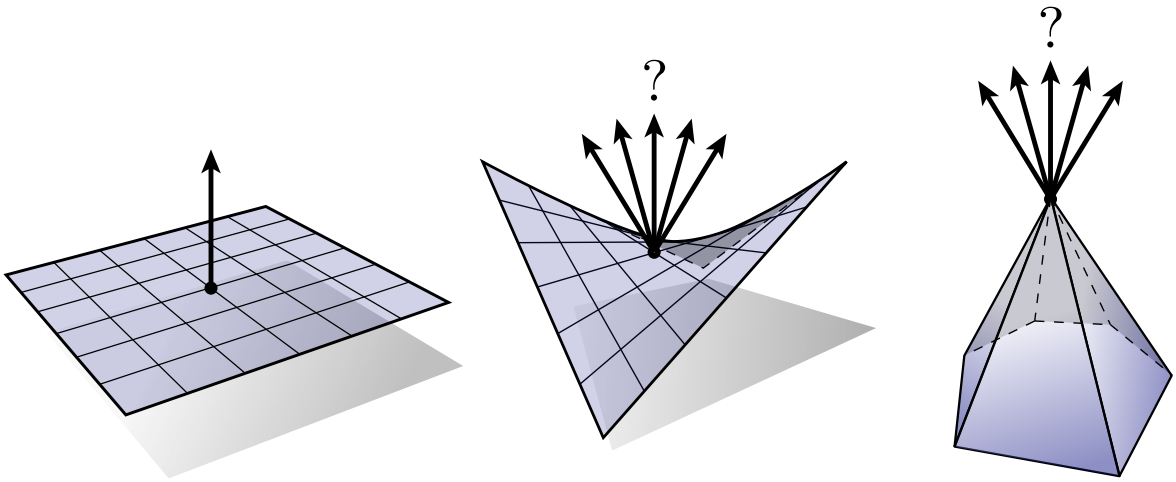
CS177: Discrete Differential Geometry (2013)

Caltech | Tu/Tr 10:30-11:55am | ANN 314

Homework 2: Normals of Discrete Surfaces

October 16, 2013 Homework

[Homework 2 has been released (below) and is due 10/28 by 5pm. We suspect there will be a lot of questions about C++, `libDDG`, etc., so please do not hesitate to ask questions either in the comments (preferred) or via email. This assignment has both theory and implementation parts. The code for the implementation part is here: ddg_normals. Please read the documentation included to see what it takes to compile this code on your machine. There are instructions for MACOS, Linux, and Windows/cygwin. While we have made every effort to ensure that everything compiles on these architectures there may be small hiccups due to new versions of the OS or the involved libraries. We will collect any "gotchas" in the comment section below. For this reason I ***URGE*** *everyone to pretty much try right away to build the library and run it. I am here to help, but please do not wait until the last minute.]*

