Scientific visualization hands-on session: discrete surfaces
Pascal Frey*, Chantal Oberson Ausoni*

The purpose of this hands-on session is to give an overview of the importance of discrete surfaces in scientific visualization through the resolution of a small toy problem. First, we focus on the local reconstruction of a portion of a surface associated with a triangle $T$ or a collection of triangles. Then, we focus on the estimate of the local Gaussian curvature at the vertices of a triangulation.

1. Local reconstruction: Bézier surface

The problem we are investigating is part of a broader problem, of utmost importance in practical applications. We consider a surface $\Sigma$ embedded in $\mathbb{R}^3$, which is only known through a triangulation or a mesh $\mathcal{S} = (T_i)_{i=1,\ldots,N_\mathcal{S}}$. For sake of simplicity, we make the following assumptions on $\mathcal{S}$ and $\Sigma$:

1. the only information available about $\mathcal{S}$ is the list of triangles composing the triangulation and the list of vertices (coordinates);
2. the triangulation $\mathcal{S}$ is conforming, that is, the intersection between any two triangles $T_i, T_j, i \neq j$ is either the empty set, a common vertex or a common edge;
3. the underlying surface $\Sigma$ is a compact orientable manifold, without boundary;
4. the triangulation $\mathcal{S}$ is endowed with an orientation, i.e. the direct normal vectors to all the triangles of $\mathcal{S}$ consistently point towards one side of $\mathcal{S}$.

The triangulation $\mathcal{S}$ is intended as an interpolating under-sampled piecewise linear approximation of $\Sigma$.

We aim at refining the initial triangulation $\mathcal{S}$ using local subdivisions, thus producing a sequence of triangulations $\mathcal{S}_1, \ldots, \mathcal{S}_n$ getting closer and closer to a final approximation $\tilde{\mathcal{S}}$ of $\Sigma$, in the sense that the Hausdorff distance $d^H(\tilde{\mathcal{S}}, \Sigma)$ between $\tilde{\mathcal{S}}$ and $\Sigma$ is no larger than a given tolerance value $\varepsilon$.

![Figure 1. Poor geometric approximation (left) of a smooth surface (right).](image)

As mentioned before, the surface $\Sigma$ is not known analytically, and so as to drive the subdivision procedure (in order to refine the triangulation), we have to ’guess’ (invent, actually) it from the triangulation $\mathcal{S}$. At first, we have to compute approximations of some geometrical data attached to $\Sigma$ at the vertices of $\mathcal{S}$.

* UPMC Univ Paris 06, Institut du calcul et de la simulation, F-75005 Paris, France.
1.1. Normal estimates. In the neighborhood of a regular vertex $x$, the surface $\Sigma$ is sufficiently smooth (of class $C^1$, at least). It will reveal necessary to compute an approximation of $n(x)$, the unit normal vector to $\Sigma$ at $x$.

This can be achieved from the discrete surface $S$ by using a weighted sum of the normal vectors to the triangles $B_S(x)$ of the form:

$$n(x) \approx \frac{\sum_{T \in B_S} \alpha_T n_T}{|\sum_{T \in B_S} \alpha_T n_T|},$$

where $\alpha_T$ are suitable coefficients in $[0, 1]$ such that $\sum_{T \in B_S} \alpha_T = 1$, and $n_T$ is the unit normal to a triangle $T$. This formula only allows to compute an approximation of the normal if $S$ has already been oriented, as assumed. Several choices are possible as for the value of $\alpha_T$. For instance, some advocate to taking them all equal to one another, or to taking each $\alpha_T$ proportional to the area of $T$.

This additional piece of information about the surface $\Sigma$, approximate from the discrete geometry, allows us to define a local surface model for $\Sigma$ as shown hereafter. This model is broadly inspired by the one introduced in [2].

1.2. Local reconstruction of the surface $\Sigma$. The purpose of this paragraph is to explain the process of defining the local geometry of the underlying surface $\Sigma$ around a triangle $T$ of $S$ from the entities attached to $T$ and its three vertices.

