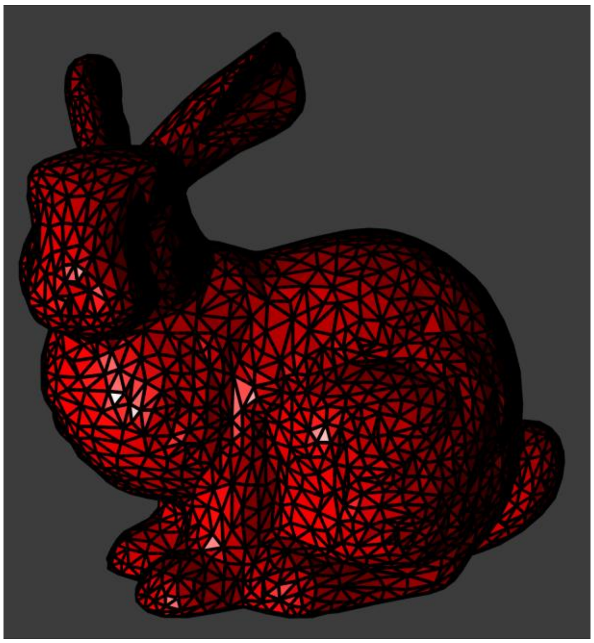
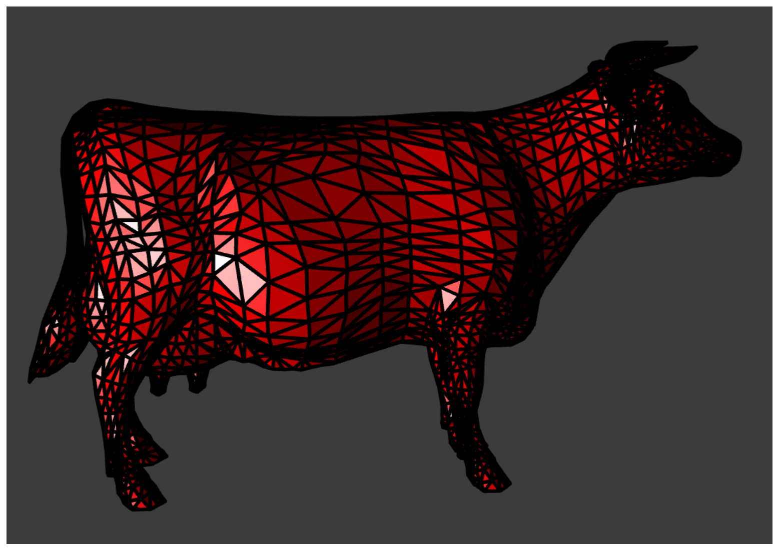
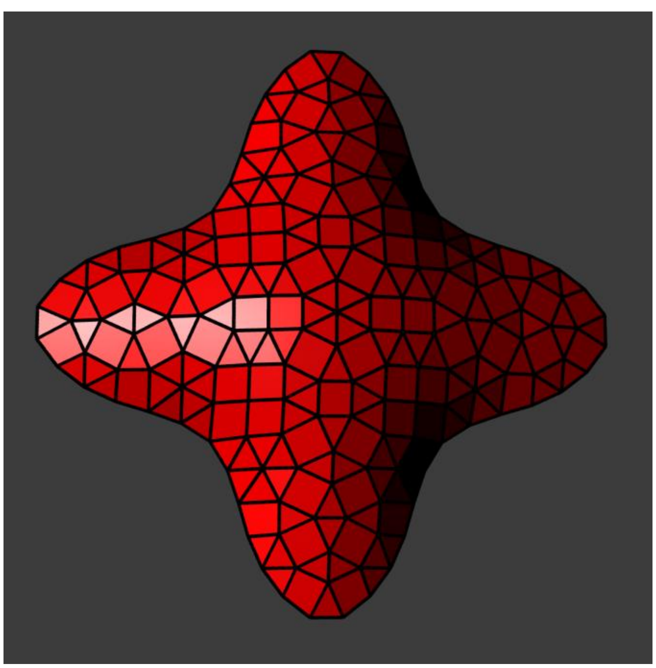
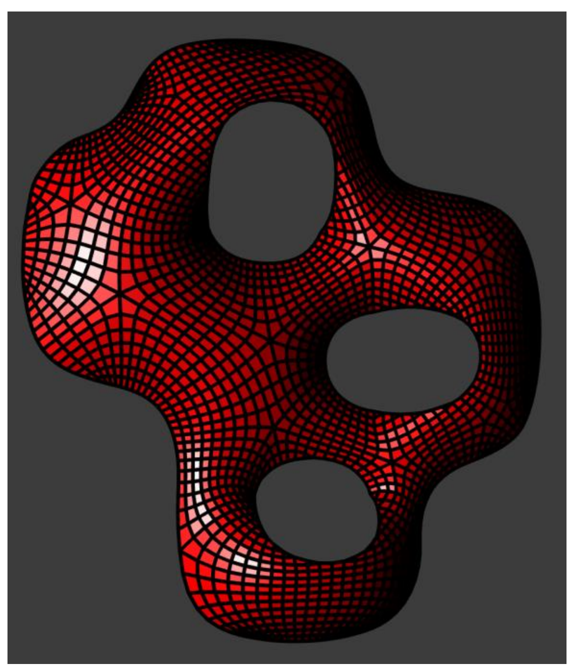
2020 春学期 CS271 计算机图形学 II

