Computing Curvature
CS468 Lecture 8 Notes

Scribe: Andy Nguyen

April 24, 2013

1 Motivation

In the previous lecture we developed the notion of the curvature of a surface, showing a bunch of nice theoretical properties. But aside from theoretical elegance, what's so great about curvature? Well, remember from the plane curve setting that the curvature function uniquely defines a curve parameterized by arc length up to a rigid motion. A similar result holds for surfaces, though the proof is beyond the scope of this course: If you know the curvature in every direction at every point on the surface, then you've encoded all of the geometry of the surface. Recall that the curvature of a surface at a point can be fully encoded in two scalars (maximum and minimum curvature) and two directions (principal curvature directions); this means that this small list of values encodes all the geometry in a format that is meaningful to manipulate directly.

One application is using curvature as a descriptor: Recall that we derived two scalars from the maximum and minimum curvature, namely Gaussian curvature (the product) and mean curvature (the arithmetic mean). Both of these have intuitive interpretations: Gaussian curvature is positive when the surface is parabolic (convex/concave), and negative when it is hyperbolic (saddle-shaped); while mean curvature describes the extent to which a surface bends. Another application is using curvature as an alternate representation of a surface in which to perform operations such as smoothing before converting back into Euclidean space (though be warned, this back-conversion is highly non-trivial) [9]. Alternatively, we can perform smoothing directly on the surface by minimizing curvature subject to keeping the surface “close" to the original, since curvature is a measure of the deviation of the surface from “flatness", which is the extreme of smoothness [10].

While these applications thus far have focused on the curvature values, we can also find applications for the curvature directions. For one, these principal curvature directions can be used to trace out principal curves, which tend to follow primary geometric curvatures; as a result, we can use these principle curves to create highlights in stylized renderings [2]. For another, since we know that the principle curvature directions are orthogonal and lie in the tangent plane, we can use these directions as a grid of sorts on a surface, which we can use to guide remeshing, a particularly nasty problem because it involves both combinatorial changes (changing the mesh structure) and continuous changes (changing the vertex locations) [1].

2 First Attempt

Now that we've convinced ourselves that we can do a lot of things with curvature, how do we go about computing it on a discrete mesh? Let's start with the seminal paper on the topic [8].

Let's define the Taubin matrix M as a function of position on a surface S as follows:

$$M = \frac{1}{2\pi}\int_{-\pi}^{\pi} \kappa_\theta T_\theta T_\theta^T \, d\theta$$

where

$$\kappa_\theta = \kappa_1 \cos^2\theta + \kappa_2 \sin^2\theta$$

$$T_\theta = T_1 \cos\theta + T_2 \sin\theta$$

Note that $T_\theta T_\theta^T$ is an outer product, giving us a matrix as output. This matrix is useful because we can recover the principal curvature directions as eigenvectors and the principal curvature values from the eigenvalues. First we observe that T_θ is always in the tangent space, so the normal vector N must be an eigenvector with eigenvalue 0. We can therefore factorize M in terms of the tangent space, as follows:

$$M = T_{12}^T \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} T_{12}$$

where $T_{12} = [T_1, T_2]$. Now let's solve for the m's above. We know that $m_{12} = m_{21}$ by symmetry. We first solve for this value:

$$\begin{aligned} m_{12} &= T_1^T M T_2 \\ &= \frac{\kappa_1}{2\pi}\int_{-\pi}^{\pi} \cos^3(\theta)\sin(\theta)d\theta + \frac{\kappa_2}{2\pi}\int_{-\pi}^{\pi} \cos(\theta)\sin^3(\theta)d\theta \\ &= 0 \end{aligned}$$

since both integrands are odd functions. This means our matrix of m's is diagonal, so the two remaining eigenvectors are T_1 and T_2. We can now solve for their corresponding eigenvalues by finding m_{11} and m_{22}:

$$\begin{aligned} m_{11} &= T_1^T M T_1 \\ &= \frac{\kappa_1}{2\pi}\int_{-\pi}^{\pi} \cos^4(\theta)d\theta + \frac{\kappa_2}{2\pi}\int_{-\pi}^{\pi} \cos^2(\theta)\sin^2(\theta)d\theta \\ &= \frac{3}{8}\kappa_1 + \frac{1}{8}\kappa_2 \\ m_{22} &= T_2^T M T_2 \\ &= \frac{\kappa_1}{2\pi}\int_{-\pi}^{\pi} \cos^2(\theta)\sin^2(\theta)d\theta + \frac{\kappa_2}{2\pi}\int_{-\pi}^{\pi} \sin^4(\theta)d\theta \\ &= \frac{1}{8}\kappa_1 + \frac{3}{8}\kappa_2 \end{aligned}$$

We can approximate the Taubin matrix for a vertex v_i on a mesh by replacing the integral with a weighted sum over the neighbors of v_i:

$$\tilde{M}_{v_i} = \sum_{v_j \; v_i} w_{ij}\kappa_{ij} T_{ij} T_{ij}^T$$

Here, we choose the weight of an edge to be proportional to the sum of the areas of the two incident triangles, and we use a divided difference approximation for the curvature along the edge (namely, we divide the difference between the vertex normals by the edge length).