**Figure 2.** Approximation of the normal vector to $\Sigma$ at $x$ as a weighted average of the normals $n_T$, to the triangles of the ball of a regular vertex $x$.

**Figure 3.** A piece of parametric Bézier cubic surface, associated to a triangle $T \in S$, with control points $b_{i,j,k}$ (left). Interpolation of the normal coefficient (right).
We suppose that each triangle \( T = a_0a_1a_2 \in S \) accounts for a smooth portion of the surface \( \Sigma \). The portion of \( \Sigma \) associated with \( T \) is modeled as a cubic piece of surface \( \sigma(\hat{T}) \), where

\[
\hat{T} := \{(u,v) \in \mathbb{R}^2; \; u \geq 0, \; v \geq 0, \; w := 1 - u - v \geq 0\}
\]

is the reference triangle in the plane (Fig. 3), and each component of \( \sigma : \hat{T} \rightarrow \mathbb{R}^3 \) is a polynomial of total degree 3 in the two variables \( u, v \in \hat{T} \).

Likewise, it is possible to parametrize the same portion of surface directly from the triangle \( T \), without involving the reference triangle \( \hat{T} \) in the plane, using another mapping \( \phi : T \rightarrow \mathbb{R}^3 \). In such case, it is obvious that \( \sigma = \phi \circ A_T \), where \( A_T : \hat{T} \rightarrow \mathbb{R}^3 \) is the unique affine mapping which transforms reference triangle \( \hat{T} \) into surface triangle \( T \).

It is convenient to write \( \sigma \) under the form of a Bézier cubic polynomial [1]:

\[
(1) \quad \forall (u,v) \in \hat{T}, \quad \sigma(u,v) = \sum_{i,j,k \in \{0,1,2,3\}} \frac{3!}{i!j!k!} w^i u^j v^k b_{i,j,k},
\]

where \( b_{i,j,k} \in \mathbb{R}^3 \) are control points, to be specified (Fig. 3). The boundary curves \( \gamma_0, \gamma_1, \gamma_2 \) of the portion of surface \( \sigma(\hat{T}) \) are respectively:

\[
\forall t \in [0,1], \quad \gamma_0(t) = \sigma(1-t,t), \quad \gamma_1(t) = \sigma(0,t), \quad \gamma_2(t) = \sigma(t,0).
\]

The choice of the control points \( b_{i,j,k} \) is dictated by the geometry of the surface \( \Sigma \).

1.2.1. Choice of the three 'vertex' control points. Since the triangulated surface \( S \) interpolates the underlying smooth surface \( \Sigma \), all triangle vertices lie on \( \Sigma \). This prompts the natural choice of the three vertices of \( T \) as the three extremities of \( \sigma(\hat{T}) \). Hence, we impose (see Fig 3):

\[
b_{3,0,0} = a_0, \quad b_{0,3,0} = a_1, \quad b_{0,0,3} = a_2.
\]

1.2.2. Choice of the six 'curve' control points. We want \( \sigma(\hat{T}) \) to be a smooth piece of surface. In particular, \( \sigma(\hat{T}) \) has a tangent plane \( T_{a_i} \Sigma \) and a normal vector \( n_i \) to \( \Sigma \) at each regular vertex \( a_i \), which can easily be approximated thanks to the reconstructed information of the previous section. The whole geometry of Bézier curves and surfaces can be expressed in terms of their control points. A mere derivation in (1) shows that, for instance, the tangent vector at \( a_0 \) to the boundary curve \( \gamma_2 \) is \( 3(b_{2,1,0} - b_{3,0,0}) \) and that to \( \gamma_1 \) is \( 3(b_{2,0,1} - b_{3,0,0}) \). Similar relations hold for \( a_1, a_2 \) and the control points \( b_{0,2,1}, b_{1,2,0}, b_{1,0,2} \) and \( b_{0,1,2} \).