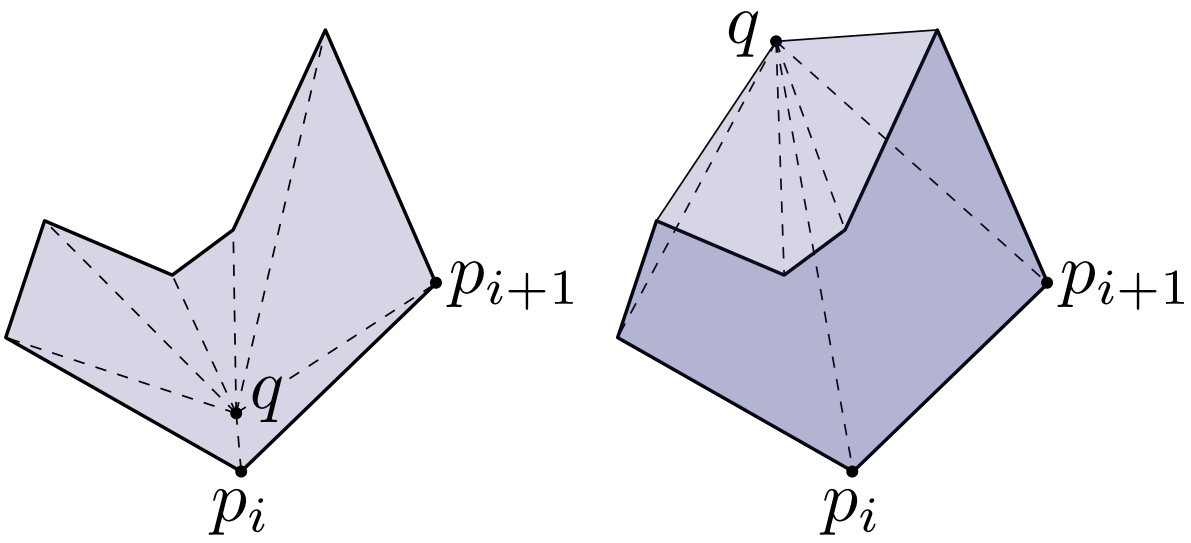


For a smooth surface in $\mathbb{R}^3$, the *normal* direction is easy to define: it is the unique direction orthogonal to all tangent vectors — in other words, it's the direction sticking "straight out" of the surface. For discrete surfaces the story is not so simple. If a mesh has *planar* faces (all vertices lie in a common plane) then of course the normal is well-defined: it is simply the normal of the plane. But if the polygon is *nonplanar*, or if we ask for the normal at a *vertex*, then it is not as clear how the normal should be defined.

In practice there are a number of different possibilities, which arise from different ways of looking at the smooth geometry. But before jumping in, let's establish a few basic geometric facts.

Area of a Polygon

Here's a simple question: how do you compute the area of a polygon in the plane? Suppose our polygon has vertices $p_1, p_2, \ldots, p_n$. One way to compute the area is to stick another point q in the middle and sum up the areas of triangles q, p_i, p_{i+1} as done on the left:



A cute fact about life is that if we place q *anywhere* and sum up the *signed* triangle areas, we still recover the polygon area! (Signed area just means *negative* if our vertices are oriented clockwise; *positive* if they're counter-clockwise.) You can get an idea of why this happens just by looking at the picture: positive triangles that cover "too much" area get accounted for by negative triangles.

The proof is an application of Stokes' theorem — consider a different expression for the area A of a planar polygon P:

$$A = \int_P dx \wedge dy.$$

Noting that $dx \wedge dy = d(xdy) = -d(ydx)$, we can also express the area as

$$A = \frac{1}{2}\int_P d(xdy) - d(ydx) = \frac{1}{2}\int_{\partial P} xdy - ydx,$$

where we've applied Stokes' theorem in the final step to convert our integral over the entire surface into an integral over just the boundary. Now suppose that our polygon vertices have coordinates $p_i = (x_i, y_i)$. From here we can explicitly work out the boundary integral by summing up the integrals over each edge e_{ij} :

$$\int_{\partial P} xdy - ydx = \sum \int_{e_{ij}} xdy - ydx.$$

Since the coordinate functions x and y are *linear* along each edge (and their differentials dx and dy are

therefore *constant*), we can write these integrals as

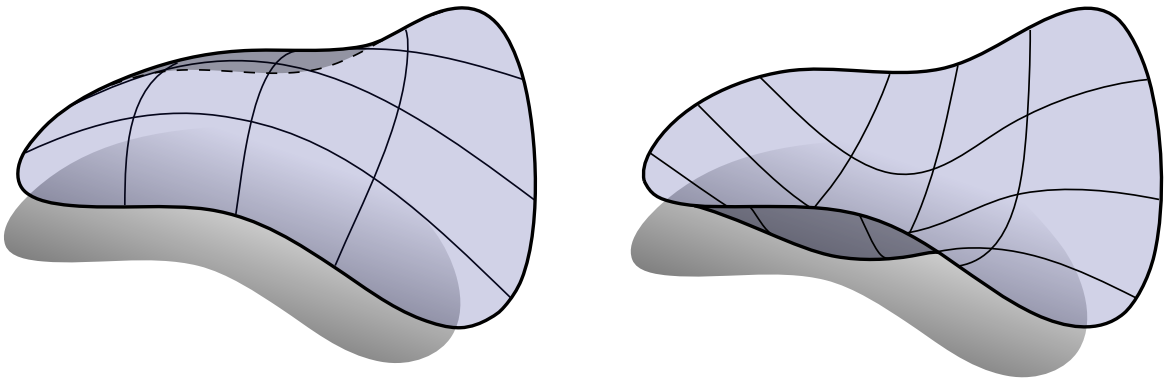
$$\begin{aligned}\sum \int_{e_{ij}} x dy - y dx &= \sum \frac{x_i + x_j}{2}(y_j - y_i) - \frac{y_i + y_j}{2}(x_j - x_i) \\ &= \frac{1}{2}\sum (p_i + p_j) \times (p_j - p_i) \\ &= \frac{1}{2}\sum p_i \times p_j - p_i \times p_i - p_j \times p_j - p_j \times p_i \\ &= \sum p_i \times p_j .\end{aligned}$$