What's wrong with this approach? It turns out that this formulation is really vulnerable to local noise (Figure 1). Remember that curvature is a second derivative of sorts, which means that it amplifies noise. We might be tempted to try smoothing the mesh before doing this computation, but unfortunately, as we mentioned earlier, smoothing is frequently reliant on computing curvature (for example, mean curvature flow), which results in a chicken-and-egg problem.

This problem in general has fueled a lot of research, often tailored to specific applications. This research basically takes the form of a lot of math to derive some formula from which we can extract curvature, all of which actually serves to justify a discrete computation that turns out to work well in practice, at least for the application that motivated the paper.

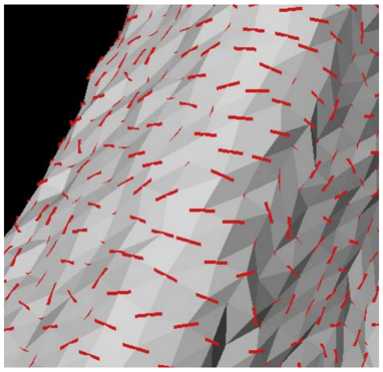

Figure 1: Taubin's approximation is noisy on noisy meshes.

3 Divided Difference Approach

So how can we find an estimate of curvature that's more robust against noise? Well, we can observe that if derivatives amplify noise, then integrating should mitigate it. In other words, every time we introduce an averaging step to our computation, we reduce the effect of noise on our final answer. This is the approach taken in [7], which we'll look at now.

We can write the Second Fundamental Form in matrix form by choosing any two orthogonal vectors $\vec{u}$ and $\vec{v}$ and then doing the following:

$$\begin{aligned}\mathbf{II} &= \begin{pmatrix} D_u n & D_v n \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial n}{\partial u}\cdot\vec{u} & \frac{\partial n}{\partial v}\cdot\vec{u} \\ \frac{\partial n}{\partial u}\cdot\vec{v} & \frac{\partial n}{\partial v}\cdot\vec{v} \end{pmatrix}\end{aligned}$$

From this formulation, we can recover the directional derivative of the normal in any direction $s = c_1\vec{u} + c_2\vec{v}$ simply using matrix multiplication:

$$\mathbf{II}\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = D_s n$$

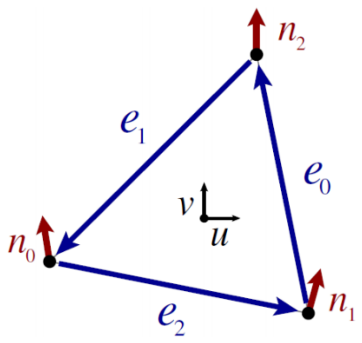

Figure 2: Setup for computing the Second Fundamental Form on a triangle.

Now let's consider a single triangle as in Figure 2, and imagine what the Second Fundamental Form should be for this triangle. We know what the right hand side is for each of the edges of the triangle, so for each edge we can write an equation that our Second Fundamental Form matrix must satisfy:

$$\mathbf{II}\begin{pmatrix} e_0 \cdot u \\ e_0 \cdot v \end{pmatrix} = \begin{pmatrix} (n_2 - n_1) \cdot u \\ (n_2 - n_1) \cdot v \end{pmatrix}$$
$$\mathbf{II}\begin{pmatrix} e_1 \cdot u \\ e_1 \cdot v \end{pmatrix} = \begin{pmatrix} (n_0 - n_2) \cdot u \\ (n_0 - n_2) \cdot v \end{pmatrix}$$
$$\mathbf{II}\begin{pmatrix} e_2 \cdot u \\ e_2 \cdot v \end{pmatrix} = \begin{pmatrix} (n_1 - n_0) \cdot u \\ (n_1 - n_0) \cdot v \end{pmatrix}$$

But notice that we have 6 equations with only 4 unknowns, so we're overconstrained. This is a good thing, because it means we can use least squares when solving for this matrix, which means we're performing some averaging at this step, which should make our answer a bit more robust. Now that we've computed something for each triangle, we can find a Second Fundamental Form matrix for each vertex by taking a Voronoi-weighted average of these matrices (realigned to each other by rotating the tangent plane about the cross product of the normals). From there we can compute our curvature values and directions directly.

