8 Assignment P6 [Assignment ID: `subdivision`]

8.1 Preamble (Please Read Carefully)

Before starting work on this assignment, it is **critically important** that you **carefully** read Section 1 (titled "General Information") which starts on page 1 of this document.

You are **very strongly advised** to start working on this assignment as far in advance of the assignment submission deadline as possible. This assignment is not something that can be successfully completed in only a few days before the submission deadline. Start early enough to allow yourself sufficient time to seek help from the instructor (or teaching assistant) when you encounter problems. You have been warned.

In this assignment, **all of your code must be fully commented**. Marks will be deducted for code that is not fully commented. For examples of what constitutes fully commented code, please refer to the source code for the `wireframe` program in Listing D.5 (in Section D.11.1 on pages 512–524) of the textbook (2013-09-26 version).

8.2 Topics Covered

This assignment covers material primarily related to the following: OpenGL, CGAL, subdivision surfaces.

8.3 Part A (Geometry Processing with CGAL)

1. `triMeshInfo` program. Write a program called `triMeshInfo` that reads a mesh in OFF format from standard input and writes the following information about the mesh to standard output (in order):
   (a) The bounding box of the mesh, specified in terms of its minimum x coordinate, maximum x coordinate, minimum y coordinate, maximum y coordinate, minimum z coordinate, and maximum z coordinate. All of these quantities are to be output on the same line, separated by a single space character, with the last quantity being immediately followed by a newline character.
   (b) The volume of the bounding box. This quantity is to be output followed immediately by a newline character.
   (c) The minimum, maximum, and mean edge length. All of these quantities are to be output on the same line, separated by a single space character, with the last quantity being immediately followed by a newline character.
   (d) The minimum, maximum, and mean face area. All of these quantities are to be output on the same line, separated by a single space character, with the last quantity being immediately followed by a newline character.
   (e) The total surface area (i.e., the sum of the areas of all faces). This quantity is to be output followed immediately by a newline character.

   The program must not produce any other output to standard output other than that which is explicitly requested. Moreover, it is critical to follow the specified output formatting exactly (including the placement of newlines). When printing floating-point values, use the **default formatting and precision settings** (i.e., do not override the number of decimal places, etc.). The code need only support triangle meshes. Since the OFF format supports arbitrary polygonal meshes, you must ensure that the input mesh is a triangle mesh. The source code for the `triMeshInfo` program should be placed in a file called `triMeshInfo.cpp`.

   As an example of correct program behavior, consider a triangle mesh with the following characteristics:
   - bounding box $[-1,1] \times [-1,1] \times [0,1]$ (i.e., x range $\times$ y range $\times$ z range)
   - bounding box volume 4
   - minimum edge length 1.73205
   - maximum edge length 2
   - mean edge length 1.86603
   - minimum face area 1.41421
   - maximum face area 1.41421
   - mean face area 1.41421
   - total surface area 5.65685

   When the `triMeshInfo` program is run on this mesh, the output produced should be formatted exactly as follows:

```
-1 1 -1 1 0 1
4
1.73205 2 1.86603
1.41421 1.41421 1.41421
5.65685
```

You should use the `Polyhedron_3` class. Some potentially useful `Polyhedron_3` member functions and types include:

`is_pure_triangle`, `size_of_vertices`, `size_of_halfedges`, `size_of_facets`, `vertices_begin`, `vertices_end`, `edges_begin`, `edges_end`, `facets_begin`, `facets_end`, `Halfedge`, `Vertex_const_iterator`, `Vertex_iterator`, `Edge_const_iterator`, `Edge_iterator`, `Facet_const_iterator`, `Facet_iterator`

Some other potentially useful functions and types include:

`Vector_3`, `Point_3`, `squared_distance`

NOTE: The area of a triangle can be computed using Heron's formula. In particular, the area A of a triangle whose sides have lengths a, b, and c is

$$A = \sqrt{p(p-a)(p-b)(p-c)} \quad \text{where } p = \tfrac{1}{2}(a+b+c).$$

NOTE: Several polygon meshes in the OFF format can be downloaded from the course web site, including `tetrahedron.off`, `corner_quadmesh.off`, `bunny_999.off`, and `dragon_1000.off`.

2. `limitSurfInfo` **PROGRAM.** In this problem, we will use position and tangent masks for Loop subdivision in order to calculate the limit positions of vertices as well as their corresponding surface normals.