In short, we've shown that the area of a polygon can be written as simply

$$A = \frac{1}{2}\sum_i p_i \times p_j .$$

Exercise 2.1 Complete the proof by showing that for any point q the signed areas of triangles (q, p_i, p_{i+1}) sum to precisely the expression above.

Vector Area



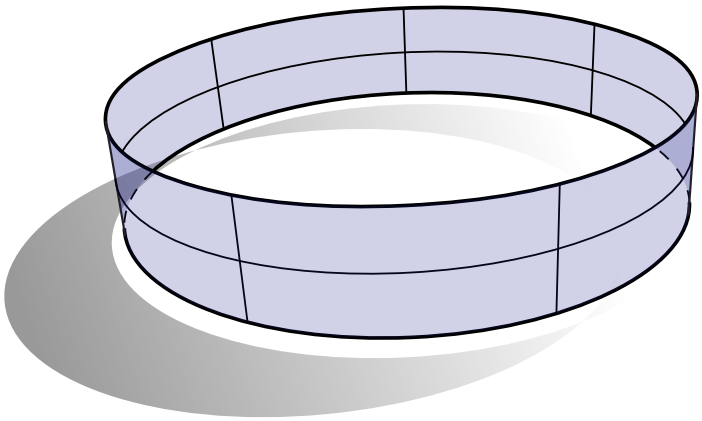
A more general version of the situation we just looked at with polygon areas is the *vector area* of a surface patch $f : M \to \mathbb{R}^3$, which is defined as the integral of the surface normal over the entire domain:

$$N_{\mathcal{V}} := \int_M N dA.$$

A very nice property of the vector area is that it depends only on the shape of the boundary ∂M (as you will demonstrate in the next exercise). As a result, two surfaces that look very different (such as the ones above) can still have the same vector area — the physical intuition here is that the vector area measures the total *flux* through the boundary curve.

For a *flat* region the normal is constant over the surface and we get just the usual area times the unit normal vector. Things get more interesting when the surface is not flat — for instance, the vector area of

a circular band is *zero* since opposing normals cancel each-other out:

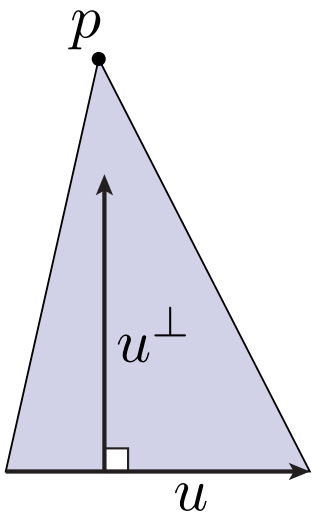


Exercise 2.2 Using Stokes' theorem, show that the vector area can be written as

$$N\mathcal{V} = \frac{1}{2}\int_{\partial M} f \wedge df,$$

where the product of two vectors in $\mathbb{R}^3$ is given by the usual cross product $\times$.

Gradient of Triangle Area



Here's another fairly basic question: consider a triangle sitting in $\mathbb{R}^3$, and imagine that we're allowed to pull on one of its vertices p. What's the quickest way to increase its area A? In other words, what's the *gradient* of A with respect to p?

Exercise 2.3 Show that the area gradient is given by

$$\nabla_p A_\sigma = \frac{1}{2}\mathbf{u}^\perp$$

where $\mathbf{u}^\perp$ is the edge vector across from p rotated by an angle $\pi/2$ in the plane of the triangle (such that it points toward p). You should require only a few *very simple* geometric arguments — there's no need to write things out in coordinates, etc.

