, we know that the normal at the closest point on the nearby smooth surface will point towards that point. A similar property would be desirable for vertex normals in meshes.

Having this in mind, we arrive at the following.

Definition 8.1 The angle weighted pseudo normal AWPN at a vertex is

$$\mathbf{n}_\alpha = \frac{\sum \alpha_i \mathbf{n}_i}{\| \sum \alpha_i \mathbf{n}_i \|}, \tag{8.1}$$

where i runs over all faces incident on the vertex in question.

The AWPN has both these desirable properties. It is almost the same as just averaging the normals of incident faces, but now the normals are weighted by the angle which the face makes at the vertex as illustrated in Fig. 8.1. It is obvious that simply splitting a face does not affect the AWPN since if we split a face by a line going through a vertex, we merely obtain two faces both with the normal of the old face, and the angles sum to the angle of the old face [3].

The AWPN also has the second property. Let $\mathbf{p}_i \in \mathbb{R}^3$ be the point in space associated with vertex i. If we are given a second point $\mathbf{q}$ such that $\mathbf{p}_i$ is the point on the mesh closest to $\mathbf{q}$, we have

$$\mathbf{n}_\alpha \cdot (\mathbf{q} - \mathbf{p}_i) > 0, \tag{8.2}$$

where $\mathbf{n}_\alpha$ is the normal at vertex i, precisely if $\mathbf{q}$ is outside the mesh. Of course, we can insert a vertex on a face or an edge without changing the geometry so the AWPN is defined everywhere: on a face it is simply the face normal. On an edge the AWPN is the average of the normals of the two incident faces. For a proof of (8.2) the reader is referred to [4].

8.2 Estimating the Mean Curvature Normal

In the case of plane curves, curvature is unambiguously defined: it is the rate of change of the unit tangent vector. For surfaces, the situation is more involved, but, as discussed in Chap. 3, we can define the *normal curvature* at a point as the curvature of a *normal section*. A normal section is the intersection of the surface and a plane containing the normal. Rotating the plane around the normal, we obtain a different normal curvature for each angle. For two perpendicular directions we obtain, respectively, the smallest and the greatest normal curvature. These are denoted the minimum and maximum *principal curvatures*. The mean curvature is the average of the principal curvatures.

The mean curvature is also related to the notion of a *minimal surface* [5], i.e., a surface that has the smallest area among all surfaces with the same boundary. If the mean curvature is zero everywhere on a surface, any small local change to the surface will increase the area. Thus, $H = 0$ everywhere on a minimal surface.

To be a bit more precise, imagine a small patch around a point $\mathbf{p}$. We can think of $\mathbf{p}$ as a control point which can be moved freely, and the patch (a small neighborhood of $\mathbf{p}$) will deform smoothly as this happens. Note that the patch is also assumed to join the rest of the surface in a smooth fashion and that the rest of the surface is not affected by the deformation. Let the area of the patch be A. The area A can be seen as a function of the position of $\mathbf{p}$, and, hence, we can define the gradient of A.

The mean curvature is defined as the following limit:

$$2H\mathbf{n} = 2\mathbf{H} = \lim_{A \to 0} \frac{\nabla A}{A}, \tag{8.3}$$

where the vector $\mathbf{H}$ is the *mean curvature normal* and with the condition that the greatest distance between two points on the curve enclosing the area A also tends to zero. This is a slightly different formulation from the one in Sect. 3.7, because we consider the change of the area, A, to be a function of a deformation of the local patch in any direction and not just the normal direction. The salient difference is that (8.3) yields the mean curvature normal, and (3.40) is a formula for the scalar mean curvature.

In the case of discrete surfaces, we have a natural definition of a "local patch", namely the 1-ring of the vertex. This led Desbrun et al. [2] to define the mean curvature normal simply as the gradient of the area of the 1-ring normalized by the area of the 1-ring.

