Assignment #2

CS 200, Fall 2024

Due Friday, September 20

Mesh interface class

I will provide you with the file `Mesh.h`, which contains the following interface class (base class with pure virtual functions) declaration

```
namespace cs200 {
  struct Mesh {
    struct Face {
      unsigned index1, index2, index3;
      Face(int i=0, int j=0, int k=0)
        : index1(i), index2(j), index3(k) {}
    };

    struct Edge {
      unsigned index1, index2;
      Edge(int i=0, int j=0)
        : index1(i), index2(j) {}
    };

    virtual ~Mesh(void) {}
    virtual int vertexCount(void) const = 0;
    virtual const glm::vec4* vertexArray(void) const = 0;
    virtual glm::vec4 dimensions(void) const = 0;
    virtual glm::vec4 center(void) const = 0;
    virtual int faceCount(void) const = 0;
    virtual const Face* faceArray(void) const = 0;
    virtual int edgeCount(void) const = 0;
    virtual const Edge* edgeArray(void) const = 0;
  };
}
```

(the header files `glm/glm.hpp` and `Affine.h` have been included). Actual triangular meshes are created by deriving (publicly) from this class and implementing the pure virtual functions, which are described below.

Interface details

`~Mesh()` — (destructor) called when the mesh is destroyed. Unless your mesh makes use of dynamically allocated memory, you need not implement this function.

`vertexCount()` — returns the number of vertices in the vertex array of the mesh.

`vertexArray()` — returns a pointer to the vertex array of the mesh. The vertices are given in object coordinates.

`dimensions()` — returns the vector $\langle \Delta x, \Delta y \rangle$ that gives the dimensions of the (tight) axis–aligned bounding box that contains the mesh.

`center` — returns the center (C_x, C_y) of the axis–aligned bounding box of the object. Note that any vertex point (x, y, z) of the mesh satisfies

$$C_x - \tfrac{1}{2}\Delta x \leq x \leq C_x + \tfrac{1}{2}\Delta x \quad \text{and} \quad C_y - \tfrac{1}{2}\Delta y \leq y \leq C_y + \tfrac{1}{2}\Delta y$$

`faceCount()` — returns the number of triangular faces in the face list of the mesh.

`faceArray()` — returns a pointer to the face list of the mesh.

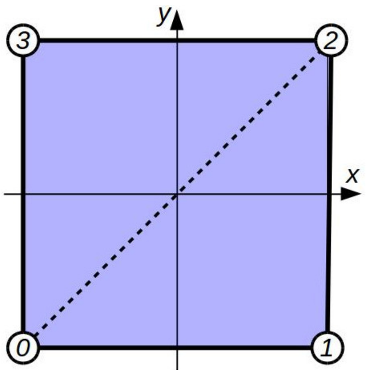
`edgeCount()` – returns the number of edges in the edge list of the mesh.

`edgeArray()` — returns a pointer to the edge list of the mesh.

An example mesh

As an example, suppose that our object is the standard square shown in blue below. In the figure, the dashed line between vertices 0 and 2 indicates how the square is to be triangulated. We would subclass the `Mesh` class as follows.

```
namespace cs200 {
  class SquareMesh : public Mesh {
    public:
      int vertexCount(void) const;
      const glm::vec4* vertexArray(void) const;
      glm::vec4 dimensions(void) const;
      glm::vec4 center(void) const;
      int faceCount(void) const;
      const Face* faceArray(void) const;
      int edgeCount(void) const;
      const Edge* edgeArray(void) const;
    private:
      static const glm::vec4 vertices[4];
      static const Face faces[2];
      static const Edge edges[4];
  };
}
```



The array `vertices` gives the vertex list of the triangular mesh that describes our object, and would be defined as:

```
const glm::vec4 cs200::SquareMesh::vertices[4]
  = { cs200::point(-1,-1), cs200::point(1,-1),
      cs200::point(1,1),   cs200::point(-1,1) };
```

(as indicated in the above figure). Similarly, the arrays faces and edges give, respectively, the face and edge lists of the mesh, and would be defined as

```
const cs200::Mesh::Face cs200::SquareMesh::faces[2]
  = { Face(0,1,2),  Face(0,2,3) };

const cs200::Mesh::Edge cs200::SquareMesh::edges[4]
  = { Edge(0,1), Edge(1,2), Edge(2,3), Edge(3,0) };
```

The functions vertexCount, faceCount, and edgeCount return the lengths of each of the above arrays. For instance, the function vertexCount would simply return the value 4. The functions vertexArray, faceArray, and edgeArray, return pointers to the desired array. E.g., we would define the function faceArray as

```
const cs200::Mesh::Face* cs200::SquareMesh::faceArray(void) {
  return faces;
}
```

Finally, the functions dimensions and center return the information about the axis–aligned bounding box of the object. In our case, the bounding box of the object is the square itself, which has width and height 2 centered at the point $(0, 0)$. Thus the function Dimensions would both return the vector $\langle 2, 2\rangle$, while the function Center would return the point $(0, 0)$.

Your task

Create your own triangular mesh. The only requirement is that the mesh that you design should be nontrivial: not a regular polygon, and should have at least 4 triangles. Bonus points will be awarded for artistic merit!

I will provide you with the files SquareMesh.h and SquareMesh.cpp which give the full declaration and implementation of the SquareMesh class. Feel free to use this code as a basis for your own mesh code.

What to turn in

Your submission for this assignment will consist of two files: (1) the interface file MyMesh.h, and (2) the implementation file MyMesh.cpp. You may only include the header files Mesh.h, MyMesh.h, and Affine.h, as well as any *standard* C++ header file. Your derived class **must** be named MyMesh, and should work without modification with the test driver MyMeshTest.cpp that I will provide.