Vertex Normals from Area Gradient

Ok, with these facts out of the way let's take a look at some different ways to define vertex normals. There are essentially only two definitions that arise naturally from the smooth picture: the *area gradient* and the *volume gradient*; we'll start with the former.

The area gradient asks, "which direction should we `push' the surface in order to increase its total area A as quickly as possible?" Sliding all points tangentially along the surface clearly doesn't change anything: we just end up with the same surface. In fact, the only thing we can do to increase surface area is move the surface in the *normal* direction. The idea, then, is to *define* the vertex normal as the gradient of area with respect to a given vertex.

Since we already know how to express the area gradient for a single triangle σ, we can easily express the area gradient for the entire surface:

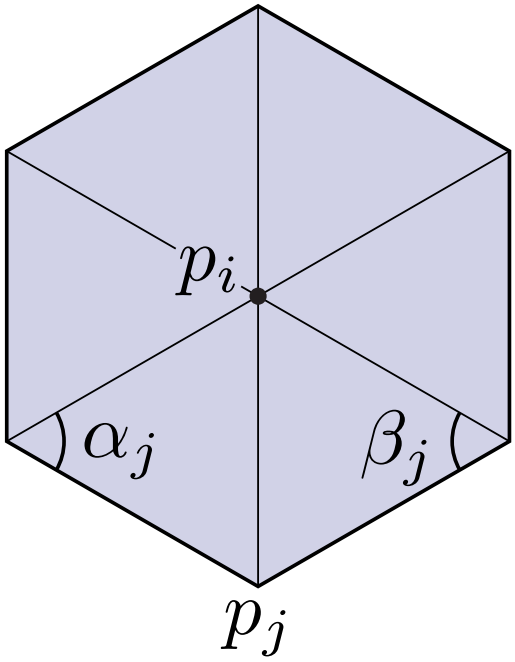
$$\nabla_p A = \sum_\sigma \nabla A_\sigma.$$

Of course, a given vertex p influences only the areas of the triangles touching p. So we can just sum up the area gradients over this small collection of triangles.

Exercise 2.4 Show that the gradient of surface area with respect to vertex p_i can be expressed as

$$\nabla_p A = \frac{1}{2} \sum_j (\cot \alpha_j + \cot \beta_j)(p_j - p_i)$$

where p_j is the coordinate of the jth neighbor of p_i and α_j and β_j are the angles across from edge (p_i, p_j).



Mean Curvature Vector

The expression for the area gradient derived in the last exercise shows up all over discrete differential geometry, and is often referred to as the *cotan formula*. Interestingly enough this same expression appears when taking a completely different approach to defining vertex normals, by way of the *mean curvature vector* HN. In particular, for a smooth surface $f : M \to \mathbb{R}^3$ we have

$$\Delta f = 2HN$$

where H is the mean curvature, N is the unit surface normal (which we'd like to compute), and Δ is the Laplace-Beltrami operator (see below). Therefore, another way to define vertex normals for a discrete surface is to simply apply a discrete Laplace operator to the vertex positions and normalize the resulting vector.

The question now becomes, "how do you discretize the Laplacian?" We'll take a closer look at this question in the future, but the remarkable fact is that the most straightforward discretization of Δ leads us right back to the cotan formula! In other words, the vertex normals we get from the mean curvature vector are precisely the same as the ones we get from the area gradient.

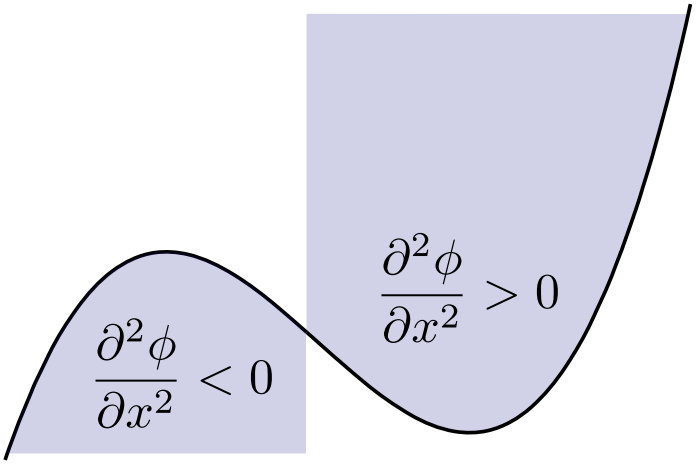
This whole story also helps us get better intuition for the Laplace-Beltrami operator Δ itself. Unfortunately, there's no really nice way to write Δ — the standard coordinate formula you'll find in a textbook on differential geometry is $\Delta\phi = \frac{1}{\sqrt{|g|}} \frac{\partial}{\partial x^i} (\sqrt{|g|} g^{ij} \frac{\partial}{\partial x^j} \phi)$, where g is the metric. However, this obfuscated expression provides little intuition about what Δ really does, and is damn-near useless when it comes to discretization since for a triangle mesh we *never* have a coordinate representation of g! Later, using exterior calculus, we'll see that the (0-form) Laplacian can be expressed as $\Delta = \star d \star d$, which leads to a fairly straightforward discretization. But for now, we'll make use of a nice tool we learned about earlier: *conformal parameterization*. Remember that if f is a conformal map, then lengths on M and lengths on $f(M)$ are related by a positive scaling e^u. In other words, $|df(X)| = e^u |X|$ for