Definition 8.2 The mean curvature normal for a mesh vertex is

$$\begin{aligned} \mathbf{H}(\mathbf{p}_i) &= \frac{1}{2} \frac{\nabla A_i^{\text{1-ring}}}{A_i^{\text{1-ring}}} \\ &= \frac{1}{4A_i^{\text{1-ring}}} \sum_{\mathbf{p}_j \in \mathcal{N}_i} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{p}_i - \mathbf{p}_j), \end{aligned} \tag{8.4}$$

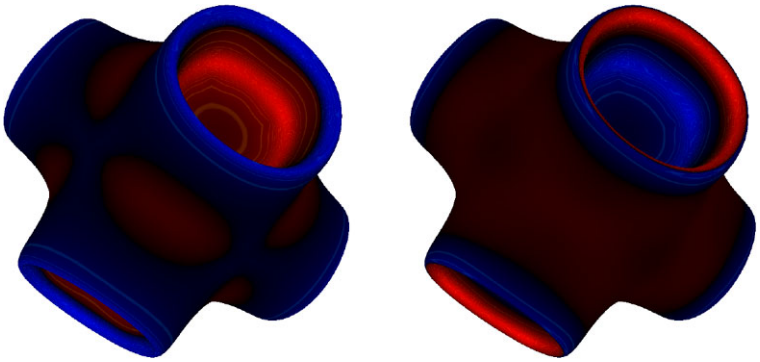

Fig. 8.2 A comparison of mean curvature (*left*) to Gaußian curvature (*right*). *Blue* is positive and *red* is negative, intensity indicates *magnitude* and the *white* stripes are spaced at equal intervals to indicate the rate of change

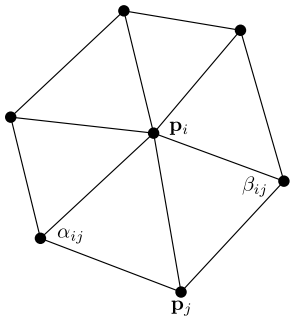

Fig. 8.3 The angles α and β are the angles opposite to edge ij

where $A_i^{\text{1-ring}}$ is the area of the 1-ring. In this equation we sum over all the edges in the 1-ring and weight the edge by the cotangent of the angles at the vertices opposite the edge as illustrated in Fig. 8.3. The details of how to derive this formula are found in the Appendix to this chapter.

Figure 8.2 shows the difference between mean and Gaußian curvature.

A desirable property of the mean curvature normal as defined above is that if the 1-ring is flat, the mean curvature normal is $\mathbf{0}$. In practice this means that the overall size and shape of the triangles are unchanged if the mean curvature normal is used for smoothing as discussed in Sect. 8.7.

It is important to note that the mean curvature normal is really the Laplace–Beltrami operator (cf. (3.53)) applied to the vertex coordinates independently. Thus, (8.4) can be seen as a discrete Laplace–Beltrami operator applied to the vertex coordinate $\mathbf{p}$.

8.3 Estimating Gaußian Curvature using Angle Defects

While mean curvature is the average of the principal curvatures, Gaußian curvature is their product, but we recall from Chap. 3 that there is also a more intuitive way to define Gaußian curvature.

We can construct a closed curve $\mathbf{c} \in S$ around $\mathbf{p}$ in the surface S. This curve will have a corresponding curve $\mathbf{c}'$ in the image of S on the Gauß map. Let the area enclosed by $\mathbf{c}$ be denoted A_S and the area enclosed by the corresponding curve $\mathbf{c}'$ be denoted A_G. The Gaußian curvature is defined as the limit of the ratio of these two areas

$$K = \lim_{A_S \to 0} \frac{A_G}{A_S}, \tag{8.5}$$

with the further condition that the greatest distance between two points on the curve $\mathbf{c}$ also tends to zero (cf. Sect. 3.7).

Gaußian curvature is clearly zero almost anywhere on a triangle mesh. The neighborhood around a point on a *triangle face* is completely flat and maps to a single point on the sphere, and the neighborhood around a point on an edge can be unfolded into a flat surface. However, for a vertex it is rarely so. If the 1-ring around a vertex is to be "flattened", the angles which the incident triangles make at the vertex must sum to 2π. Otherwise, the vertex has a non-zero Gaußian curvature.

To compute the Gaußian curvature, the limit of the areas in (8.5) can be approximated using finite areas. We begin by finding the area on the unit sphere of a curve around the vertex v. Clearly there is a normal for each incident face, and these normals become the vertices of a spherical polygon[1] whose area is

$$A_G(\mathbf{p}_i) = 2\pi - \sum_j \theta_j, \tag{8.6}$$

where θ_j is the angle of face j at the vertex $\mathbf{p}_i$, and j is an index running over all faces in the 1-ring. Equation (8.6) is sometimes called the angle deficit.

