Homework #3: NURBS surfaces; geometric primitive fitting; slippable motions; basic point cloud and mesh processing [100 points]
Due Date: Wednesday, 26 February 2020

Theory and implementation

This is a mixed theory/implementation homework. Problem 1 provides some practice on calculations with NURBs surfaces (non-uniform B-splines). Problem 2 gives you a chance to actually model and visualize these NURBs surface as they are in 3D. Problems 3 and 4, as well as the forthcoming assignment 4, focus on fitting and segmenting low-level geometry data (point clouds, meshes) using geometric primitives (planes, spheres, cylinders, cones). They lead to an implementation according to the methods described in two papers [1, 2], one using classical geometry processing ideas and the other based on deep learning.

In this assignment, problems 3 and 4 ask you to derive the theory behind an algorithm for estimating primitive shape types and parameters fitting a given point cloud, as well as to implement some of these derivations. These estimations require some rudimentary linear algebra and optimization tools. As explained in the guidelines given with Homework 1, you may hand in a joint group write up for the problem parts that ask for an implementation. However, the theory parts should still be written individually, naming your collaborators.

A secondary goal of this problem is to introduce you to two standard programs for playing with surface meshes, **OpenMesh***,* `http://www.openmesh.org/` *and* **Meshlab***,* `http://meshlab.sourceforge.net/`*. You can download helper code for all problems in this assignment from the Handouts class web page.*

Problem 1. [A Splined Torus, 10 points]

Let r and R be real numbers with $0 < r < R$. As the angles α and β vary, the varying point $V = (X,Y,Z)$ given by the functions

$$\begin{aligned} X &:= (R + r\cos\alpha)\cos\beta \\ Y &:= (R + r\cos\alpha)\sin\beta \\ Z &:= r\sin\alpha \end{aligned}$$

traces out a *torus*, that is, the surface of an ideal donut or bagel. The torus has rotational symmetry about the Z axis, which passes through the middle of the torus hole. Any plane π through the Z axis cuts the torus in two circles of radius r, whose centers lie along the line where π cuts the XY plane, R units on each side of the origin. Varying α moves the point V around one of those circles. Varying β rotates the plane π around the Z axis.

Find a biquadratic, rational parameterization of this torus. That is, express each of the homogeneous coordinates $[w;x,y,z]$ of the varying point V as a polynomial in two parameters

that has degree at most 2 in each parameter when the other is held fixed. For consistency in notation, use Q and T as your two parameters, writing

$$V(Q;T) = [w(Q;T);x(Q;T),y(Q;T),z(Q;T)]$$

for certain polynomials w, x, y, and z[1].

Homogenize and polarize your parameterization. You may carry out these two steps in either order; the result will be the same. If you homogenize first, use p as the weight coordinate for the Q parameter, writing $Q = q/p$, and use s as the weight coordinate for the T parameter, writing $T = t/s$. If you polarize first, split the T parameter into two separate parameters T_1 and T_2, and split the Q parameter into Q_1 and Q_2. The homogenized polar form v of V will have the form

$$v((p_1;q_1),(p_2;q_2);(s_1;t_1),(s_2;t_2)) = [w;x,y,z],$$

where each coordinate w, x, y, and z is a polynomial in the eight variables p_1, q_1, p_2, q_2, s_1, t_1, s_2, and t_2.

One quarter of the torus consists of points that have Y and Z positive. (If you followed the hint above, those points will correspond, under your parametrization, to parameter pairs $(Q;T)$ where both Q and T are positive.) Describe this portion of the torus as a biquadratic, rational Bézier surface patch, that is, as a rectangular, tensor-product patch of degree $(2;2)$. In particular, give the coordinates of the nine Bézier sites of this patch. (If you followed the hint above, the patch will be $V([0..\infty] \times [0..\infty])$[2].

Problem 2. [Viewing the Torus, 15 points]

In this problem, we will use the surface from Problem 1 to model a torus using the *OpenMesh* software. The file you will edit is called `main-torus.cpp` in the `RenderTorus` folder of the assignment zip folder.

We have provided a bare-bones 3D viewing tool as part of your starter code. The controls are as follows:

- Dragging the left mouse button rotates the coordinate frame
- Dragging the middle mouse button pans the camera
- Dragging the right mouse button zooms

We have also provided code for rendering your mesh as wireframe, so that you can focus on the shape modeling and not have to worry about orientations and normals.

[1] Hint: Let $Q := \tan(\alpha/2)$ and $T := \tan(\beta/2)$ and remember some high-school trigonometry.

[2] Hint: Don't be distressed if one of the nine Bézier sites turns out to be the zero site, that is, the site all four of whose coordinates are zero.

2(a). [Evaluating a point on the surface, 5 points] We start with only the quarter of the torus from Problem 1, which we modeled using a biquadratic tensor-product surface. First, fill in the code section labeled STUDENT CODE SECTION 1 with the control points you found in Problem 1. Recall that we can evaluate a point using the Bézier sites by interpolating along the Q and T directions separately. Given a parameter value of $u \in [0,1]$ in the Q direction and $v \in [0,1]$ in the T direction, fill in the function $Calculate(u,v,k)$ in STUDENT CODE SECTION 2, which should return the homogeneous coordinates for a point on the surface (k is the index for the quarter of the torus we are modeling; for now just set it to 0).