some real-valued function u on M. Moreover, a conformal parameterization always exists — in other words, we don't have to make any special assumptions about our geometry in order to use conformal coordinates in proofs or other calculations. The reason conformal coordinates are useful when talking about Laplace-Beltrami is that we can write Δ as simply a rescaling of the standard Laplacian in the plane, i.e., as the sum of second partial derivatives divided by the metric scaling factor e^{2u}:

$$\Delta\phi = \frac{d(d\phi(X))(X) + d(d\phi(Y))(Y)}{e^{2u}},$$

where X and Y are any pair of unit, orthogonal directions.

What's the geometric meaning here? Remember that for a good old-fashioned function $\phi : \mathbb{R} \to \mathbb{R}$ in 1D, second derivatives basically tell us about the *curvature* of a function, e.g., is it concave or convex?



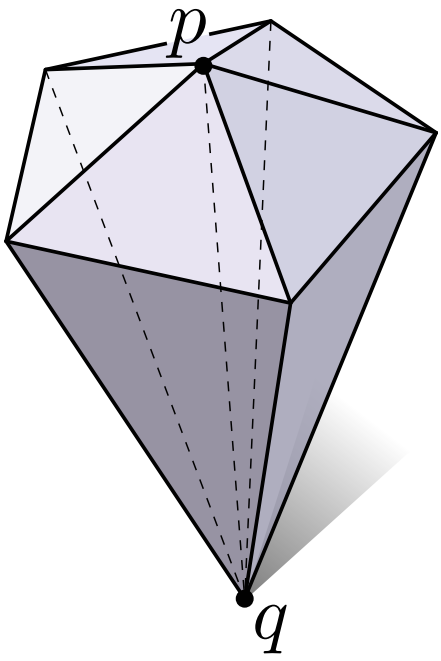
Well, since Δ is a *sum* of second derivatives, it's no surprise that it tells us something about the *mean* curvature!

Exercise 2.5 Show that the relationship $\Delta f = 2HN$ holds.

Vertex Normals from Volume Gradient

An alternative way to come up with normals is to look at the *volume gradient*. Suppose that our surface encloses some region of space with total volume $\mathcal{V}$. As before, we know that sliding the surface along itself tangentially doesn't really change anything: we end up with the same surface, which encloses the same region of space. Therefore, the quickest way to increase $\mathcal{V}$ is to again move the surface in the normal direction. A somewhat surprising fact is that, in the discrete case, the volume gradient actually yields a *different* definition for vertex normals than the one we got from the area gradient. To express this gradient, we'll use three-dimensional versions of of our "basic facts" from above.

First, much like we broke the area of a polygon into triangles, we're going to decompose the volume enclosed by our surface into a collection of tetrahedra. Each tetrahedron includes exactly one face of our discrete surface, along with a new point q. For instance, here's what the volume might look like in the vicinity of a vertex p:



Just as in the polygon case the location of q makes no difference, as long as we work with the *signed* volume of the tetrahedra. (Can you prove it?)