The next step is to find the corresponding area on the triangle mesh. Here it is reasonable to divide the area of each triangle among its three vertices; hence

$$A_S(\mathbf{p}_i) = \frac{1}{3} \sum_j A_j,$$

again, index j runs over all triangles in the 1-ring of $\mathbf{p}_i$, and A_j is the area of triangle j in the 1-ring. This leads to the following expression for the Gaußian curvature of a vertex.

[1] A spherical polygon is a polygon on a unit sphere: a polygon whose vertices are points on a unit sphere and whose edges are segments of great circles connecting these vertices.

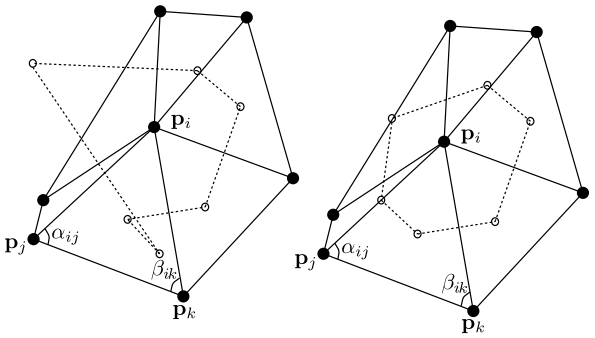

Fig. 8.4 The Voronoi region (*left*) and the region used for computing the mixed area (*right*). The region used for the Voronoi area is not included in the 1-ring in the case of obtuse triangles which is the motivation for the mixed area. For each obtuse triangle, the mixed-area region is obtained by connecting a corner of the region to the middle of the edge opposite the obtuse angle

Definition 8.3 The Gaußian curvature at a vertex of a triangle mesh is

$$K(\mathbf{p}_i) = \frac{A_G(\mathbf{p}_i)}{A_S(\mathbf{p}_i)} = \frac{2\pi - \sum_j \theta_j}{\frac{1}{3}\sum_j A_j}. \tag{8.7}$$

The Gauß–Bonnet theorem (cf. Sect. 3.8) can also be used to justify this formula, see [6] for a very lucid explanation.

Assigning a third of the triangle areas to each triangle is not ideal. We would like to assign to a given vertex the area of the region of the mesh closest to that vertex. In other words, the area of the Voronoi neighborhood [6]. Unfortunately, as observed by Meyer et al. [6], the Voronoi neighborhood is not contained in the 1-ring in the presence of obtuse triangles (see Fig. 8.4). Following [6] the solution is embodied in Algorithm 8.1 for the mixed area where the symbols are shown in Fig. 8.4. This algorithm uses the Voronoi area where possible but restricts the region used for area computation to the 1-ring area. The regions are also non-overlapping and tile the mesh [6].

Algorithm 8.1 Mixed Area of vertex i

1. Let $A_{\text{mix}} = 0$
2. For each face f incident on vertex i:
3. If f is not obtuse then
4. $A_{\text{mix}} = A_{\text{mix}} + \frac{1}{8}(\cot(\alpha_{ij})\|\mathbf{p}_i - \mathbf{p}_j\|^2 + \cot(\beta_{ik})\|\mathbf{p}_i - \mathbf{p}_k\|^2)$
5. Else If the obtuse angle is at vertex i: $A_{\text{mix}} = A_{\text{mix}} + \frac{1}{2}A(f)$
6. Else $A_{\text{mix}} = A_{\text{mix}} + \frac{1}{4}A(f)$

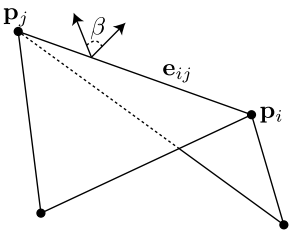

Fig. 8.5 The dihedral angle of an edge is the angle between the two normals sharing that edge

