CS 548: Assignment 01

Programming Assignments (95%)

python/Assign01.py

This Python script will contain the following functions:

- **def load_pcd(filename)**
    - This will read in a file in version 0.7 of the PCD format and return it as a (legacy) Open3D PointCloud.
    - PCD File Format Documentation: https://pointclouds.org/documentation/tutorials/pcd_file_format.html
    - You may make the following simplifications/assumptions:
        - The file is assumed to be in the correct format (e.g., no error checking is necessary).
        - Header section
            - You SHOULD be able to skip over comment lines (first character "#") or blank lines in the header.
            - The following lines will need to be processed:
                - FIELDS
                    - You can expect the following possible fields:
                        - x
                        - y
                        - z
                        - normal_x
                        - normal_y
                        - normal_z
                        - rgb
                    - rgb represents a SINGLE value.
                    - You may assume all values are a 32-bit float.
                    - The ORDERING of these fields may be arbitrary.
                - POINTS
                    - You may use this to read in the number of points (thus you can ignore the WIDTH and HEIGHT fields).
                - DATA
                    - You need only find this to determine where the data section starts.

        - Data section
            - You may assume that the data is in ascii format (as opposed to binary or binary_compressed).
            - You may assume there are no comment lines or blank lines in the data section.
            - Remember that rgb is stored as a float. You will want to do the following:
                - Store the value as a numpy array with dtype=np.float32
                - Use the view() function to interpret it as a 32-bit unsigned int:
                    - rgb_i = rgb_f.view(np.uint32)
                - The individual RGB values are stored in different bytes:
                    - blue → lowest order byte
                    - green
                    - red
                    - (padding) → highest order byte
                - Use bit-shifting and masking to extract the individual channels.
                - Each channel has value range [0,255]; convert to float and rescale to [0,1].
                - Store floating-point color values.
            - In terms of the storage within your point cloud, you can assume numpy arrays of shape (point count, 3) of dtype=np.float32.
                - These will need to be converted via open3d.utility.Vector3dVector() before storage in the PointCloud object.
    - You are free to implement this in any reasonable fashion, EXCEPT you CANNOT use any Open3D loading functions!
        - You can of course use the no-arg constructor for PointCloud and the open3d.utility.Vector3dVector() conversion function.
- **def main():**
    - If the number of command line arguments (len(sys.argv)) is less than 2, print an error and exit(1)
    - Read the input file path from sys.argv[1]
    - Load the PointCloud using your load_pcd function.
    - Visualize your PointCloud with this function:
        - **o3d.visualization.draw_geometries([pcd], point_show_normal=pcd.has_normals())**

In addition, have the customary main function call on the bottom of your program:

if __name__ == "__main__": main()

src/include/PCD.hpp

You should have the following includes:

```
#pragma once
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/common/common.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <cmath>
#include <cstring>
#include <unordered_map>
using namespace std;
```

This file will also include the prototype for one function:

pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr loadPCD(string filename);

src/lib/PCD.cpp

Include "PCD.hpp" and define the following:

- **pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr loadPCD(string filename)**
  - Create a Ptr to a PointCloud<PointXYZRGBNormal>.
  - Open the file identified by filename for reading.
  - If the file fails to open, print an error message and return nullptr.
  - The logic and rules for reading this file are similar to the Python version with two notable exceptions:
    - You will have to set the width and height fields of the point cloud (width = number of points, height = 1)
    - The rgb values stored in the points will REMAIN unsigned integer-like values in the range [0,255]
  - Remember to close the file once you are done.
  - As will the Python version, you are largely free to make any reasonable implementation that passes the tests, EXCEPT that you cannot use PCL's file loading functionality!
  - Return the Ptr to the point cloud.

src/app/Assign01.cpp

In the main function:

- If argc is less than 2, print an error message and exit.
- Load the cloud from the file at string(argv[1]) using your loadPCD function.
- Create a PCLVisualizer with:
    - A gray background (0.7, 0.7, 0.7)
    - Your point cloud
    - Your point cloud's normals (every 10th point, length of 0.01, black)
- Run the visualizer.

