Normalizing Body Positions

Programming Assignment #2
CS253 Spring 2014
Due 6:00pm Wednesday, February 10th, 2016

Motivation

In PA1, you read in sequences of body positions from a Microsoft Kinect II sensor, and displayed them to the screen (using the PoseDisplay class provided). The display program assumed the coordinates of body points were always within a cube that ran from -1 to 1 in all three dimensions, and we conveniently provided data in this range. This is not the range of the Kinect II sensor, however. For PA2, you will extend your PA1 assignment so that is normalizes the data to be within the -1 to 1 cube.

Pedagogically, the first assignment was about I/O and writing your first C++ program. You had to learn about splitting objects into .h and .cpp files, writing a main function, using the g++ compiler, etc. For this assignment, you have to extend your program to manipulate persistent data, namely the sequence of poses. This means your program gets larger, and the operations on the data get more complex. In particular, you have to manipulate data at the level of 3D points, Poses (i.e. ordered sets of 25 points), and Pose sequences. There is now more reward for defining multiple classes. At the same time, it is important to think about when data should and should not be modified, and therefore the difference between call by value function arguments and call by reference arguments.

Task

The command line arguments to PA2 are the same as the command line arguments to PA1, and the format of the input and output files are the same (although the range of numbers in the input files will now be different). The difference is that in PA2 your program will normalize the poses before displaying and outputting them.

There are two steps to normalizing the data. The first is to translate the poses in the video such that the average coordinate of the base of the spine is (0, 0, 0). In the Kinect II output format, the base of the spine is the 1st of the 25 points in every pose[1]. Therefore, you should compute the average (x, y, z) coordinate of all the 1st points of poses in the input data, and then subtract this point from every point in

[1] If you are interested in the body parts that correspond to each of the 25 points in a pose, look at the implementation of the PoseDisplay::InitializeSkeleton() method in PoseDisplay.cpp

every pose. The result will be a pose sequence in which the average position of the base of the spine is (0, 0, 0).

The second normalization step is to scale the data so that it is neither too big nor too small. To do this, compute the maximum of the absolute values of all the coordinates across all the points in all of the poses in the sequence, after you have translated the data. If the data is already the right size, this number should be 1. Otherwise, the scale factor is 1.0 divided by this maximum number. Multiply every coordinate by this scale factor to resize your data.

As always, never trust a user or a file. In addition to the data format errors possible from PA1 (which we may continue to test with), there are now new error cases. As before, if the input is not legal, your program should print an error message to std::cerr (not std:cout) and return the value -1 from main.

Submitting your program

You will submit your program through Canvas. You should submit a single tar file, and this file should contain all the source files of your program *and a makefile*, but no object or executable files. The makefile should create an executable program called PA2. To grade your programs, the GTAs will write a script that untars your file in a clean directory, runs 'make', and then tests PA2 on novel inputs. If your program does not compile, whether because of an error in your makefile or an error in your code, you will receive no credit on any of the test cases. Note that as always, we will test your code on the department Linux machines.

Hints

1. After normalization, you know what properties the pose sequence should have. Test for them.
2. You are allowed to extend, modify and in other ways change the Point3D class. In fact, I recommend it. Just make sure that the X(), Y(), and Z() methods remain, as the PoseDisplay class depends on them. (I do not recommend changing PoseDisplay.)
3. Un-tar your submission in a clean directory, make it and test it before you submit. This can catch a lot of unfortunate mistakes.
4. It is better to have many short files than a few long ones, and keep main.cpp as short as possible.