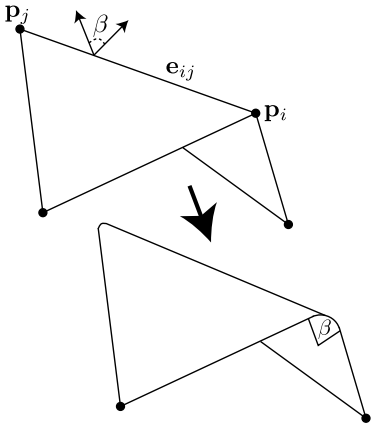

Fig. 8.6 An edge replaced by a cylindrical blend

8.4 Curvature Estimation based on Dihedral Angles

Estimation of both the mean curvature and the shape operator (cf., Sect. 3.3) may be based on the fact that for triangle meshes the surface only bends across edges. If we imagine that a sharp edge is replaced by a cylindrical blend of radius r, the curvature is well-defined on this blend. The curvature is zero in the direction of the edge and $\frac{1}{r}$ in the direction perpendicular to the edge (see Fig. 8.6). At any point on the cylinder, the mean curvature is $H = \frac{1}{2r}$.

The angle β between the face normals on either side of the edge is denoted the *dihedral angle*, see Fig. 8.5. Since the cylindrical blend should meet the faces on either side smoothly (i.e., the cylinder normal should be the same as the face normal where they meet) the slice of the cylinder should also have an angle of β—regardless of the radius. Let $\mathbf{e}$ be a vector in the direction of the edge and having the same length. The area of the cylindrical blend is $\beta r \|\mathbf{e}\|$, and thus the mean curvature integrated over the entire cylindrical blend, B, is

$$\int_B H = \frac{1}{2}\beta\|\mathbf{e}\| \, \mathrm{d}A,$$

regardless of r. Consequently, the formula also holds in the limit as r tends to zero, and it is reasonable to define the integral of the mean curvature as follows.

Definition 8.4 The integral of the mean curvature over the edge is

$$H(\mathbf{e}) = \frac{1}{2}\beta\|\mathbf{e}\|. \tag{8.8}$$

The edge may be concave as well as convex. Assign a positive sign to β if the edge is concave and a negative sign if it is convex.

To integrate the mean curvature over the surface we only need to sum H_e over all edges leading to the following formula for the integral absolute mean curvature:

$$\int_S H \, \mathrm{d}A = \sum_{i=1}^{|\mathcal{E}|} H(\mathbf{e}_i) = \frac{1}{2}\sum_{i=1}^{|\mathcal{E}|} \beta_i \|\mathbf{e}_i\|. \tag{8.9}$$

In practice one often needs the *integral absolute mean curvature* which is defined as follows:

$$\int_S |H| \, \mathrm{d}A = \frac{1}{2}\sum_{i=1}^{|\mathcal{E}|} |\beta_i| \|\mathbf{e}_i\|, \tag{8.10}$$

where the sign of β is ignored. For instance, this curvature measure is sometimes minimized when one wishes to optimize a triangulation. See Sect. 8.7.

It is also possible to define a shape operator integral for edges using the notions above. Instead of the ordinary 2×2 matrix, we will define the shape operator as a 3×3 matrix. This matrix should have an eigenvalue of zero and a corresponding eigenvector pointing in the normal direction. Its two non-zero eigenvectors correspond to the min and max curvature, but its eigenvectors are swapped. In other words, the eigenvector corresponding to the maximum eigenvalue is in the direction of minimum curvature.

For an edge, the 3×3 shape operator is defined as follows:

$$\mathbf{S}(\mathbf{e}) = \frac{\beta \mathbf{e}\mathbf{e}^T}{\|\mathbf{e}\|^2}, \tag{8.11}$$

where β is defined in the same way as above (also with regard to sign). Note that $\mathbf{S}(\mathbf{e})\mathbf{e} = \beta\mathbf{e}$, and $\mathbf{S}(\mathbf{e})\mathbf{v} = \mathbf{0}$ for any vector, $\mathbf{v}$, that is orthogonal to $\mathbf{e}$.

To obtain the shape operator at a given vertex (or, generally, point) on the mesh, a good strategy is to select a region, R, around the vertex. One then sums the shape operators for each edge wholly or partially within R multiplied by the length of the intersection of R and the edge segment. Finally, divide by the area of R to obtain the average for the region. We have the following definition.