Testing Screenshot (5%)

I have provided several files for testing:

- data/assign01
    - BunnyXYZ.pcd
    - BunnyXYZN.pcd
    - BunnyXYZNRGB.pcd
    - BunnyXYZRGB.pcd
- python/
    - Test_Assign01.py – the test program for the Python code
- src/
    - include/
        - doctest.h
    - tests/
        - Test_Assign01.cpp – the test program for the C++ code
- CMakeLists.txt – updates to include testing

Run the testing program through the testing section of Visual Code.

You MUST run the tests and send a screenshot of the test results! Even if your program(s) do not pass all the tests, you MUST send this screenshot!

None of the tests check the main functions; I will test these manually.

Python Tests

You may have to do "Command Palette" → "Python: Configure Tests" → pytest → python (directory)

You should then be able to run the Python tests in your testing window in Visual Code.

ALTERNATIVELY: open a terminal and enter: **pytest python/Test_Assign01.py**

…then take a screenshot of the terminal output.

C++ Tests

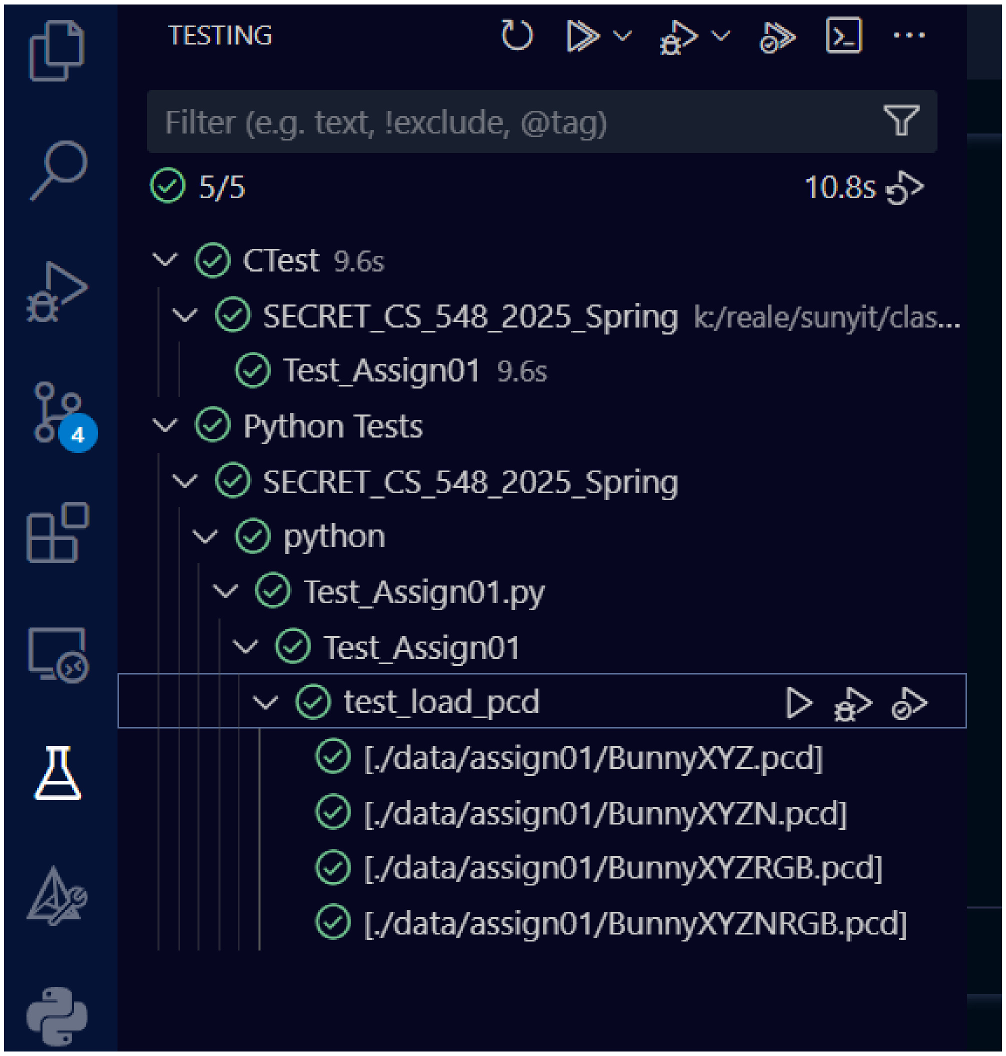
To run the tests and get the checkboxes → run the tests in NON-debug mode from the testing window.