Using CGAL, write a program called `limitSurfInfo` that provides the functionality described below. The program should consist of a single source code file called `limitSurfInfo.cpp`. The program should read a polygon mesh in OFF format from standard input, and then compute (for Loop subdivision) the limit position and corresponding unit-norm surface normal for each vertex in the mesh. The appropriate position and tangent masks for Loop subdivision can be found in the textbook. The surface normal for a vertex v on the border of the mesh should simply be computed as the average of the unit normals of all faces incident on the vertex v (where the normals are oriented to point outward according to a CCW ordering and right-hand rule). For a detailed example showing how to compute surface normals for boundary vertices, refer to Example 9.11 on pages 391–394 of the textbook (version 2015-06-06), which is available as a separate handout since this version of the textbook is not publicly available. The program must accept the following command-line arguments in order:

(a) The x, y, and z coordinates of a query point p (with each coordinate specified as separate command-line argument).
(b) A search radius d.

For each vertex v that is within a distance d of the query point p (i.e., $\|v-p\| \leq d$), the program should write to standard output a single line containing the following information, with each field separated by a single space character:

(a) The x, y, and z coordinates of the original vertex v (with fields separated by a single space character).
(b) An indication of whether the original vertex v is on the border of the mesh. This value is an integer equal to 1 for a border vertex and 0 otherwise.
(c) The x, y, and z coordinates of the limit position of the vertex v (each separated by a single space character).
(d) The x, y, and z coordinates of a unit vector in the direction of the surface normal at the limit position of the vertex v (each separated by a single space character). The normal must be oriented in accordance with the **right-hand rule**.

When printing floating-point values, use the **default formatting and precision settings** (i.e., do not override the number of decimal places, etc.). For example, the output of the program might resemble the following:

```
-1 -1 0 1 -0.666667 -0.666667 0 -0.408248 -0.408248 0.816497
1 -1 0 1 0.666667 -0.666667 0 0.408248 -0.408248 0.816497
```

Some types/functions that might be useful include:

- `Polyhedron_3`, `is_pure_triangle`
- `Vertex_const_handle`, `Vertex_handle`, `Vertex_const_iterator`, `Vertex_iterator`

- `Vertex::Halfedge_around_vertex_const_circulator`, `Vertex::Halfedge_around_vertex_circulator`, `Vertex::vertex_begin`, `Vertex::vertex_end`, `Vertex::point`
- `Halfedge::is_border_edge`, `Halfedge_const_handle`, `Halfedge_handle`, `Halfedge::opposite`, `Halfedge::vertex`
- `ORIGIN`, `cross_product`, `squared_distance`, `SPL::normalize`

In order to solve this problem, you will probably want to store additional information in the vertices of the polyhedral mesh (e.g., limit position and surface normal). Since it is tricky (i.e., requires knowledge beyond the scope of this course) to specify the types to allow this, the type definitions have been provided for you. The type definitions are available from the course web site in the file `limitSurfInfo_partial.cpp`.

NOTE: Be careful about the order in which circulators visit items. Not all circulators visit items in CCW order. For example, `Facet::Halfedge_around_face_circulator` visits halfedges around a facet in CCW order, while `Halfedge::Halfedge_around_vertex_circulator` visits halfedges around a vertex in CW order. Of course, if the order is not what you desire, you can use the decrement operator instead of the increment operator to reverse the order.

8.4 Part B (Geometry Processing with OpenGL/GLUT)

1. `simple3` **PROGRAM.** In this problem, we will develop an OpenGL program to draw a few simple 3-dimensional geometric objects.

   Using the OpenGL and GLUT libraries, write a program called `simple3` that provides the functionality described below. The program should consist of a single source code file called `simple3.cpp`. The program should draw the following objects in a window with a nominal size of 1024×1024 pixels:

   (a) A cube having sides of unit length with its center at $(-1.5, 1.5, 0.0)$. This can be drawn using the `glutSolidCube` function. The color of the cube should be blue (i.e., RGB color value $(0.0, 0.0, 0.9)$).

   (b) A sphere of radius $\frac{1}{\sqrt{2}}$ and center $(0,0,0)$. This can be drawn using the `glutSolidSphere` function. The color of the sphere should be magenta (i.e., RGB color value $(0.9, 0.0, 0.9)$).