Definition 8.5 The shape operator for a point on a triangle mesh is

$$\mathbf{S}(\mathbf{p}) = \frac{\sum_{\mathbf{e}_i \cap R \neq \emptyset} [\text{length}(\mathbf{e}_i \cap R)\beta_i \mathbf{e}_i \mathbf{e}_i^T \frac{1}{\|\mathbf{e}_i\|^2}]}{\text{area}(R)}. \tag{8.12}$$

See [7] for a more rigorous derivation of these formulas based on the theory of normal cycles. Also, a very similar shape operator has been defined by Polthier et al. [8]. If we make the simplifying assumption that all edges are split at their midpoint we obtain the somewhat more readable formula

$$\mathbf{S}(\mathbf{p}_i) = \frac{\sum_{j \in \mathcal{N}_i} [\frac{1}{\|\mathbf{e}_{ij}\|} \beta_{ij} \mathbf{e}_{ij} \mathbf{e}_{ij}^T]}{2A_S(\mathbf{p}_i)}. \tag{8.13}$$

8.5 Fitting a Smooth Surface

In this section, we discuss an alternative way of obtaining the shape operator, this time as a 2×2 matrix.

An obvious way of estimating the curvature of a triangle mesh is based on fitting a smooth surface to the mesh. Finding a global surface that fits the mesh is a difficult problem. Instead, one simply approximates the surface in a local neighborhood of a vertex, $\mathbf{p}_i$. Fortunately, any smooth surface can be represented locally as a height function $f(u, v)$ where u and v are coordinates in an estimated tangent plane with normal $\mathbf{n}$. We require that $f(0, 0) = \mathbf{p}_i$ and that the surface normal of f at $\mathbf{p}_i$ is $\mathbf{n}$. A good surface to use is the paraboloid given by

$$f(u, v) = \frac{1}{2}\left(au^2 + 2buv + cv^2\right). \tag{8.14}$$

If, in fact, $b = 0$ then a and c *are* the principal curvatures. However, this requires that the two axes spanning the tangent plane are aligned with the principal directions which is not the case in general.

The first step is to find the surface normal. Clearly, the mean curvature normal discussed in Sect. 8.2 could be used.

The next obvious question is: how many points to use? The surface has only three degrees of freedom, so only three points are needed. If too many points are used, the result is an overdetermined system, but this system can be solved in the *least squares sense* [9]. In practice, it is probably a good idea to simply use all the neighbors of a vertex. If the triangle mesh is not degenerate there will be at least three neighbors.

Based on the normal, $\mathbf{n}$, we need to compute a pair of vectors $\mathbf{a}$ and $\mathbf{b}$ spanning the tangent plane. We can find the first one by picking a random vector $\tilde{\mathbf{a}}$ (not parallel to $\mathbf{n}$) and by subtracting its projection onto $\mathbf{n}$:

$$\mathbf{a} = \frac{\tilde{\mathbf{a}} - \mathbf{n}(\tilde{\mathbf{a}} \cdot \mathbf{n})}{\|\tilde{\mathbf{a}} - \mathbf{n}(\tilde{\mathbf{a}} \cdot \mathbf{n})\|},$$

and $\mathbf{b} = \mathbf{n} \times \mathbf{a}$. $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{n}$ define a frame in space. We choose to let this frame attach to the point $\mathbf{p}_i$ for which we wish to fit the surface. By construction, this means that the uv-coordinates of $\mathbf{p}_i$ are at $(0, 0)$. For all other points, we find the uv-coordinates in the following way: let $\mathbf{T} = [\mathbf{ab}]$ be a 3×2 matrix whose columns are $\mathbf{a}$ and $\mathbf{b}$.

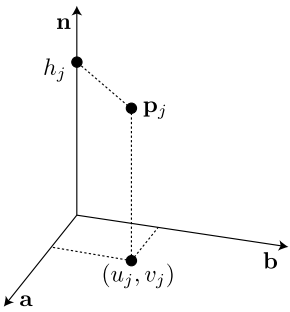

Fig. 8.7 The projection of a vertex into the tangent plane associated with **n**

The uv coordinates are

$$(u_j, v_j) = \mathbf{T}^T(\mathbf{p}_j - \mathbf{p}_i),$$

and the height values are

$$h_j = \mathbf{n} \cdot (\mathbf{p}_j - \mathbf{p}_i).$$

See Fig. 8.7. For all points $\mathbf{p}_j$, we can form the equation

$$h_j = f(u_j, v_j),$$

where the coefficients are unknown. These equations can be rewritten in matrix form $\mathbf{UX} = \mathbf{F}$ where $\mathbf{U}$ is the matrix of parameter values, $\mathbf{X}$ is the vector of coefficients for the polynomial surface, and $\mathbf{F}$ is the vector of height values

