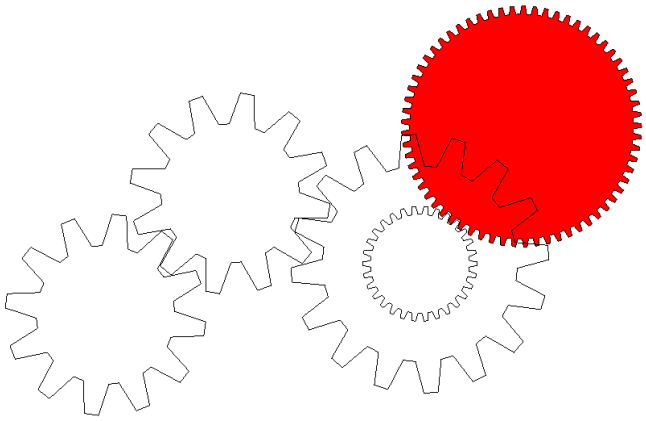


24-780 B—Engineering Computation

Assigned: Wed. Oct. 6, 2021
Due: Tues. Oct. 12, 2021, 11:59pm

Problem Set 5: Meshing Gears

Development Note

When working on assignments that build on previous work, I do not want you to start the current assignment at a disadvantage. For example, if you are not happy with your solution for PS04 for whatever reason, it would be awful if I asked you to build on that program. Thus, I am officially making clear that you have the following choice (now and for the rest of the semester):

- Take your own solution as the starting point.
  (advantage is that you carry through with your own assumptions and data model rather than trying to figure out mine)

OR

- Use my solution to the previous assignment from Piazza as the starting point.
  (advantage is that you "cut your losses" and are on the same page as everyone else)

In this assignment, we will add functionality to our gear simulator. Sadly, in keeping within the realm of what is possible in the time allotted, we will *not* be working too much on the accurate graphic display of gears. Instead, we will focus on getting gears to interact as they affect the *connected* gears around them. Some simplifying assumptions:

- Gears do not translate as part of the simulation (i.e., they are supported at their axles such that only rotation is possible)
- Gears can only affect each other in two ways:
  - They are *attached* at the center (i.e., they rotate at the same rate)
  - They are *meshed* at the teeth (i.e., they rotate in opposite directions as per their relative diameters or tooth-counts). Note only gears that have equal pitch can be meshed.
- The gearing system can actually work (i.e., gears do not lock up)

Task 1: Enhance data model

Gear

First, we need to add some data storage to *Gear* class so that we can keep track of other gears that are connected (note the pointer type):

```
vector<Gear *> attachedGears;
vector<Gear *> meshedGears;
```

We need to update the following functions for *Gear* so that rotation of the gear is "passed on" to connected gears

```
// rotate this gear by given amount
// and possibly cause the rotation of connected gears (attached and meshed)
// except do not rotate the gear that is requesting the rotation
void rotate(float rotAngle, Gear *requestingGear = nullptr);
```

Gear

You'll likely need a lot more get and set functions and even a cover-all function like:

```
      // goes through each property, showing existing values
      // and asking user to make changes
void edit();
```

Finally, we need functions like these:

```
      // create a bi-directional attach relationship between this gear and
      // otherGear, unless bi-directional is not appropriate
      // may require this gear's location to change to match otherGear's
      // returns false if connection cannot be done
bool attachToGear(Gear *otherGear, bool biDirectional = true);

      // create a bi-directional meshed relationship between this gear and
      // otherGear, unless bi-directional is not appropriate
      // may require this gear to translate (along line connecting centers)
      // and rotate to allow meshing, but do not move or rotate otherGear
      // returns false if connection cannot be done (unmatched pitches)
bool meshInto(Gear *otherGear, bool biDirectional = true);
```

Task 2: Formalize GearSystem class

GearSystem

In PS04, we let main() function get out of hand in that it both maintained the interface and the data model. For PS05, we want to create a *GearSystem* class that moves some of the functionality out of main. In particular, *GearSystem* has the following members and functions (mostly just moved from GearViewer.cpp or using stuff already in *Gear* class):

```
      vector<Gear *> allGears;
      int currGearIndex;
      int otherGearIndex;
```

With the following functions, plus any set and get functions you think you need:

```
            // draws all gears, highlighting current gear and optionally
            // highlighting otherGear.
      void draw(bool highlightOther = false);

            // displays helpful message on graphics window
      void inputRequiredMessage();

            // displays helpful message on graphics window
      void inputRequiredMessage();

            // rotates the current gear
      void rotateCurrent(float deg);

            // removes current gear, with confirmation
      void deleteCurrent();

            // edits current gear
      void editCurrent();

            // prints current gear to console
      void printCurrent();

            // changes the current gear, being careful to "loop around " as needed
      void changeCurrent(int change);
```