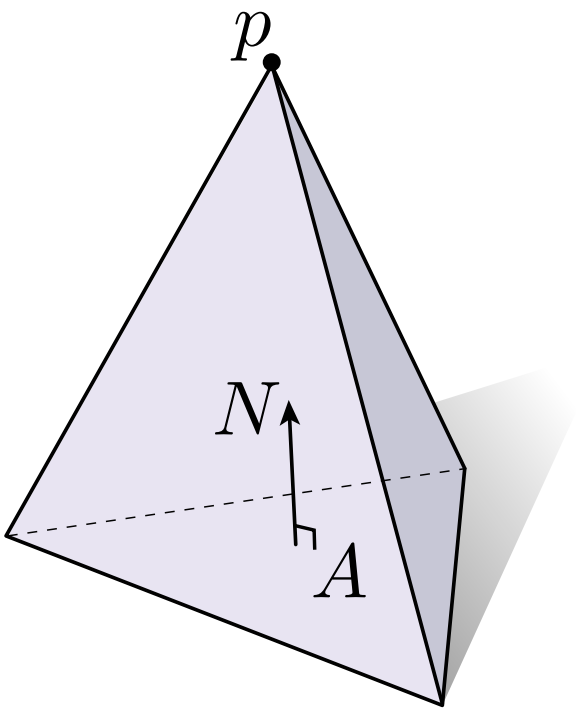
Next, what's the volume gradient for a single tetrahedron? One way to write the volume of a tet is as

$$\mathcal{V} = \frac{1}{3}Ah,$$

where A is the area of the base triangle and h is the height. Then using the same kind of geometric reasoning as in the triangle case, we know that

$$\nabla_p \mathcal{V} = \frac{1}{3}AN,$$

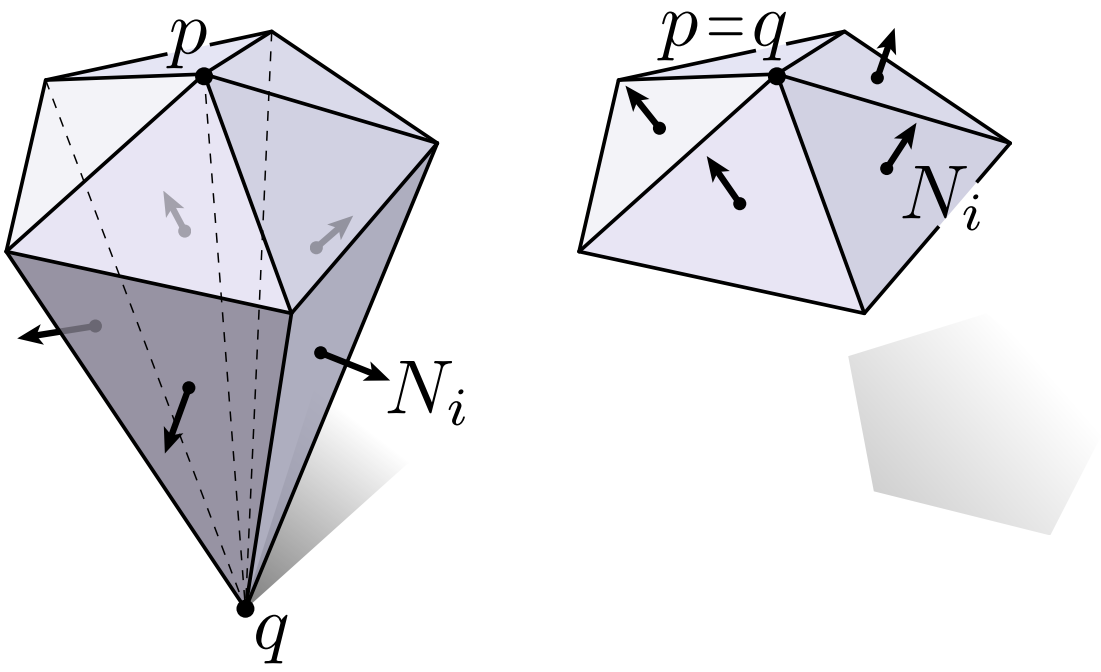
where N is the unit normal to the base.