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \frac{u_j^2}{2} & u_j v_j & \frac{v_j^2}{2} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ h_j \\ \cdot \\ \cdot \end{bmatrix}. \tag{8.15}$$

We will assume that we have at least three points. In this case, we can solve (8.15) in the least squares sense. The least squares solution is

$$\mathbf{X} = \mathbf{DF} = \left(\left(\mathbf{U}^T\mathbf{U}\right)^{-1}\mathbf{U}^T\right)\mathbf{F}. \tag{8.16}$$

The final step is to estimate the curvature values form the coefficient vector. The shape operator (as a 2×2 matrix) can be computed directly from the derivatives of f.

Definition 8.6 The Shape operator for a mesh vertex (2×2 matrix) is

$$\mathbf{S} = -\frac{1}{\sqrt{1 + f_u^2 + f_v^2}} \begin{bmatrix} 1 + f_u^2 & f_u f_v \\ f_u f_v & 1 + f_v^2 \end{bmatrix}^{-1} \begin{bmatrix} f_{uu} & f_{uv} \\ f_{uv} & f_{vv} \end{bmatrix},$$

where it is understood that all derivatives are evaluated at $(u, v) = (0, 0)$. Fortunately, $f_u(0, 0) = f_v(0, 0) = 0$ so the first matrix is the identity; we have $f_{uu} = a$, $f_{uv} = b$, and $f_{vv} = c$. Consequently,

$$\mathbf{S} = -\begin{bmatrix} a & b \\ b & c \end{bmatrix}. \tag{8.17}$$

From $\mathbf{S}$, the principal curvatures and directions at the point $\mathbf{p}_i$ can be computed as discussed in the next section. It is also straightforward to transform $\mathbf{S}$ into a 3×3 matrix using the matrix $\mathbf{T}$:

$$\mathbf{S}_{3D} = \mathbf{T}\mathbf{S}\mathbf{T}^T.$$

For a more detailed discussion of the method above, the reader is referred to Hamann [10].

8.6 Estimating Principal Curvatures and Directions

Given a shape operator, $\mathbf{S}$, represented as either a 2×2 or 3×3 matrix, the problem of finding the principal curvatures and the corresponding directions (an example of minimum curvature directions is shown in Fig. 8.8) is reduced to finding the eigenvalues and eigenvectors of $\mathbf{S}$. Note that $\mathbf{S}$ is not necessarily symmetric—this depends on the parametrization. However, the methods which we have discussed above both produce symmetric shape operators. This may be important since simple methods for computing eigensolutions rely on the matrix being symmetric.

If $\mathbf{S}$ is represented as a 3×3 matrix, the eigenvector corresponding to the numerically smallest eigenvalue (it should be zero, but it is possible that it is not exactly zero due to numerical issues) is the normal direction. The two remaining eigensolutions correspond to the principal curvatures and directions. If one of the principal curvatures is zero, it is clearly not possible to distinguish between the principal direction and the normal. However, if we know the normal (which can easily be estimated) the cross product of the normal and the first principal direction (eigenvector) will give the second principal direction. If all eigenvalues are 0, the surface is clearly planar and the principal directions are not defined.

If $\mathbf{S}$ is a 2×2 matrix, the eigensolutions simply correspond to the principal curvatures and directions.

Note, though, that if the dihedral angle method from Sect. 8.4 was used to estimate the shape operator, the directions are flipped so that the eigenvector corresponding to the greatest eigenvalue is the direction of minimal curvature.

Once the principal curvatures have been computed, it is easy to find the Gaußian and mean curvatures using the relations

$$H = \frac{1}{2}(\kappa_{\min} + \kappa_{\max}) = \frac{1}{2}\text{trace}(\mathbf{S}),$$

Fig. 8.8 The min curvature directions of the bunny. The directions were estimated using the dihedral angle method from Sect. 8.4. The shape operators at each vertex were averaged with their neighbors to smoothen the field, and then the eigensolutions were found as discussed in Sect. 8.6

and

$$K = \kappa_{\min}\kappa_{\max} = \det(\mathbf{S}).$$

8.7 Discussion

For an interesting comparison of curvature estimation methods see [11]. Surazhky et al. construct meshes from known smooth surfaces and compare the estimated curvature to the analytic.