Project 4 – Discrete Curvatures

Total is 9 points and 1.5 bonus points. Your tasks are as follows:

1. [4 points] Render your half-edge meshes (loaded from .obj files) done in the previous project as *flat-faced* meshes with visible edges in OpenFrameworks. The reference is the mesh visualization as in MeshMan (http://www.holmes3d.net/graphics/meshman/). Show your renderings for **bunny.obj**, **cow.obj**, **lilium.obj**, and **roof.obj**. They should look like (you can pick a different color):

Hints: you can try your own approaches, but the best way is probably to create a "polygon-soup" as a **ofMesh**, which means every face (not necessarily triangle) has its own vertices (so if it is a quad, it has its 4 vertices). Each face's vertices are assigned *the same normal* (the face's normal) so they are guaranteed to be rendered in a "flat-faced" manner. Render the created ofMesh using your Blinn-Phong shaders as done in project 1 should work.

For each face, create its vertices using **ofMesh**'s **addVertex()** function. For every vertex, also add its normal by calling **addNormal()**. However, since **ofMesh** supports triangles only, for non-triangle faces, you do need to triangulate a face into multiple triangles and call ofMesh's **addTriangle()** to create the tringles (as vertex indices).

To render the visible edges, this is not easy with OpenFrameworks because the ofMesh's drawWireFrame() function doesn't have depth tests. The best way is probably to create thin "cylinder" ofMeshs for each of the edges and render them separately. A function to create a cylinder ofMesh from point a to b with radius is provided for you to use (Cylinder.cpp / Cylinder.h). To avoid z-fighting between rendering the mesh itself and the edge cylinder meshes, add the following codes before rendering the mesh:

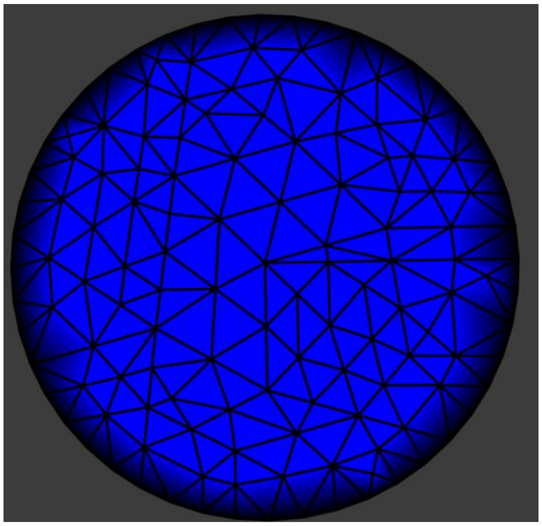
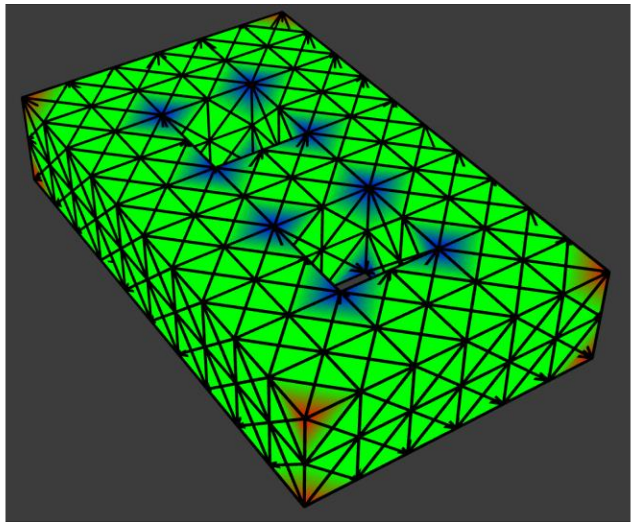
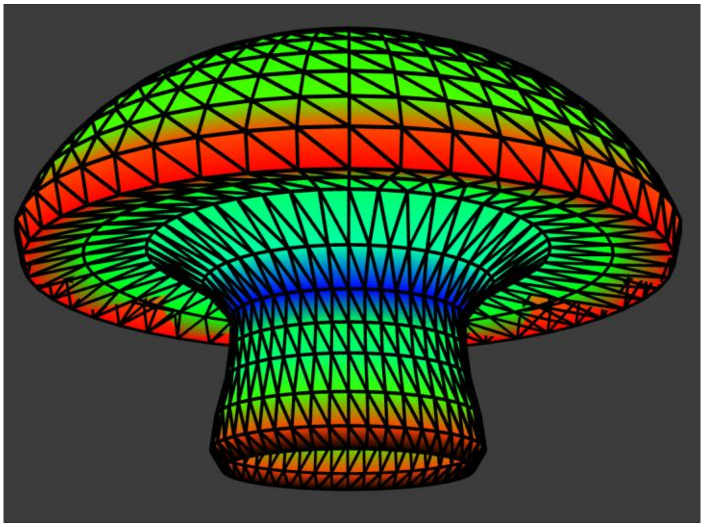
```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0f, 0.7f);
```

Be sure to disable GL_POLYGON_OFFSET_FILL after rendering the mesh. Consider using a different shader (e.g., just draw every fragment as black) for rendering the cylinder meshes.

[Grading: 3 points for rendering the flat-shaded meshes, 1 point for rendering the edges.]

