24783 Advanced Engineering Computation: Problem Set 7

(*) In the following instruction (and in all of the course materials), substitute your Andrew ID for where you see *yourAndrewId*.

START EARLY!

1 Check Out or Update Base Code and Libraries

Please make sure you have up-to-date libraries and course files before starting an assignment.

If you have not done working-directory set up as described in the first assignment (like in case you need to work from a different computer), please see Problem Set 1 and set up the working directory.

I assume you created the working directory called *24783* under you home directory and you checked out your Git repository in there.

Home directory is typically *C:\Users\username* in Windows, */Users/username* in macOS, and */home/username* in Linux, where *username* is the user name in your local computer.

First, open command-line (Developer PowerShell or Terminal), and move to your working directory by typing:

```
cd ~/24783
```

You need to check out (or clone) Git repositories once. If you have not checked out yet, do the following:

```
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/course_files.git
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

You need to replace "yourAndrewId" with your Andrew ID. You'll be asked to type in credentials.

Also we are going to use two additional repositories:

```
git clone https://github.com/captainys/MMLPlayer.git
git clone https://github.com/captainys/public.git
```

If you are successful, you should have the following directory structure under your home directory.

```
Your User Directory
├── (Other files and directories)
└── 24783
    └── course_files
```

```
└── yourAndrewID
```

If you already have checked out these repositories (most likely you did for Problem Set 1), you need to update (or git pull) in those repositories. By change directory to the location where you checked out repositries and then type:

```
git pull
```

To update all four repositories, you can type the following commands in a sequence:

```
cd ~/24783/course_files
git pull
cd ~/24783/yourAndrewID
git pull
cd ~/24783/public
git pull
cd ~/24783/MMLPlayer
git pull
```

2 Copy Base Code and Add to Git's Control

Copy ps7 subdirectory from course_files to your directory. The directory structure must look like:

```
Your User Directory
├── (Other files and directories)
└── 24783
    ├── course_files
    └── yourAndrewID
        └── ps7
```

3 Make a CMake Project

Write CMakeLists.txt for ps7. It is a graphical application, therefore make sure to use MACOSX_BUNDLE. The project name must be ps7. Case sensitive.

In this assignment, you do not have to make sub-directories. You need only one CMakeLists.txt. Make sure to include public libraries. ps7 must link ysclass and fssimplewindow libraries.

4 Parallel Bubble Packing (ps7.cpp)

In this problem set, you improve the performance of a program that packs bubbles inside a square domain $-1.0 <= x <= 1.0$ and $-1.0 <= y <= 1.0$ by multi threading.

The Bubble-Packing method was developed for generating a ideally distributed node locations generating finite element meshes. The method was fisrt published in the following paper:

Kenji Shimada and David C. Gossard, "Bubble Mesh : Automated Triangular Meshing of Non-manifold Geometry by Sphere Packing," In 3rd Shymposium on Solid Modeling and Applications, pp. 409-419, 1995

PDF of the paper is also available in the ps7 directory.

The method first packs bubbles in the target geometric domain, and then nodes are placed at the centers, which are connected to form finite elements.

To make this type of particle simulation, you typically need to do at least two things: (1) bucketing, and (2) multi threading. In this problem set, you do (2).

The single-threaded packing code is given as a base code. If you compile and run the base code, it packs bubbles (circles) inside the domain iteratively. If you press SPACE, it pauses and shows the Delaunay triangulation. However, it is very slow. Your goal is to make it faster by multi threading (use at least 8 threads). To be more specific, the bubble-packing method calculates the motions of the bubbles by:

1. Clear forces of each bubbles.
2. Compute inter-bubble forces acting between neighboring bubbles.
3. Compute damping forces acting on each bubble.
4. Numerical integrate velocity and position for a small time step.

The process repeats until bubble positions converge. The major computation comes from step 2. Therefore, multi-threading step 2 will gain substantial performance.

Each bubble has a member variable called f for calculating and storing forces acting on the bubble. While calculating inter-bubble forces, the program finds pairs of neighboring bubbles (in this example by exhaustively searching pairs because it is not using bucketing) and if found it calculates forces acting on the pair of bubbles, which are added to the data member f.

However, if you update member variable f in the multi-threaded environment, more than one thread may write to f simultaneously. Therefore, you first assign a group ID to each bubble. It makes the most sense to make groups based on the proximity, for example based on X coordinate.

Then, let each thread responsible for one group. If two bubbles belong to the same group of the thread, the thread can write to the data member f. However, if the bubble is in the different group, the write to the data member f needs to be postponed until multi-threading environment is over. Therefore, each thread needs to remember deferred calculation. When the multi-threading environment is over, add deferred inter-bubble forces.

Suggested steps are as follows (but you do not have to follow exactly):

1. Read the base code to understand the general structure.
2. Implement BubblePacking::AssignGroupId function. In this function, use bubble's X coordinate (bub.pos.x()) to calculate an integer value 0 to 7 and assign it as a group ID of the bubble.
3. Let BubblePacking::ComputeInterBubbleForces function take two additional parameters: int grouopId and a reference to std::vector ¡BubbleForcePair¿. BubbleForcePair class has an index to the bubble, and 2D vector that is a force to be added to the bubble. During the calculation, inter-bubble forces should be either directly added to the data member f of the pair of the neighboring bubbles, or deferred until the multi-threaded environment is over.
   In the base code, BubblePacking::ComputeInterBubbleForces runs a nested loop. Inside the loop, it deals with two bubbles, bubA and bubB. bubA is from the outer loop and bubB is from the inner loop.
   First, if bubA does not belong to the thread's group, skip it and move on to the next bubA.
   If bubA belongs to the thread's group, compute inter-bubble force for each bubB that is close to bubA, and add to bubA.
   Then if bubB belongs to the same group as bubA, add force to bubB.
   Else defer the addition of the force to bubB. Instead, add an item in std::vector ¡BubbleForcePair¿ so that the force can be added later.
4. Write ApplyBubbleForcePairs function. This function takes care of the deferred calculation.
5. Call BubblePacking::ComputeInterBubbleForces from the multiple threads. Use at least 8 threads. Please also see bounce.cpp that we did in class. The structure is similar to bounce.cpp. Use condition variables for starting and synchronizing threads.

5 Test Your Code on the Compiler Server

Test your source files (.cpp and .h files) on the compiler server. Some assignment may not require .h files. You do not have to test files that you don't make modifications. The files you need to test are the ones you write or modify.

We have four compiler servers:

- http://freefood1.lan.local.cmu.edu
- http://freefood2.lan.local.cmu.edu
- http://freefood3.lan.local.cmu.edu
- http://freefood4.lan.local.cmu.edu

Make sure you don't see red lines when you select your files and hit "Compile Test" button on the server.

We have multiple servers to make it less likely that all of them need to shut down for maintenance. If do not have to test on all of the servers. You need to make sure that your code passes on one of the servers.

6 Submit

Lastly, you need to submit using git. What you need to do are two things: (1) add files to git's control, and then (2) send to the git server.

6.1 Add Files to git's control

In this case, you want to add all the files under ps7 subdirectory. To do so, type:

```
git add ~/24783/yourAndrewID/ps7
```

This command will add ps7 directory and all files under the subdirectories.

6.2 Send to the Git Server

In Git, sending files to the server is a two-step process. The first step is local commit. You can do it by:

```
git commit -m "Problem Set 7 solution"
```

The message can be anything, but it is recommended to type something meaningful, at least you can see what changes you made to your repository.

Local commit is just local. Git server does not know about any local commit unless the commit is sent (or pushed) to the server. To do so, type:

```
git push
```

Make sure to do it in the CMU network. If you are working from home (probably most likely), use VPN to connect to the CMU network.

You can re-submit (commit and push) your solution as many times as you want with no penalty before the submission due.

7 Verification

It is recommended to clone your repository to a different location and make sure that all of your files have been sent to the Git server.

You can do the following:

```
cd ~
mkdir 24783Verify
cd 24783Verify
git clone https://yourAndrewID@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

Once you made sure all the files have been submitted, you can delete files and directories under 24783Verify directory.