Still, there is some room as for the choice of these coefficients. We want our local surface reconstruction by means of \( \sigma \) to be as independent as possible from the support triangle \( T \) used for its computation. This means that we would like the three boundary curves \( \gamma_0, \gamma_1, \gamma_2 \) to be independent from the choice of the points on these curves used to generate them. Because on any Riemannian manifold two close enough points are connected by a unique geodesic curve, a way to enforce this independency would be to choose their control points so that \( \gamma_0, \gamma_1, \gamma_2 \) are geodesics of \( \sigma(\hat{T}) \), that is, curves with constant speed. This property can in turn only be enforced in some kind of weak sense: for instance, in the case of \( \gamma_0 \) (similar conditions hold for \( \gamma_1, \gamma_2 \)), we impose that \( \gamma_0'(0) \) should be colinear to the orthogonal projection of \( (a_2 - a_1) \) over \( T_{a_2} \Sigma \), and have a fixed norm \( |\gamma_0'(0)| = |a_2 - a_1|/3 \), and symmetrically for \( \gamma_0'(1) \). Doing so uniquely determines the six coefficients attached to the boundary curves.

The rules for computing the four control points along each boundary curve \( \gamma_i \) only involve geometric data attached to this curve (or to its endpoints). This implies that the rules for generating a piece of \( \Sigma \) are consistent from one triangle to its neighbor, that is, if \( T_i, T_j \in S \) share a common edge \( pq \), the underlying boundary curve associated to \( pq \) via the local parametrization

Bézier surfaces were invented by French engineer Pierre Bézier in the 1960's as a method of describing the curves of automobile bodies.
generated from $T_i$ is the same as from $T_j$. It means also that there will be no visual artefact when drawing portions of $\Sigma$ from adjacent triangles along the common edge.

1.2.3. Choice of the central coefficient. We take:

$$b_{1,1,1} = m + \frac{m-v}{2}, \quad v := \frac{a_0 + a_1 + a_2}{3}, \quad m := \frac{b_{2,1,0} + b_{2,0,1} + b_{1,2,0} + b_{0,2,1} + b_{1,0,2} + b_{0,1,2}}{6},$$

which guarantees that, if there exists a quadratic polynomial parametrization $\tilde{\sigma} : \hat{T} \to \mathbb{R}^3$ whose boundary curves $t \mapsto \tilde{\sigma}(1-t,t), \tilde{\gamma}(t,0)$ and $\tilde{\gamma}_0(0,t)$ coincide with $\gamma_0, \gamma_1,$ and $\gamma_2$ respectively, then $\sigma = \tilde{\sigma}$ over $\hat{T}$.

1.2.4. Normal interpolation. The normal component at any point of the reconstructed surface is defined as a quadratic functional $n : \mathbb{R}^2 \to \mathbb{R}^3$, for $w = 1 - u - v$:

$$\forall (u,v) \in \hat{T}, \quad n(u,v) = \sum_{i+j+k=2} w^i u^j v^k n_{i,j,k}$$

where $n_{1,1,0}, n_{0,1,1}, n_{1,0,1}$ denote the mid-edge coefficients (Fig. 3, right).

Notice that considering a linear interpolation of the normal coincide with the procedure used in Phong shading in graphics rendering.

![Figure 4. From left to right: input triangulation, flat shading, smooth rendering associated with reconstructed surface (quadratic normal reconstruction), and underlying Bézier triangulation.](image)

1.3. Numerical experiments. On the course repository, we provide the archive `bezier.tgz` containing several useful functions in C language for dealing with triangulations and graphical features, as well as triangulation datasets in `examples.tgz`.