Notice that we now have two averaging steps: the least-squares solve for each triangle, and the weighted average around each vertex. These averaging steps combined give us a much more robust computation of curvature than Taubin's approximation. Naturally, there's a cost for this robustness: All of these averaging steps give no guarantees whatsoever that the numbers we get out will have any of the theoretical properties of curvature. However, this isn't a problem if we just want something that "looks like" curvature and is resistant to noise, such as in graphics applications.

4 Conserved Quantity Approach

Sometimes, we know some theoretical property of curvature that we want to hold exactly for our application, and general robustness is less important than the preservation of this property. If this is the case, then we can create our own definition of discrete curvature, choosing it in such a way that we can show the property will hold exactly. This is the strategy employed in [6], which defines a discrete version of Gaussian curvature that preserves the Gauss-Bonnet Theorem, and a discrete version of mean curvature that matches the gradient of the area functional of the mesh.

The Gauss-Bonnet Theorem, which we present here without proof, states that for any two-dimensional manifold M,

$$\int_M K dA + \int_{\partial M} k_g ds = 2\pi\chi(M)$$

In other words, if we integrate the Gaussian curvature over the surface and add it to the geodesic curvature of the boundary of the surface, we get 2π times the Euler characteristic of the surface. This is an interesting observation because it connects local values to the global topology of the surface.

Now let's apply this theorem to a Voronoi cell of a single vertex on a mesh (Figure 3). We can write the following:

$$\int_V K dA = 2\pi - \sum_j \varepsilon_j$$

The last term is the geodesic curvature of the boundary of the cell. This is such a clean expression because when we cross over an edge while following this boundary, our change in motion is in the normal direction, so it doesn't contribute to the geodesic curvature. That means all of the geodesic curvatures comes from the bends in the interiors of triangles, which is easy to compute (as we learned in Lecture 3) because the triangles are flat. As a result, the geodesic curvature is just the sum of the turning angles of the cell. Even better, for Voronoi cells, we can show (and in fact, we'll have to as a homework problem) that instead of summing the turning angles, we can just sum the interior angles of the triangles around the vertex:

$$\int_V K dA = 2\pi - \sum_j \theta_j$$

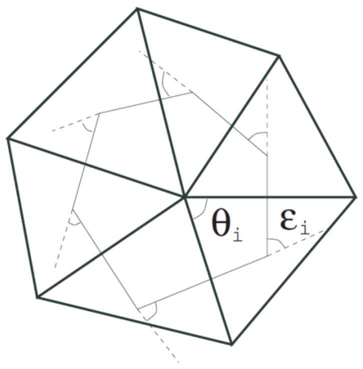

Figure 3: Applying the Gauss-Bonnet Theorem to a Voronoi cell.

Now, we're going to define discrete Gaussian curvature as the quantity that satisfies this equation, namely, that the discrete Gaussian curvature integrated over the Voronoi cell of the vertex is given by the equation above. Then, in order to obtain an estimate of the Gaussian curvature at the vertex itself, we simply divide by the area of the Voronoi cell.

Let's verify that using this definition, we can derive a discrete version of the Gauss-Bonnet Theorem for triangle meshes. We have

$$\begin{aligned}\int_M KdA &= \sum_i \int_{V_i} KdA \\ &= \sum_i \left(2\pi - \sum_j \theta_{ij}\right) \\ &= 2\pi V - \sum_{ij} \theta_{ij} \\ &= 2\pi V - \pi F \\ &= \pi(2V - F) \\ &= 2\pi\chi\end{aligned}$$

The last step comes from the fact that all faces in our mesh are triangles, so $2E = 3F$. Plugging this into the formula for the Euler characteristic $\chi = V - E + F$, we have $\chi = V - \frac{1}{2}F$. So this gives us a formula for computing a value on a mesh that behaves like Gaussian curvature on a smooth surface in the sense that it satisfies the Gauss-Bonnet Theorem.