Clearly, not all techniques for estimating curvature have been discussed. Probably a good starting point for a more in-depth look at the literature is [6] where Meyer et al. propose a systematic framework for estimating curvatures from triangle meshes. Two methods (which were not discussed here) for estimating the shape

operator were proposed by Taubin [12] and by Hildebrandt and Polthier [8]. The latter is similar but not identical to the method discussed in Sect. 8.4. Note also that this is an active area of research, and new methods for computing differential geometric properties of meshes still appear.

The most obvious application of curvature measures on triangle meshes is simply the enhancement of visualization. Small curvature variations may not be obvious from a shaded rendition of 3D model but are easy to detect if the curvature is mapped onto the surface.

A related area is non-photorealistic rendering which is concerned with the rendering of 3D models in an artistic fashion. One artistic style that has been simulated is "hatching" where the artist draws strokes in order to enhance shape and simulate shading. Very often these strokes are aligned with the directions of min or max curvature as discussed in [13].

In some cases, we wish to improve a triangulation while maintaining the vertices unchanged. The most popular way of doing this is by edge flipping. The edge shared by two adjacent triangles can be replaced by a transverse edge. One approach is to loop over all edges and flip if it reduces some energy. This means that some energy function is required, and an effective choice is the integral absolute mean curvature (8.10). See Sect. 11.2 for more details on this method.

There is a large body of literature on this topic of smoothing and removing noise from triangle meshes. A popular method is to move vertices in the opposite direction of the mean curvature normal, i.e. in the direction of $-\mathbf{H}$. This will minimize the surface area leading to a smoother surface. In fact, (8.4) was (re)introduced in the context of surface smoothing by Desbrun et al. [2]. It is worth noting, though, that volume is also reduced which is often a bit of a problem. For more on smoothing see Chap. 9.

In [14] Pierre Alliez et al. proposed a technique for "remeshing" a polygonal mesh to produce a new (quad-dominant) mesh whose edges would be aligned with the min and max curvature directions. This method was based on the shape operator discussed in Sect. 8.4. Since this paper a number of other solutions to the same problem have appeared, e.g. [15].

8.8 Exercises

Exercise 8.1 Write a program which computes the Gaußian of all vertices. Visualize the values as a scalar field defined on the surface.

[GEL Users] GEL provides a tool for visualization of scalar fields on meshes.

Exercise 8.2 Write a program which computes the mean curvature normal at all vertices. Visualize the values as a scalar field defined on the surface. Since the mean curvature normal is a vector, it must be converted to a scalar. Take the dot product of the angle weighted normal and the mean curvature normal at each vertex. The mean curvature is the sign of this dot product times the length of the mean curvature normal.

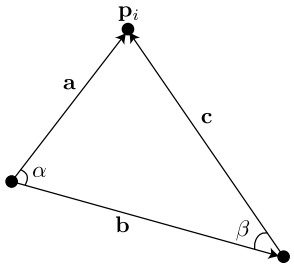

Fig. 8.9 A triangle with a vertex $\mathbf{p}_i$

Exercise 8.3 Compute the integral of the Gaußian curvature over a mesh of known Euler characteristic, χ. Verify that the integral corresponds almost exactly to $2\pi\chi$.

Exercise 8.4 Compute the shape operator for each vertex of a mesh. Obtain the principal curvature directions and visualize those.

[GEL Users] GEL provides a line field visualization tool.

Appendix

In this appendix, we derive the Cotan formula for the gradient of the area of a triangle. Given a triangle (shown in Fig. 8.9) whose vertex $\mathbf{p}_i$ is movable, compute the gradient of the area of the triangle as a function of $\mathbf{p}_i$.

It is clear that the gradient is perpendicular to the plane of the triangle since moving $\mathbf{p}_i$ either in the positive or negative direction along the triangle normal will increase the area. Hence, the present position is a minimum. Moving $\mathbf{p}_i$ parallel to the base line $\mathbf{b}$ will not change the area. It follows that the gradient is in the plane of the triangle and orthogonal to $\mathbf{b}$. It is trivial to find the length of the gradient, and this leads to the first line of the equation below. After a number of steps, we reach the expression used in (8.4).