2(b). [Construct a mesh. 5 points] We have defined a set of points on the surface patch and now we want to connect them to construct a mesh. This can be done using OpenMesh. In `generateMesh()`, as we evaluate points on the surface, we insert them into the mesh as vertices and store a 2D array of their associated *vertex handles*. Fill in STUDENT CODE SECTION 3 to construct a face by listing the face's vertex handles and then inserting the face into the mesh. Once you set up the vertices and faces correctly, OpenMesh will take care of the rest for you (i.e. connectivity between faces). In addition, OpenMesh supports both triangle meshes and more general polygon meshes, so you can use whichever you prefer for this problem (quad meshes are a natural choice for tensor product surfaces).

2(c). [Time for a whole torus, 5 points] Now you should have successfully rendered a quarter of the torus. Determine the control points for the other three quarters (using either intuition and symmetry or explicit calculation) and repeat the same approach to construct the rest of the torus. You can do this by varying the k parameter in the code. For full credit, you can simply construct the four parts separately and put them all in one big mesh. However, note that the vertices on the boundaries will have duplicates in the mesh data structure (two if they connect two quarters of the torus and four if they connect all four). Therefore, the quarters are internally disconnected on the boundaries even though they appear to connect on display.

2(d). [Optional for Extra Credit, 10 points]

- The program saves your mesh into a file `torus.off`, so you can load and view the mesh using other software (e.g., MeshLab) or the program for Problem 4. Construct the mesh with correct orientation, so that it can be rendered correctly with lighting. [5 points]
- Stitch the quarters of the torus together to address the vertex duplication problem described above. You can use MeshLab or the executable from Problem 4 to view the mesh with lighting. You should see cusps along the boundaries of the quarters initially (using smooth shading). Your task is to get rid of these cusps. [5 points]

Problem 3. [Primitive Types and Fitting, 40 points]

In this problem we consider how to fit planes, spheres, cyliders, and cones to point cloud data. Figure 1 shows the parameters used for each type of primitive. We specify the parameters $\mathbf{A}$ for the different primitive types as follows:

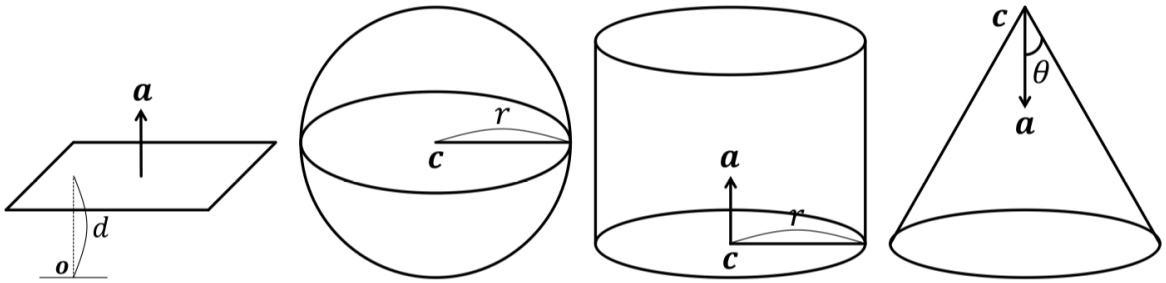

Figure 1: Primitive parameters

- Plane, $\mathbf{A} = (\mathbf{a}, d)$, the normal vector $\mathbf{a}$ and the distance from the origin d (positive if origin is above the plane).
- Sphere, $\mathbf{A} = (\mathbf{c}, r)$, the center $\mathbf{c}$ and the radius r.
- Cylinder, $\mathbf{A} = (\mathbf{a}, \mathbf{c}, r)$, the axis direction $\mathbf{a}$, an arbitrary point $\mathbf{c}$ on axis and the radius r.
- Cone, $\mathbf{A} = (\mathbf{a}, \mathbf{c}, \theta)$, the axis direction $\mathbf{a}$, the apex $\mathbf{c}$ and the half angle θ.

3(a). [Fitting error, 10 points] The fitting error of a point cloud to the primitive is defined as $\sum_i w_i D^2(\mathbf{p}_i, \mathbf{A})$, where the w_i are given weights (confidences of the point belonging to the primitive), A denotes the primitive parameters, and D is the distance from a point to the primitive surface. What is D for the four given primitives?

3(b). [Parameter estimation, 10 points] For a point cloud $\{\mathbf{p}_i\}$ with normals $\{\mathbf{n}_i\}$ and confidence weights $\{w_i\}$, we can compute the parameters of the primitives via an energy (loss) minimization. Following are proposed energies from [2].