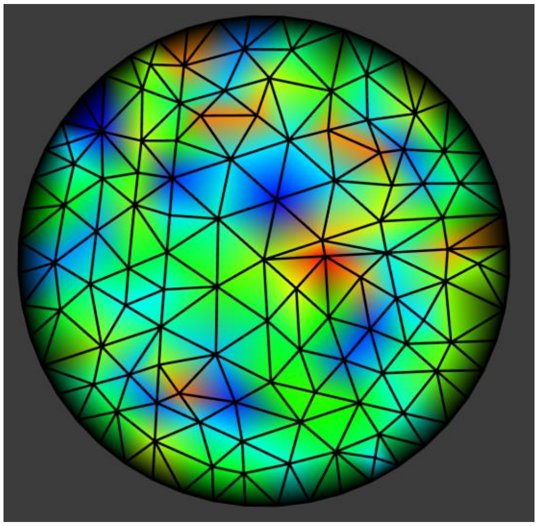
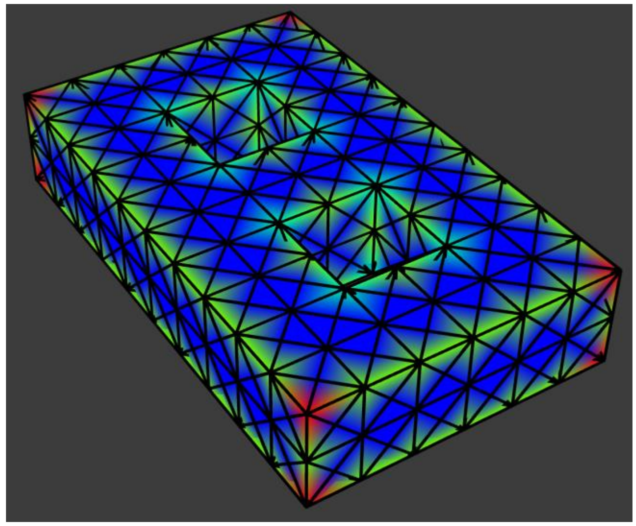
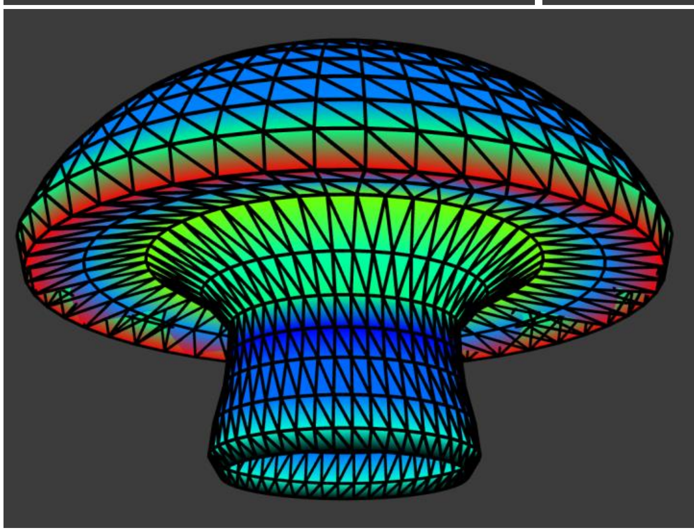
2. [5 points] Calculate two kinds of per-vertex discrete curvature measures and color the vertices using their curvature values.

   a) Calculate the **angle defects** (in radians) of every vertices (i.e., they are used as proxies for per-vertex *discrete Gaussian Curvatures*). Skip boundary vertices. Consider round values very close to 0 to 0 (e.g., "**if abs(value)<1e-5, value = 0;**"). Then, calculate a color for every vertex according to their angle defects. To do so, find the min and max values of all the angle defects, and map every angle defect value to a color from blue (min) to green (middle) to red (max). Show your results for double_torus.obj, mushroom.obj, and disk.obj. The results should look like:

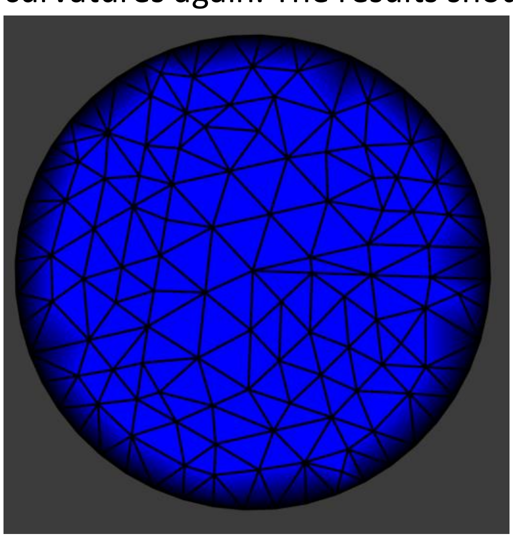
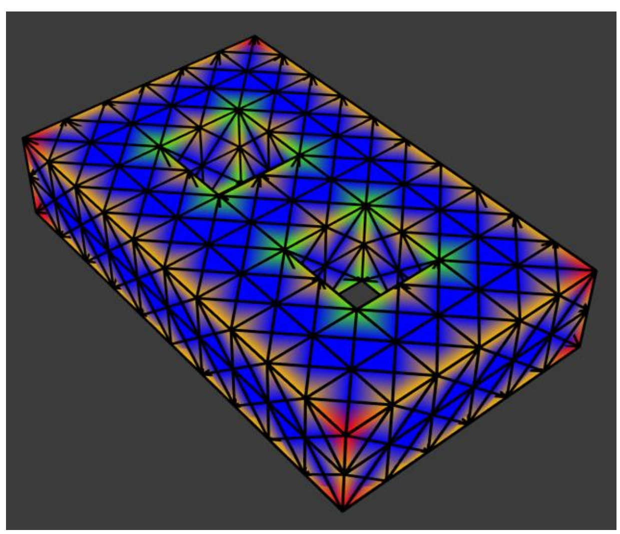




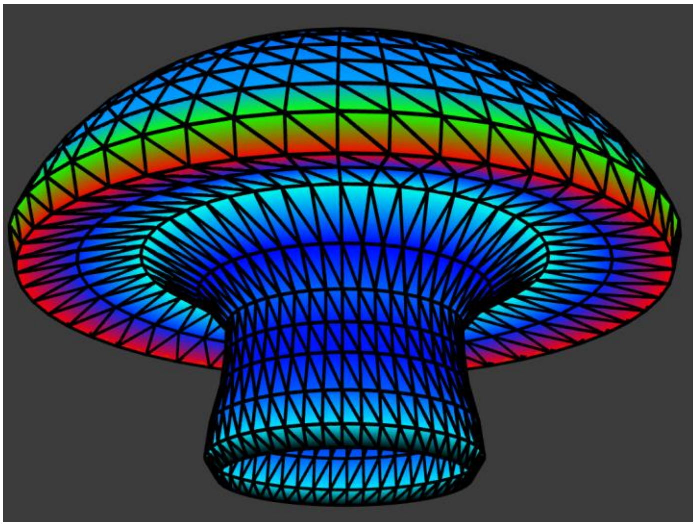
Q1: For each result, point to regions with positive and negative Gaussian curvatures, respectively (and some explanations will be great) in your video. Be careful for the disk.obj!

b) Calculate the **Laplacian vectors** of every vertices (i.e., they are used as proxies for per-vertex *discrete Mean Curvatures*). Skip boundary vertices. Use uniform-weight 1-ring neighborhood scheme. The per-vertex mean curvatures are approximated as the lengths of these Laplacian vectors. Consider round values very close to 0 to 0 (e.g., "**if abs(value)<1e-5, value = 0;**"). Then, calculate a color for every vertex according to their mean curvatures. To do so, find the min and max values of all the mean curvatures, and map every mean curvature value to a color from blue (min) to green (middle) to red (max). Show your results for double_torus.obj, mushroom.obj, and disk.obj. The results should look like:

(c) Now, use cotangent-weight Laplacian vector scheme and calculate the per-vertex mean curvatures again. The results should look like:



Q2: why the mean curvature values calculated by the two Laplacian schemes are very different for disk.obj? which one is more correct?

[Grading: 2 points for doing (a) correctly. 1 points each for doing (b) and (c) correctly each. 0.5 points each for answering Q1 and Q2 correctly each.]

Deliver your results as a video and your source codes.

0.5 bonus point for making a good video (clear and not overly long).

[Challenge] 1.0 bonus point for suggesting and implementing an even more efficient way to render the edges. (The current scheme involves creating a lot of "cylinder" meshes and becomes quite slow for big meshes. I don't like it but I couldn't find better ways to do so.)