To express the gradient of the enclosed volume with respect to a given vertex p, we simply sum up the gradients for the tetrahedra containing p:

$$\nabla_p \mathcal{V} = \sum_i \mathcal{V}_i = \frac{1}{3} \sum_i A_i N_i.$$

At first glance this sum does not lead to a nice expression for $\Delta_p \mathcal{V}$ — for instance, it uses the normals N_i of faces that have little to do with our surface geometry. However, remember that we can place q anywhere we please and still get the same expression for volume. In particular, if we put q directly on top of p, then the N_i and A_i coincide with the normals and areas (respectively) of the faces containing p from our original surface:

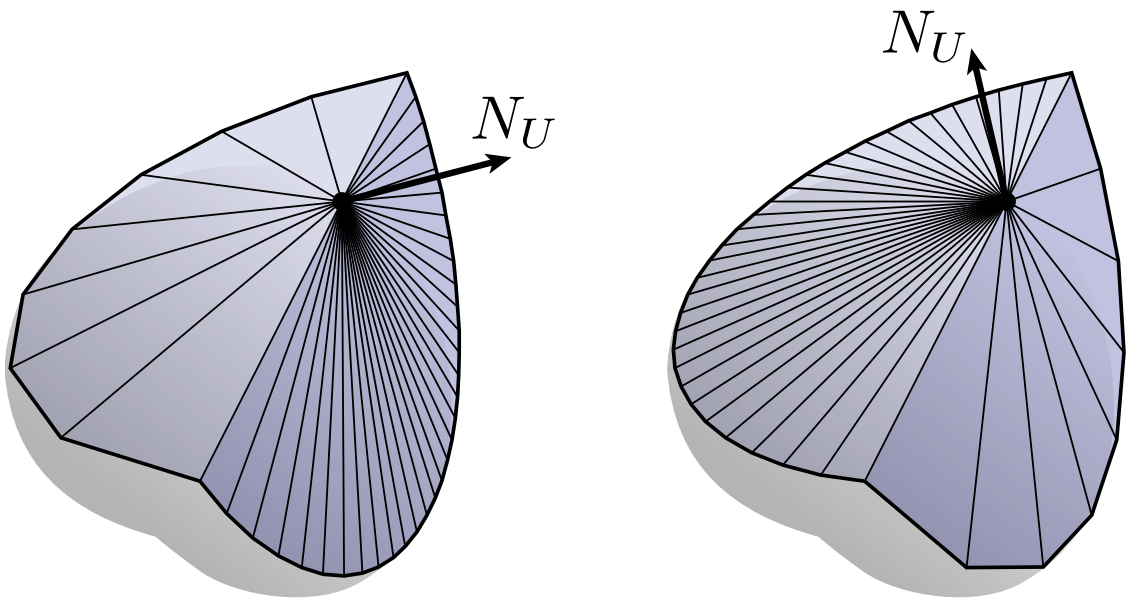


Exercise 2.6 Show that the volume gradient points in the same direction as the vector area $N_{\mathcal{V}}$ (i.e., show that they are the same up to a constant factor).

Other Definitions

So far we've only looked at definitions for vertex normals that arise from some smooth definition. This way of thinking captures the essential spirit of discrete differential geometry: relationships from the smooth setting should persist unperturbed in the discrete setting (e.g., $\Delta f = 2HN$ should be true independent of whether Δ, H, and N are smooth objects or discrete ones). Nonetheless, there are a number of common definitions for vertex normals that do not have a known origin in the smooth world. (Perhaps you can find one?)

Uniform Weighting

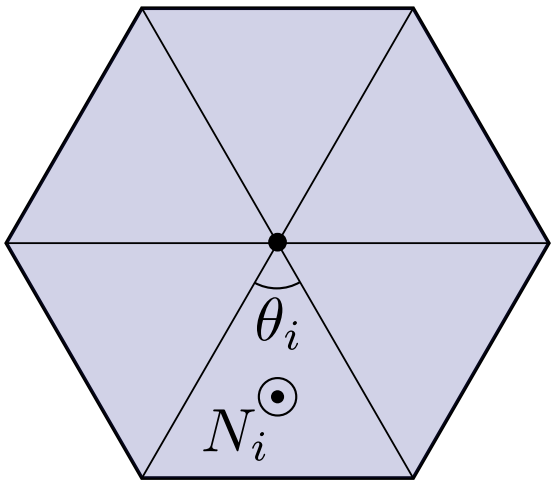


Perhaps the simplest way to get vertex normals is to just add up the neighboring face normals:

$$N_U := \sum_i N_i$$

The main drawback to this approach is that two different tessellations of the same geometry can produce very different vertex normals, as illustrated above.