Now, let's move on to mean curvature, since we need two curvature measurements in order to characterize a surface. In a previous homework assignment, we saw that the curvature vector of a curve points in the direction that moving the point would decrease the length of the curve the fastest. In other words, the curvature vector points opposite the gradient of the area functional of the curve. It turns out (refer to Adrian's Lecture 7 Supplement for details) that mean curvature behaves similarly for a surface. So again, let's define discrete mean curvature to be tied to the gradient of the area functional of a mesh, where we've fixed the topology of the mesh and are only allowed to change the locations of the vertices. First, we need to find the gradient of the area functional of the mesh. To do this, we look at one triangle at a time, and look at the gradient of the area as we vary a single vertex $\vec{p}$ of the triangle. We choose a convenient basis for our computation, namely, one basis vector $\vec{e}$ parallel to the edge opposite $\vec{p}$, one basis vector $\vec{e_\perp}$ in the same plane as the triangle but perpendicular to $\vec{e}$ and pointing towards $\vec{p}$, and one basis vector $\vec{n} = \vec{e} \times \vec{e_\perp}$. Using this basis, we can write a formula for the gradient of the triangle as follows (Figure 4):

$$\vec{p} = p_n\vec{n} + p_e\vec{e} + p_\perp\vec{e_\perp}$$

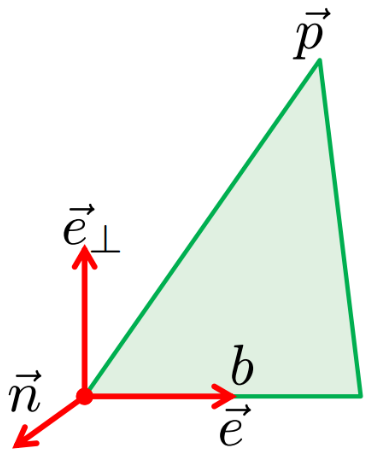

Figure 4: Setup for deriving the area gradient.

$$A = \frac{1}{2} b \sqrt{p_n^2 + p_\perp^2}$$

$$\frac{\partial A}{\partial p_e} = 0$$

$$\frac{\partial A}{\partial p_n} = \frac{b p_n}{2\sqrt{p_n^2 + p_\perp^2}} = 0$$

$$\frac{\partial A}{\partial p_\perp} = \frac{b p_\perp}{2\sqrt{p_n^2 + p_\perp^2}} = \frac{b p_\perp}{2\sqrt{p_\perp^2}} = \frac{b}{2}$$

where the last two lines follow from the fact that $p_n = 0$ by our choice of basis. Therefore we have

$$\nabla_{\vec{p}} A = \frac{1}{2} b \vec{e_\perp}$$

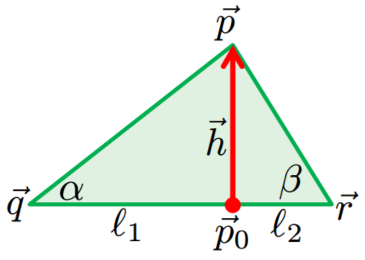

Figure 5: Setup for deriving the cotan formulation of the area gradient.

So the gradient points entirely in the direction of the basis vector $\vec{e_\perp}$. Now let's rewrite this entirely in terms of the vertices of the triangle (Figure 5). We project point $\vec{p}$ onto the base of the triangle to obtain a point $\vec{p}_0$ which is a convex combination of $\vec{q}$ and $\vec{r}$. If we define $\vec{h} = \vec{p} - \vec{p}_0$ and write $vecp_0 = t\vec{r} + (1-t)\vec{q}$, then we can observe

$$||\vec{h}|| = \ell_1 \tan\alpha = \ell_2 \tan\beta$$

Plugging this in, we can solve for t:

$$\begin{aligned} t &= \frac{\ell_1}{\ell_1+\ell_2} \\ &= \frac{\ell_1}{\ell_1+\ell_1\frac{\tan\alpha}{\tan\beta}} \\ &= \frac{\tan\beta}{\tan\alpha+\tan\beta} \end{aligned}$$

Plugging this back into our expression for $\vec{p}_0$, we get

$$\vec{p}_0 = \frac{1}{\tan\alpha+\tan\beta}(\vec{r}\tan\beta+\vec{q}\tan\alpha)$$

and so we have

$$\begin{aligned} \vec{h} &= \frac{1}{\tan\alpha+\tan\beta}((\tan\alpha+\tan\beta)\vec{p}-\vec{r}\tan\beta-\vec{q}\tan\alpha) \\ &= \frac{1}{\tan\alpha+\tan\beta}((\vec{p}-\vec{r})\tan\beta+(\vec{p}-\vec{q})\tan\alpha) \end{aligned}$$