   (c) A tetrahedron with sides of length $\sqrt{2}$ centered at $(1.5, -1.5, 0.0)$. The color of the tetrahedron should be cyan (i.e., RGB color value $(0.0, 0.9, 0.9)$). You will need to write your own function to draw a tetrahedron. A tetrahedron with all sides of **unit length** has vertices v_0, v_1, v_2, v_3, and faces f_0, f_1, f_2, f_3 with the corresponding unit normals n_0, n_1, n_2, n_3, where

$$
\begin{gathered}
v_0 = (0.00000, 0.00000, 0.61237), \quad v_1 = (0.50000, 0.28867, -0.20412), \\
v_2 = (-0.50000, 0.28867, -0.20412), \quad v_3 = (0.00000, -0.57735, -0.20412), \\
f_0 = (v_0, v_1, v_2), \quad f_1 = (v_0, v_2, v_3), \quad f_2 = (v_0, v_3, v_1), \quad f_3 = (v_3, v_2, v_1), \\
n_0 = (0.00000, 0.942809, 0.333334), \quad n_1 = (-0.816497, -0.471405, 0.333333), \\
n_2 = (0.816497, -0.471405, 0.333333), \quad n_3 = (0.00000, 0.00000, -1.00000).
\end{gathered}
$$

   For each face f_i, the vertices are specified with a CCW orientation such that the corresponding unit normal n_i chosen by the right-hand rule will point in the outward direction.

   A diffuse light source should be positioned at an infinite distance in the direction $(0.25, 0.5, 1.0)$ with the color of the light being white (i.e., RGB color value (1.0, 1.0, 1.0)). The following viewing parameters should be used:

   - A perspective projection (as defined by `gluPerspective`) with a field of view of 70° and near and far clipping plane values of 1.0 and 100.0. The aspect ratio should be chosen to match the aspect ratio of the window.
   - An eye position of $(5\cos\theta, 5\sin\theta, \rho)$, scene center $(0,0,0)$, and an eye-up direction $(0,0,1)$ (as defined by `gluLookAt`). The quantities θ and ρ should initially be set to zero.

   The program should accept the following single key commands:

   - `q`. Quit the program.
   - `R`. Increase θ by 5°.
   - `r`. Decrease θ by 5°.
   - `Z`. Increase ρ by 0.1.
   - `z`. Decrease ρ by 0.1.
   - `i`. Reset θ and ρ to their initial values at program startup.

NOTE: The position of a light source is specified relative to the current viewing transformation. So, be sure to set your viewing transformation before setting the position of the light source.

NOTE: A video demonstrating a working solution to this problem can be found on the course web site. The graphics output produced by your program should look identical to what is shown in the video, including all color and shading.

8.5 Part C (Geometry Processing with CGAL and OpenGL/GLUT)

1. `triMeshView` **PROGRAM.** In this problem, we will develop an interactive mesh-viewing application that also allows Loop subdivision to be performed.

   Use the `wireframe` program as a starting point, as it provides much of the necessary user-interface code. The `wireframe` program is available from the course web site in the file `wireframe.zip`. The `triMeshView` program should consist of a single source code file called `triMeshView.cpp`. [Note: You will have to rename the file `wireframe.cpp` from the Zip file.] The program should read a polygon mesh in OFF format from standard input, and then display the mesh. The mesh should be rendered with triangle faces (not as a wireframe model). Lighting should be enabled and the face must be drawn with the correct normals specified. Smooth shading should be used. The mesh should be drawn using the technique discussed in the lectures. That is, each vertex in the mesh should be pushed to its limit position, and the surface normal at each vertex should be its corresponding surface normal on the limit surface. The computation of the limit positions and surface normals can be done in exactly the same way as in the `limitSurfInfo` program. The program should provide the following functionality:

   - Left drag (i.e., moving the mouse with the left button down) rotates the model.
   - Middle drag (i.e., moving the mouse with the middle mouse button down) scales the model.
   - Right drag (i.e., moving the mouse with the right mouse button down) translates the model.

   The following keyboard commands should be accepted:

   - `q`. Quit the program.
   - `S`. Increase the number of subdivision levels by one.
   - `s`. Decrease the number of subdivision levels by one.
   - `e`. Toggle the display of mesh edges.
   - `a`. Toggle the display of the axes.
   - `b`. Toggle the display of the arcball (for rotation).

   The number of subdivision levels being set to zero should be treated as no subdivision being applied. When the program is first run, the following settings should be in effect:

   - the number of levels of subdivision should be set to zero;
   - the display of mesh edges should be disabled;
   - the display of the coordinate axes should be enabled;
   - the display of the arcball should be enabled.

   In the interest of preventing excessive memory usage, the program may limit the maximum number of subdivision levels such that the face count of the highest-level refined mesh (i.e., the one with the most faces) does not exceed 100000. The `CGAL::Subdivision_method_3::Loop_subdivision` function can be used to perform Loop subdivision. That is, you need not implement the Loop subdivision algorithm yourself.

   NOTE: The most straightforward way to draw edges will result in some edges not being visible or having stippling artifacts. This is okay.

   NOTE: A video demonstrating a working solution to this problem can be found on the course web site.