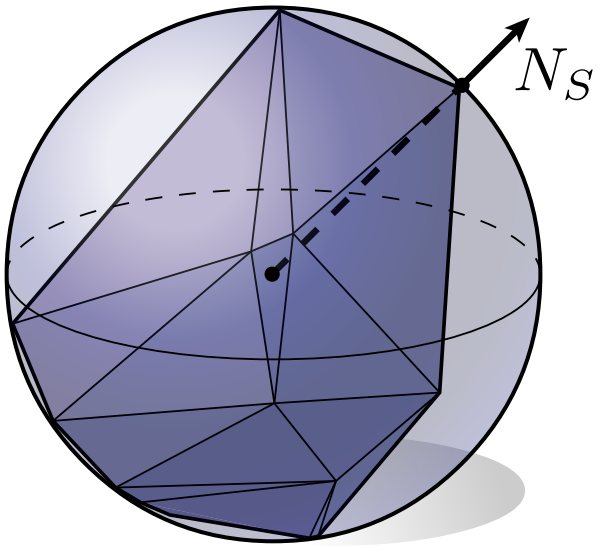
Tip-Angle Weights



A simple way to reduce dependence on the tessellation is to weigh face normals by their corresponding *tip angles* θ, i.e., the interior angles incident on the vertex of interest:

$$N_\theta := \sum_i \theta_i N_i$$

Sphere-Inscribed Polytope



Here's another interesting approach to vertex normals: consider the sphere S^2 consisting of all points unit distance from the origin in $\mathbb{R}^3$. A nice fact about the sphere is that the unit normal N at a point $x \in S^2$ is simply the point itself! I.e., $N(x) = x$. So if we start out with a polytope whose vertices all sit on the sphere, one reasonable way to define vertex normals is to simply use the vertex positions.

In fact, it's not too hard to show that the direction of the normal at a vertex p_i can be expressed purely in terms of the edge vectors $e_j = p_j - p_i$, where p_j are the immediate neighbors of p_i. In particular, we have

$$N_S = \frac{1}{c} \sum_{j=0}^{n-1} \frac{e_j \times e_{j+1}}{|e_j|^2 |e_{j+1}|^2}$$

where the constant $c \in \mathbb{R}$ can be ignored since we're only interested in the *direction* of the normal. (For a detailed derivation of this expression, see Max, *"Weights for Computing Vertex Normals from Facet Normals."*) Of course, since this expression depends only on the edge vectors it can be evaluated on any mesh (not just those inscribed in a sphere).

Coding Assignment

Implement the following methods in `libDDG`:

- `Vertex::normalEquallyWeighted()`
  Purpose: returns unit vertex normal using uniform weights N_U
- `Vertex::normalAreaWeighted()`
  Purpose: returns unit vertex normal using face area weights $N_\mathcal{V}$
- `Vertex::normalAngleWeighted()`
  Purpose: returns unit vertex normal using tip angle weights N_θ
- `Vertex::normalMeanCurvature()`

Purpose: returns unit mean curvature normal Δf

- `Vertex::normalSphereInscribed()`
  Purpose: returns unit sphere-inscribed normal N_S

(The definitions for these methods can be found in `libddg/src/Vertex.cpp`.)

Once you've successfully implemented these methods, test them out on the provided meshes. For convenience you can flip through the different methods using the keys `[1]-[5]`. You can also see what the mesh looks like by viewing it in "wireframe" mode. Do you notice that some definitions work better than others? When? Why? The only thing you need to submit for the coding assignment (via email) is your modified version of `Vertex.cpp`. (Of course, if you modify other files you should submit these too! For instance, you might find it convenient to add a method `HalfEdge::cotan()` that computes the cotangent of the angle across from a given half edge.)

Example meshes can be found here: ddg_hw2_meshes

One thought on "Homework 2: Normals of Discrete Surfaces"

★Peter Schröder

October 16, 2013 at 9:31 pm

For folks running Mountain Lion: you may need to change the makefile to include the library -lsuitesparseconfig just before -lm in the DDG_SUITESPARSE_LIBS variable if you get complaints during linking that some timer routines can't be found. Whether this is also an issue with the corresponding make variables for Linux and Windows/cygwin I do not know.