1.3.1. Data structures. We briefly recall here the (simplified) mesh structure used in the experiments (cf. Listing 1).

```c
/* mesh vertex coordinates */
typedef struct {
    double   c[3];
} Point;
typedef Point * pPoint;

/* mesh triangle connectivity */
typedef struct {
    int      v[3];
} Tria;
```
typedef Tria * pTria;

/* normal vector */
typedef struct {
    double n[3];
} Normal;
typedef Normal * pNormal;

/* surface triangulation */
typedef struct {
    int np, nt, nn, dim, ver;
pPoint point;
pTria tria;
pNormal normal;
} Mesh;

Listing 1. Mesh data structure

(1) The structure Point allows to store the Cartesian vertex coordinates (3 real numbers). If p0 is an object of type pPoint, then p0->c[i], i = 0, ..., 3 gives access to its coordinates. Likewise, the structure Normal is used to store normal vectors at mesh vertices.

(2) The structure Tria, devoted to triangles, is basically an array of 3 integer values v[3]. If pt is an object of type pTria, then pt->v[i] gives the global index of the three vertices composing the triangle pt.

(3) A triangulation (structure Mesh) is described as a collection of vertices (pPoint point), of triangles (pTria tria) and of normals (pNormal normal). Let mesh be an object of type pMesh. Then mesh->np, mesh->nt and mesh->nn denote the number of vertices, triangles and normals (identical to the number of vertices) in the mesh, respectively. Hence, mesh vertices will be denoted as:

    mesh->point[1], ..., mesh->point[k], ..., mesh->point[np].

    Notice that mesh->point[0] exists but it is never used.

The function loadMesh reads a surface triangulation and update the data structure mesh, defined as a global variable of type Mesh. Its first argument is the mesh structure, the second is the name of the input file. It returns the value 1 upon successfull completion and 0 otherwise.

Mesh mesh;

int main(int argc, char *argv[])
{
  memset(&mesh, 0, sizeof(Mesh));
  if ( argc < 2 )
  {
    printf(stdout,"%usage: %s
", argv[0]);
    return(1);
  }
  if (!loadMesh(&mesh, argv[1])) return(1);
  initGL(argc, argv);
  return(0);
}

Listing 2. Main program
1.3.2. Experiments. The objective is to implement the procedure to construct and represent a Bézier surface $\Sigma$ given a triangulation $\mathcal{S}$.

Several datasets are provided to allow for quick checking of the routines you develop. For instance, the file `triangle.mesh` contains a single surface triangle and 3 normals at its vertices. The file `d1.mesh` contains a unit sphere centered at the origin.

**Question 1:** Let $T = a_0a_1a_2$ be a surface triangle. Write the mathematical expressions of all ten coefficients $b_{i,j,k}$ in terms of the vertices $a_i$ and the normals $n_{2,0,0}, n_{0,2,0}, n_{0,0,2}$ at these vertices.

**Question 2:** Construct the C function:

```c
void compNormal(Mesh *mesh)
```

to compute an approximation of all unit outward normal vectors at mesh vertices. The normal vectors will be stored in the structure `Normal *normal` which needs to be allocated first. What is the order of time complexity (number of iterations in the algorithm) of this procedure?

**Question 3:** Construct the C function:

```c
void calcBezier(Mesh mesh, int iel, double bijk[10][3])
```

to compute the Bézier coefficients (control points) $b_{i,j,k}$ of a given triangle $iel$ in the mesh.

**Question 4:** OpenGL rendering of a Bézier triangle.

a. Write a C function:

```c
void bezierInt(double bijk[10][3], double np[3][3], double u, double v, double o[3], double no[3])
```

to compute the mapping $\sigma(u,v)$ on $\Sigma$ for any $(u,v) \in \hat{T}$ (cf. Eq. (1)). The input parameters are the Bézier coefficients $b_{i,j,k}$ of the triangle, the set of unit normal vectors at the triangle vertices, $(u,v)$ the 2d coordinates in $\hat{T}$. The output parameters are the location of the corresponding point $o$ onto the surface $\Sigma$ and its unit normal vector $no$. To simplify, a linear interpolation of the normal at point $o$ can be computed first.

b. Complete the OpenGL function

```c
void drawBezier(Mesh mesh, int iel, double bijk[10][3])
```

which currently draws the planar mesh triangles in $\mathcal{S}$, so that it allows to visualize the triangles corresponding to mid-edge refinement (Fig. 5 and 6, second left).