Some of the steps may be a little tricky. The bottom line is that we need to use the fact that the cotangent of an angle between two vectors $\mathbf{a}$ and $\mathbf{b}$ is equal to $\frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\times\mathbf{b}\|}$:

$$\begin{aligned}
\nabla A(\mathbf{p}_i) &= \frac{(\mathbf{b}\times\mathbf{a})\times\mathbf{b}}{2\|\mathbf{b}\times\mathbf{a}\|} \\
&= \frac{(\mathbf{b}^t\mathbf{b})\mathbf{a} - (\mathbf{b}^t\mathbf{a})\mathbf{b}}{2\|\mathbf{b}\times\mathbf{a}\|} \\
&= \frac{(\mathbf{b}^t\mathbf{b})\mathbf{a} - (\mathbf{b}^t\mathbf{a})\mathbf{a} + (\mathbf{b}^t\mathbf{a})\mathbf{a} - (\mathbf{b}^t\mathbf{a})\mathbf{b}}{2\|\mathbf{b}\times\mathbf{a}\|} \\
&= \frac{-(\mathbf{c}^t\mathbf{b})\mathbf{a}}{2\|\mathbf{c}\times-\mathbf{b}\|} + \frac{(\mathbf{b}^t\mathbf{a})\mathbf{c}}{2\|\mathbf{b}\times\mathbf{a}\|}
\end{aligned}$$

$$= \frac{(\mathbf{c}^t - \mathbf{b})\mathbf{a}}{2\|\mathbf{c} \times -\mathbf{b}\|} + \frac{(\mathbf{b}^t \mathbf{a})\mathbf{c}}{2\|\mathbf{b} \times \mathbf{a}\|}$$

$$= \frac{1}{2}(\mathbf{a}\cot\beta + \mathbf{c}\cot\alpha).$$

References

1. Pinkall, U., Polthier, K.: Computing discrete minimal surfaces and their conjugates. Exp. Math. **2**(1), 15–36 (1993)
2. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99, pp. 317–324. ACM Press, New York (1999). doi:10.1145/311535.311576
3. Thürmer, G., Wüthrich, C.A.: Computing vertex normals from polygonal facets. J. Graph. Tools **3**(1), 43–46 (1998)
4. Bærentzen, J., Aanæs, H.: Signed distance computation using the angle weighted pseudo-normal. IEEE Trans. Vis. Comput. Graph. **11**(3), 243–253 (2005)
5. do Carmo, M.P.: Differential Geometry of Curves and Surfaces. Prentice-Hall, Englewood Cliffs (1976)
6. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege, H.-C., Polthier, K. (eds.) Visualization and Mathematics III, pp. 35–57. Springer, Heidelberg (2003)
7. Cohen-Steiner, D., Morvan, J.-M.: Restricted Delaunay triangulations and normal cycle. In: Proceedings of the Nineteenth Annual Symposium on Computational Geometry, SCG '03, pp. 312–321. ACM Press, New York (2003). doi:10.1145/777792.777839
8. Hildebrandt, K., Polthier, K.: Anisotropic filtering of non-linear surface features. Comput. Graph. Forum **23**(3), 391–400 (2004)
9. Golub, G.H., van Loan, C.F.: Matrix Computations, 3rd edn. John Hopkins, Baltimore (1996)
10. Hamann, B.: Curvature approximation for triangulated surfaces. In: Geometric Modelling, pp. 139–153. Springer, London (1993)
11. Surazhsky, T., Magid, E., Soldea, O., Elber, G., Rivlin, E.: A comparison of Gaussian and mean curvatures triangular meshes. In: Proceedings of IEEE International Automation (ICRA2003), Taipei, Taiwan, 14–19 September, pp. 1021–1026 (2003)
12. Taubin, G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In: Proceedings of the Fifth International Conference on Computer Vision, ICCV '95, p. 902. IEEE Comput. Soc., Washington (1995)
13. Hertzmann, A., Zorin, D.: Illustrating smooth surfaces. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '00, pp. 517–526. ACM, New York (2000). doi:10.1145/344779.345074
14. Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M.: Anisotropic polygonal remeshing. ACM Trans. Graph. **22**(3), 485–493 (2003). doi:10.1145/882262.882296
15. Kälberer, F., Nieser, M., Polthier, K.: QuadCover-Surface Parameterization using Branched Coverings. Comput. Graph. Forum **26**(3), 375–384 (2007)