To run the tests with debugging:

- Run individual tests (or all tests) in DEBUG mode from the testing window.
- Select the "Test" configuration appropriate for your OS when it pops up (e.g., "(Windows) Test" as opposed to "(Windows) Launch").
- You may have to select this configuration multiple times if you run all tests at once.
- Note that in debug mode, the checkboxes may not show up correctly, but it should be easier to step through issues.

Screenshot Example

This screenshot should show clearly the testing view in Visual Code:



```
✓ 5/5                                              10.8s
∨ ✓ CTest 9.6s
  ∨ ✓ SECRET_CS_548_2025_Spring k:/reale/sunyit/clas...
      ✓ Test_Assign01 9.6s
∨ ✓ Python Tests
  ∨ ✓ SECRET_CS_548_2025_Spring
    ∨ ✓ python
      ∨ ✓ Test_Assign01.py
        ∨ ✓ Test_Assign01
          ∨ ✓ test_load_pcd
              ✓ [./data/assign01/BunnyXYZ.pcd]
              ✓ [./data/assign01/BunnyXYZN.pcd]
              ✓ [./data/assign01/BunnyXYZRGB.pcd]
              ✓ [./data/assign01/BunnyXYZNRGB.pcd]
```

Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 95% - Programming assignments

I reserve the right to take points off for not meeting the specifications in this assignment description. In general, these are things that will be penalized:

- **Code that is not syntactically correct (up to 60 points off!)**
- Sloppy or poor coding style
- Bad coding design principles
- Code that crashes, does not run, or takes a VERY long time to complete
- Using code from ANY source other than the course materials
- Collaboration on code of ANY kind; this is an INDIVIDUAL PROJECT
- Sharing code with other people in this class or using code from this or any other related class
- Output that is incorrect
- Algorithms/implementations that are incorrect
- Submitting improper files
- Failing to submit ALL required files

Hints

C++ Strings

Strings in C++ can be compared with ==

if(token == "DATA") { … }

C++ Bit Manipulation

You can shift things left and right in C++. For example, to shift two bytes to the right (puts it lower):

int new_val = (value >> 16);

You can also use a bit-wise AND to mask values; the following only keeps the lowest order byte:

int new_val = value & 0x000000FF;

Handling the Fields

To keep track of the fields and ordering, I recommend a **dictionary** in Python and an **unordered_map<string, int>** in C++. This will allow you to connect field to index.

You can check for the presence of a key in an unordered_map via count(): **if(myMap.count("x") > 0) {…}**

C++ Vectors

You can create a resizable list in C++ using a vector:

vector<string> myNames;
myNames.append("Bob");
int cnt = myNames.size();   // Returns 1
string n = myNames.at(0);

You can also allocate space ahead of time:

vector<string> myNames;
myNames.resize(10);   // myNames now has 10 items
myNames[2] = "Joe";

C++ File IO

Opening, checking, reading, and closing a file (as well as creating a string stream to parse a string):

```
ifstream file(filename);
// Is file open?
if(!file) {
    cerr << "ERROR: Could not open " << filename << "!" << endl;
    return nullptr;
}

string line;
while(getline(file, line)) {
    // Create string stream
    stringstream ss(line);
    // Read in token
    string token;
    ss >> token;
    // Read in float
    float f;
    ss >> f;
    // While it still has tokens, keep reading a new token
    while(ss >> token) {
       // Do stuff
    }
}

// Close file
file.close();
```