![Figure 5](image-url)  
**Figure 5.** Midpoint subdivision of a surface triangle, with a lifting of the midpoints of $T$ onto the underlying surface $\Sigma$. 

c. To evaluate the efficiency and the accuracy of the proposed reconstruction algorithm, consider the file d1.mesh which corresponds to a poor discretization of a unit sphere centered at the origin.
Write a C function to compute a scalar map of the discrepancy between the reconstructed surface $\Sigma$ and the true surface (e.g. compute distance between current and optimal vertex coordinates). Use the function: (in the file inout.c)

```c
saveSol(Mesh mesh, char *name, double *map)
```
to save the scalar map in the output file name.mesh, and then visualize the result using the program medit.

d. (facultative) Modify the OpenGL function (in the file grafic.c)

```c
void drawBezier(Mesh mesh, int iel, double bijk[10][3])
```
to visualize the triangles of a second order tessellation (two levels of refinement).
2. Curvature estimates

Next, we turn to another interesting problem: the estimate of local curvatures given a surface triangulation. Again in this case, we consider a discrete surface $\Sigma$ embedded in $\mathbb{R}^3$ known via a surface triangulation $\mathcal{S}$.

2.1. Curvature estimates. We recall that:

- the Gaussian curvature discrete operator can be defined as:
  $$\kappa_G = \frac{1}{A_i} \left( 2\pi - \sum_{v_j \in \mathcal{B}(v_i)} \theta_j \right)$$

  where $\mathcal{B}(v_i)$ denote the set of triangles sharing vertex $v_i$ and $\theta_j$ is the incident triangle angle.

- the mean curvature at a vertex is defined as:
  $$\kappa_H(v_i) = \frac{1}{2} \| \Delta\Sigma x_i \|,$$

  where $\Delta\Sigma$ provides a discrete approximation of the mean curvature normal $\kappa_H$:
  $$\Delta\Sigma(x_i) = \frac{1}{2A(v_i)} \sum_{v_j \in \mathcal{B}(v_i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})(x_i - x_j),$$

- the principal curvatures can be deduced easily from the previous formulas:
  $$\kappa_{1,2} = \kappa_H(v_i) \pm \sqrt{\kappa_H(v_i)^2 - \kappa_G(v_i)}.$$

2.2. Numerical experiments. On the course repository, we provide the archive `curve.tgz` containing the framework of a C program to deal with surface triangulations.

2.2.1. Data structures. In this second part, we use almost the same data structures as in part I. The sole minor modification is the addition of three auxiliary values in the `Point` structure.

1. The structure `Point` allows to store the Cartesian vertex coordinates (3 real numbers). If `p0` is an object of type `pPoint`, then `p0->c[i]`, $i = 0, \ldots, 3$ gives access to its coordinates. The fields `aux`, `alpha`, `beta` can be used to store any useful scalar information (e.g. the surface area of $\mathcal{B}(v_i)$, etc.).

2.2.2. Experiments. The objective is to implement the curvature estimates described in the course session and evaluate their efficiency and level of accuracy for surface triangulations.

**Question 5:** [easy] Construct the C function: (in the file `curvature.c`)

```c
void kappaGauss(Mesh mesh, double *map)
```
to compute an approximation of the Gaussian curvature at mesh vertices. Use the function

```c
saveSol(Mesh mesh, char *name, double *map)
```
to save this scalar map in the output file. Visualize the resulting map using `medit`. Is there any correlation between the valence of a vertex and the accuracy of the approximation?

**Question 6:**[difficult] Construct the C function: (in the file `curvature.c`)

```c
void kappaMean(Mesh mesh, double *map)
```
to compute the mean curvature at mesh vertices. Use the auxiliary fields to store temporary information.
Figure 8. Curvature plots of a triangulated saddle using pseudo-colors: (a) Mean, (b) Gaussian, (c) Minimum, (d) Maximum (from [2]).

References