Furthermore, we observe from our triangle that

$$\frac{\ell_1+\ell_2}{||\vec{h}||} = \frac{\ell_1+\frac{\tan\alpha}{\tan\beta}\ell_1}{\ell_1\tan\alpha} = \frac{\tan\alpha+\tan\beta}{\tan\alpha\tan\beta}$$

Now let's put all of this back into our expression for the gradient of the area.

$$\begin{aligned} \nabla_{\vec{p}}A &= \frac{1}{2}b\vec{e}_\perp \\ &= \frac{1}{2}(\ell_1+\ell_2)\frac{\vec{h}}{||\vec{h}||} \\ &= \frac{1}{2}((\vec{p}-\vec{r})\cot\alpha+(\vec{p}-\vec{q})\cot\beta) \end{aligned}$$

This shows that we can write the gradient of the area of the triangle in terms of its vertices and the cotangents of its interior angles. This is a useful fact that's going to come up over and over again. Finally, we sum over the triangles around a vertex to get the gradient of the area functional of the mesh:

$$\nabla_{\vec{p}}A = \frac{1}{2}\sum_j(\cot\alpha_j+\cot\beta_j)(\vec{p}-\vec{q}_j)$$

Note that as we refine the mesh and the vertices get closer and closer to each other, this quantity vanishes, so just like with our discrete Gaussian curvature definition, we're going to define the discrete mean curvature integrated over the Voronoi cell of the vertex to be this quantity:

$$\int_V H\vec{n}dA = \frac{1}{2}\sum_j(\cot\alpha_j+\cot\beta_j)(\vec{p}-\vec{q}_j)$$

Just like with Gaussian curvature, to obtain our estimate of mean curvature at vertex p we'll need to divide by the area of the Voronoi cell.

5 Choosing an Approach

As mentioned earlier, there are a LOT of different papers (e.g. [4] [3] [5]) describing all kinds of approaches to computing curvature, including ones that don't fall into the two categories presented here. Each one is optimized for a specific application, but potentially useful in other applications as well. The question, then, is which one to choose for your application. It turns out that for many applications, it doesn't matter; if your input meshes are well-behaved and your application can tolerate some inaccuracy, then you can pick an approach more or less at random and it should work. If it then turns out that that particular approach doesn't work for your application, you can figure out what the failure cases are, and then pick a different approach that claims to address those failure cases; there are so many papers out there that a paper that matches your failure case is virtually guaranteed to exist. If you can't put a finger on what your failure cases are, then just try another approach at random; the odds are in your favor. Even better, most of these methods are easy to implement, so playing this search game is pretty inexpensive. As an added benefit, you'll build up your own collection of curvature implementations, so the next time you need curvature for something, you can just try out your suite so far to see if something fits out of the box.

References

[1] ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. Anisotropic polygonal remeshing. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 485–493.

[2] DECARLO, D., AND RUSINKIEWICZ, S. Highlight lines for conveying shape. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering* (New York, NY, USA, 2007), NPAR '07, ACM, pp. 63–70.

[3] GRINSPUN, E., GINGOLD, Y., REISMAN, J., AND ZORIN, D. Computing discrete shape operators on general meshes, 2006.

[4] JIAO, X., AND ZHA, H. Consistent computation of first- and second-order differential quantities for surface meshes. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling* (New York, NY, USA, 2008), SPM '08, ACM, pp. 159–170.

[5] KALOGERAKIS, E., SIMARI, P., NOWROUZEZAHRAI, D., AND SINGH, K. Robust statistical estimation of curvature on discretized surfaces. In *Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), SGP '07, Eurographics Association, pp. 13–22.

[6] MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Visualization and Mathematics III* (2002), H. C. Hege and K. Polthier, Eds., Mathematics and Visualization, Springer, pp. 113–134.

[7] RUSINKIEWICZ, S. Estimating curvatures and their derivatives on triangle meshes. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium* (Washington, DC, USA, 2004), 3DPVT '04, IEEE Computer Society, pp. 486–493.

[8] TAUBIN, G. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the Fifth International Conference on Computer Vision* (Washington, DC, USA, 1995), ICCV '95, IEEE Computer Society, pp. 902–.

[9] WANG, Y., LIU, B., AND TONG, Y. Linear surface reconstruction from discrete fundamental forms on triangle meshes. *Comp. Graph. Forum 31*, 8 (Dec. 2012), 2277–2287.

[10] ZHAO, H., AND XU, G. Triangular surface mesh fairing via gaussian curvature flow. *J. Comput. Appl. Math. 195*, 1 (Oct. 2006), 300–311.