24-780 B—Engineering Computation

Assigned: Wed. Sept. 23, 2020
Due: Thurs. Oct. 1, 2020, 11:59pm (A little extra time)

Problem Set 4: Mesh Viewer

Many computational analyses in engineering require the discretization of physical components into individually defined elements. This Finite Element Modeling (FEM) is used to break down the continuum that is "the universe" into chunks that can be handled internally as separate element, leaving the macro-scale behavior as a series of interactions between elements. For this assignment, we will create a simple graphical viewer for a two-dimensional finite element model. We will augment the viewer in future assignments.

In order to better represent our model, we will make use of object-oriented programming. A finite element model is composed of nodes to define the coordinates of the model, and elements that are configured to fit withing those nodes. In our simplified implementation, we will limit ourselves to quadrilateral plates in a planar configuration.

Plate Class

FEM plate will be stored as instances of the *Plate* class. All plates have a pointer reference to 4 nodes. Create a *Plate* class and provide the following:

Member variables (all private):

```
string label;
Node* nodes[4]; // pointers to each node (avoid data duplication)
```

Member Functions (all public):

Constructor for the class, setting label and initializing node pointers to nullptr. See attached Plate.h file to see that it is already coded.

```
Plate(const string& label);
```

Get and set for node pointers. See attached Plate.h file to see that this is already coded.

```
Node* getNode(int number);
void setNode(Node* newNode, int number);
```

Drawing function that draws outline of the plate using the current color. Includes the node label somewhat centered on the plate if showLabel parameter is true.The parameter *theModel* is used to access the *screenVertex()* function instead of using glVertex2i() directly. This allows drawing at any scale with any translation (panning).

```
void drawOutline(Model2D &theModel, bool showLabel = false);
```

Drawing function that shows a filled quadrilateral of the plate using the current color. Includes the plate label somewhat centered on the plate if showLabel parameter is true.The parameter *theModel* is used to access the *screenVertex()* function instead of using glVertex2i() directly. This allows drawing at any scale with any translation (panning).

```
void drawFill(Model2D &theModel);
```

Plate

Plate

Drawing function that shows a filled quadrilateral of the plate using the gradation to create a contour using the following rules for each corner color.

- if defl of node <= lowValue, use H = 240 (blue)
- if defl of node >= highValue, use H = 0 (red)
- otherwise, interpolate the defl of the node between low and high values to get corresponding color H between 240 and 0.

The parameter *theModel* is used to access the *screenVertex()* function instead of using glVertex2i() directly. This allows drawing at any scale with any translation (panning).

```
void drawFill(Model2D &theModel, double lowValue, double highValue);
```

Node

Node Class

FEM nodes will be stored as instances of the *Node* class. Create a *Node* class and provide the following:

Member variables (all private):

```
string label;
double x, y, z;   // coordinates of the node
double deflection;  // a preliminary result
```

Member Functions (all public):

Constructor for the class, setting all parameters

```
Node(const string &aLabel, double xCoord, double yCoord,
        double zCoord = 0., double defl = 0.);
```

Get and set for all member variables

(see attached Node.h file to see that they are already coded)

Drawing function that draws a symbol (programmer's choice) of the node, filled as given. Uses *size* (default = 1) to determine model size of node symbol. The parameter *theModel* is used to access the *screenVertex()* function instead of using glVertex2i() directly. This allows drawing at any scale with any translation (panning).

```
void draw(Model2D &theModel, int size = 10,
        bool filled = true, bool showLabel = false);
```

Model2D

Model2D Class

This class will hold the whole model together. Create a *Model2D* class and provide the following (noting that the attached Model2D.h file is a better guide than even this write-up):

Member variables (all private):

```
string name;
vector<Plate> thePlates;
vector<Node> theNodes;

// view preferences
NodeStyles nodeFormat;
bool showNodeLabels;
```

Model2D

```
PlateStyles plateFormat;
bool showPlateLabels;
bool showPlateOutline;

int nodeColor;  // H value for node color (S = 1, V = 1)
int plateColor; // H value for plate color (S = 1, V = 1)

int xOrigin = 0, yOrigin = 800;  // screen coords of model coords 0,0
double viewScale;  // must be greater than zero
```

Where the enumerated types are:

```
enum NodeStyles { hide, outline, filled };
enum PlateStyles { hide, outline, filled, contour };
```

Member Functions (all public):

Constructor for the class that reads through input file and creates nodes and plates as needed.

```
Model2D(ifstream &inFile);
```

Functions to allow the user the change the view preferences for the model.

```
void toggleNodeLabels();
void togglePlateLabels();

void nextNodeColor();
void nextPlateColor();

void nextNodeFormat();
void nextPlateFormat();
```

Get and set for origin and view scale (to allow for panning and zooming) variables. See attached Model2D.h file to see that they are already coded

```
void setOrigin(double newX, double newY);
void setScale(double newScale);

double getXOrigin() { return xOrigin; }
double getYOrigin() { return yOrigin; }
double getViewScale() { return viewScale; }
```

Drawing function that draws all items in FEM model, according to preferences

```
void draw();
```

Mesh Viewer

Write a program that makes use of the classes above to create a mesh viewer that can be used to visualize the mesh. Allow for the following functionality (in no particular order, but numbered for reference)

1. Press C on screen to clear all data (ask for confirmation on console)
2. Press L on screen to ask user (in console) for file name to load (this does not clear all previous data)
3. Press arrow keys on screen to pan model up/down/left/right
4. Press + on screen to zoom into model (make model appear bigger)
5. Press – on screen to zoom out of model (make model appear smaller)
6. Press N on screen to cycle through node style (see Model2D.h for enum of node display styles)
7. Press B on screen to cycle through 8 distinct node colors
8. Press V on screen to toggle node labels on/off
9. Press P on screen to cycle through plate styles (see Model2D.h for enum of plate display styles)
10. Press O (letter ohh) on screen to cycle through 8 distinct plate colors
11. Press I on screen to toggle plate labels on/off

Deliverables

7 files (zipped together):

Model2D.h, Model2D.cpp

Node.h, Node.cpp

Plate.h, Plate.cpp

ps04_meshviewer.cpp

Upload the zip file to the class Canvas page before the deadline (Thursday, Oct.1, 11:59pm).

Learning Objectives

Use of classes and objects in C++.

Creating and maintaining of sequential data structure (std::vector).

Introductory understanding of graphical data representation.

Reading data from a file and understanding how constructors can be effectively used to initialize a data model.

Implementing algorithms developed by others.