- Plane. $\sum_i w_i(\mathbf{a}_i^T \mathbf{p}_i - d)^2$.
- Sphere. $\sum_i w_i(||\mathbf{p}_i - \mathbf{c}||^2 - r^2)^2$.
- Cylinder. Given that the axis is orthogonal to all point normals, we derive the energy as $\sum_i w_i(\mathbf{n}_i^T \mathbf{a})^2$ and solve for $\mathbf{a}$ first. Once the axis is known, points can be projected to a plane perpendicular to the axis to form a ring; there we can use the sphere loss to derive r.
- Cone. Given that the point normal for each point is orthogonal to the offset from the apex to this point, we derive the energy as $\sum_i w_i(\mathbf{n}_i^T(\mathbf{p}_i - \mathbf{c}))^2$ and solve for $\mathbf{c}$ first. Second, treating the coordinate of each point normal as a 3D point, the collection of point normals sit in a 3D plane whose normal is the axis $\mathbf{a}$. Therefore, we can estimate the axis with a plane fitting. Finally, for each point i we can derive a half angle θ_i between $\mathbf{a}$ and $\mathbf{p}_i - \mathbf{c}$, and the half angle of the cone is the weighted average $\theta = \sum_i w_i \theta_i$.

For example, to obtain the parameters for the plane, the problem can be converted to the standard form $\min_{\mathbf{a}} ||\text{diag}(\sqrt{\mathbf{w}})\mathbf{P}\mathbf{a}||^2$ with the constraint that $||\mathbf{a}|| = 1$. The solution is the right singular vector corrsponding to the smallest singular value of $\text{diag}(\sqrt{\mathbf{w}})\mathbf{P}$.

For the sphere, the optimal solution for r^2 is $\frac{\sum w_i ||\mathbf{p}_i - \mathbf{c}||^2}{\sum w_i}$. By replacing r^2 into the original energy, we get $\min_c ||\text{diag}(\sqrt{\mathbf{w}})(\mathbf{A}\mathbf{c} - \mathbf{b})||^2$ where $\mathbf{A}_i = 2(-\mathbf{p}_i + \frac{\sum_j w_j \mathbf{p_j}}{\sum_j w_j})$, $\mathbf{b}_i = \mathbf{p}_i^2 - \frac{\sum_j w_j \mathbf{p}_j^2}{\sum_j w_j}$. This is a weighted linear least square problem, where a linear solver (e.g., Cholesky factorization) can be used.

This problem asks you to derive the parameter solution for the cylinder and the cone.

3(c). [Implement the fitting error, 10 points] Please implement the fitting error functions as `compute_residue_loss` in `spfn/src/primitive/` for the cone and the cylinder. Implementation of the plane and the sphere losses are provided for your reference. Test script is provided for verification.

3(d). [Implement the parameter estimation (see README), 10 points] Please implement the functions `compute_parameters` in `spfn/src/primitive/` for the cone and the cylinder. Implementation of the plane and the sphere estimation is provided for your reference. Test script is provided for verification.

Problem 4. [Slippable Motions, 35 points]

Suppose we have a point cloud $\mathscr{P}$ consisting of a set of points $\{\mathbf{p}_i\}$ and associated surface normals $\{\mathbf{n}_i\}$, The points lie on or near a primitive geometric shape of the four allowed types: plane, sphere, cylinder, or cone. The task is to determine the primitive type and its parameters. The insight in [1] is that these primitives accept certain types of special tangential rigid motions that allow points to still lie on the surface after the motion. In other words, the primitive is *slippable* over itself under such motions. For example, a sphere is slippable under any rotation around its center.

4(a). [Describe the slippable motions, 10 points] Describe and illustrate the slippable motions for the four primitives we consider: the plane, the sphere, the cylinder, and the cone. Discuss their degrees of freedom.

4(b). [Equations for the slippable motion solutions, 5 points] Given N points in $\mathscr{P}$ with 3D locations $\{\mathbf{p}_i\}$ and surface normals $\{\mathbf{n}_i\}$, derive an equation to describe the relationship between the point cloud and a slippable motion consisting of a 3-dimensional angular velocity ω and a translational velocity $\mathbf{v}$.

4(c). [Formulation as an energy minimization problem, 5 points] The motion can be viewed as a six-dimensional vector consisting of the angular and the translational velocity. Formulate the motion estimation as an energy minimization problem, so that the motion can be simply derived from a linear equation of the form $\mathbf{A}\mathbf{X} = \mathbf{0}$. What is the objective energy and what is $\mathbf{A}$?

4(d). [Decide the primitive type, 5 points] Acceptable motion vectors that satisfies $\mathbf{AX} = \mathbf{0}$ form a subspace of the six-dimensional linear space. Given the subspace of the slippable motion, please describe how to determine the type of the primitive.

4(e). [Implement the primitive estimation, 10 points] Implement the function `EstimatePrimitive` in `slippage/src/primitive.cpp`. Test script is provided for verification.

What to hand in for the programming parts

You are required to submit "assignment3.zip" for this assignment by email to the TA with the same file distribution as you downloaded. Finally, please provide a "./report.pdf" including names of group members, the discussion of your code/method for problem 2, 3 and 4.

References

[1] Natasha Gelfand and Leonidas J Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223. ACM, 2004.

[2] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised fitting of geometric primitives to 3D point clouds. In *CVPR*. 2019.