GearSystem

```
        // changes the other gear, being careful to "loop around " as needed
        // but skipping over current gear
    void changeOther(int change);

        // OPTIONAL:
        // changes the current gear based on given model coords
    void changeCurrent(Point2D givenCoords);

        // OPTIONAL:
        // changes the other gear based on given model coords
        // but skipping over current gear
    void changeOther(Point2D givenCoords);

        // asks for filename and loads a gear or gears,
        // adding it/them to system
    void addGear();

        // returns a pointer to the gear that matches the givenID,
        // returns nullptr if not found
    Gear *getGear(const std::string &gearID);

        // returns a pointer to the gear at given index,
        // returns nullptr if not found
    Gear *getGear(int gearIndex);

        // figures out best origin and scale for display
    void getOriginAndScale(float &originX, float &originY, float &scale);

        // attaches the current gear to the other gear
    bool attachGears();

        // meshes the current gear into the other gear
    bool meshGears();

        // asks for a file name and saves all gears in a single file
    void saveAll();
```

Task 3: Improve the Viewer

There are some extra features that the viewer needs now that we have extra functionality. Add the following features to the viewer (and to the console menu)

1. Edit gear: Pressing letter “E” on graphics window allows user to edit current gear
2. Attach gear: Pressing letter “T” on graphics window allows user to attach the current gear to another user-selected gear. The “otherGear” can be selected by ID or by click or by cycling through non-current gears
3. Mesh gear: Pressing letter “M” on graphics window allows user to mesh the current gear into another user-selected gear. The “otherGear” can be selected by ID or by click or by cycling through non-current gears
4. Save System: Pressing “S” on graphics window will save all gears to a single file

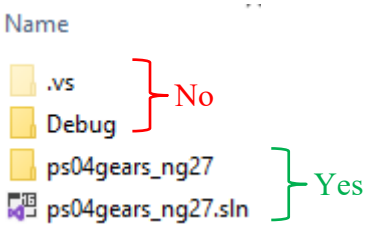
Deliverables

3 files (zipped together):

Gear.h, Gear.cpp	<< the files for *Gear* class
GearSystem.h GearSystem.cpp	<< the files for *GearSystem* class
GearViewer.cpp	<< start with mine and edit to your liking

Upload the zip file to the class Canvas page before the deadline (Tuesday, Oct. 12, 11:59pm).

Alternatively, if you are using Visual Studio, it may be easier to submit your entire solution rather than a collection of files. To do this, create a *zip file* of the whole project (the .sln file and the associated folder), being careful NOT to include the hidden folder called ".vs". This folder is used only to manage the IDE and is typically huge (100MB). Erasing or omitting it will just force Visual Studio to rebuild it when needed. The Debug folder should be kept out of the zip file too to avoid including executable files that some firewalls may disallow. *The name of the project should include your AndrewID*



Learning Objectives

Use of classes and objects in C++.

Pointers, pointers, pointers

Making use of console input/output

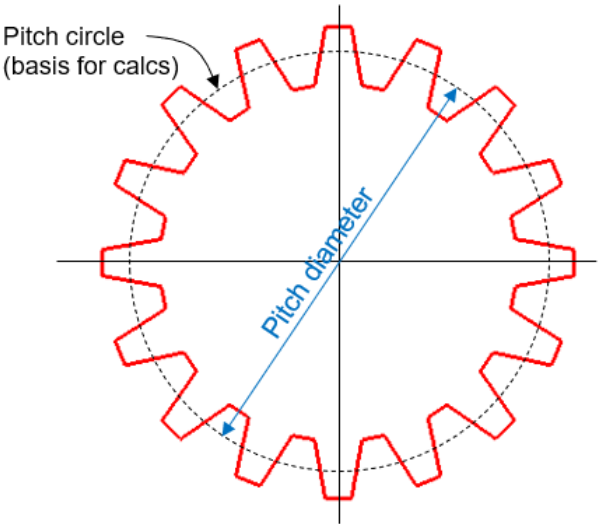
File input and output

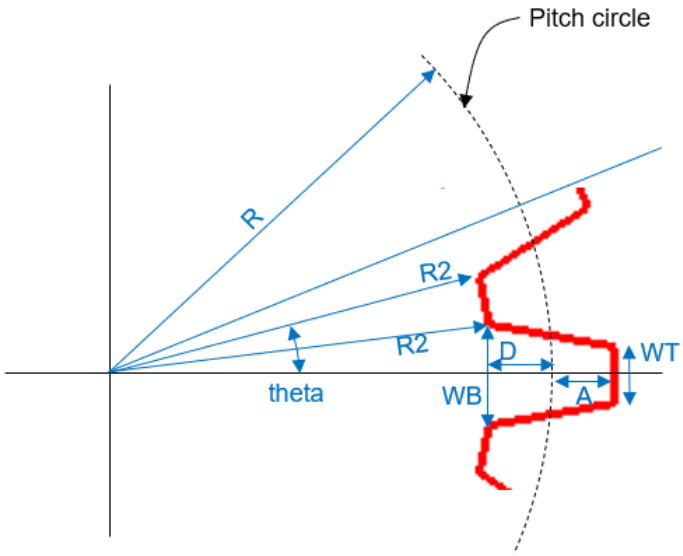
Using functions effectively

Searching references (online and/or textbook) for C++ library functions.

Gear Geometry

M = module = 1 / P
D = dedendum = 1.25 * M
A = addendum = 1.0 * M
WB = width at tooth bottom = 1.8 * M
WT = width at tooth top = 1.0 * M





N = number of teeth
P = pitch = teeth per inch
Pitch diameter = N / P

R2 = SRSS(WB/2, R-D)
theta = 360/N – atan(WB/2 / (R-D))