Programming Assignments 3 and 4 – 600.445/645 Fall 2016

Score Sheet (hand in with report) Also, PLEASE INDICATE WHETHER YOU ARE IN 600.445 or 600.645

Name 1	Doran Walsten
Email	dwalste1@jhu.edu
Other contact information (optional)	
Name 2	Ravi Gaddipati
Email	ravigaddipati@gmail.com
Other contact information (optional)	
Signature (required)	I (we) have followed the rules in completing this assignment *Doran Walsten* *Ravi Gaddipati*

Grade Factor		
Program (40)		
Design and overall program structure	20	
Reusability and modularity	10	
Clarity of documentation and programming	10	
Results (20)		
Correctness and completeness	20	
Report (40)		
Description of formulation and algorithmic approach	15	
Overview of program	10	
Discussion of validation approach	5	
Discussion of results	10	
TOTAL	100	

600.445 Computer Integrated Surgery
Programming Assignment 3

Ravindra Gaddipati, Doran Walsten

November 17, 2016

Closest Point

1.1 Mathematical Approach

On the math side, this assignment can be broken down into two pieces: registering the tip of the pointer to the reference frame of the fixed rigid body and computing the closest point on a surface mesh to a point in space.

1.1.1 Registration from Pointer Coordinate frame to Fixed Body frame

As outlined in the assignment, we are provided data representing the location of LED markers on both a pointer and a fixed rigid body relative to the coordinate system of the objects themselves. We will call these point clouds A and B for the pointer and fixed body frames respectively. In addition, we are given frames of data $a_{i,k}$ and $b_{i,k}$ representing measurements of these points relative to the optical measurement device. Using a point cloud to point cloud transformation between the point cloud A and the points $a_{i,k}$, we can find the transformation $F_{A,k}$ from pointer coordinates to optical tracker coordinates on each frame k. The same can be done for $b_{i,k}$ and B to generate the transformation $F_{B,k}$ from fixed body coordinates to optical tracker coordinates. Thus, to determine the location of the tip of the pointer $\vec{A}_{tip}$ in fixed body coordinates on each frame, we compute:

$$\vec{d_k} = F_{B,k}^{-1} F_{A,k} \vec{A}_{tip}$$

We then determine the closest points on the mesh using the math described below. We first compute the distance to triangles on the mesh then determine which triangle is closest.

1.1.2 Closest Point On Triangle within Mesh

We chose to use the method proposed by Wolfgang Heidrich [1] due to its efficiency and ease of implementation. This method computes barycentric coordinates of the projection using the normal (or area) of the faces of the pyramid formed by the triangle T and point of interest P. T is represented as 3 vertices V_1, V_2, and V_3. We express this triangle as a point and two edges.

$$u = V_2 - V_1 \tag{1.1}$$

$$v = V_3 - V_1 \tag{1.2}$$

To complete the pyramid we define a new edge with respect to P.

$$w = P - V_1 \tag{1.3}$$

[1] Heidrich, W. (2005). Computing the Barycentric Coordinates of a Projected Point. Journal of Graphics, GPU, and Game Tools, 10(3), 9-12. doi:10.1080/2151237x.2005.10129200

With the three adjacent faces we can compute the barycentric coordinates as

$$b_2 = \frac{(\vec{u} \times \vec{w}) \cdot \vec{n}}{\vec{n}^2} \tag{1.4}$$

$$b_1 = \frac{(\vec{w} \times \vec{v}) \cdot \vec{n}}{\vec{n}^2} \tag{1.5}$$

$$b_0 = 1 - b_0 - b_1 \tag{1.6}$$

where $\vec{n} = \vec{u} \times \vec{v}$. The projected point can be computed from the barycentric coordinates as $P' = b_0 V_1 + b_1 V_2 + b_2 V_3$. If the projected point lies within the triangle, than $0 \leq b_{0,1,2}$. If this condition is not met, we need to project the point onto the edges to determine the closest point.

$$\lambda = \frac{(c - p) \cdot (q - p)}{(q - p) \cdot (q - p)} \tag{1.7}$$

$$\lambda^* = \max(0, \min(\lambda, 1)) \tag{1.8}$$

$$c^* = p + \lambda^* \cdot (q - p) \tag{1.9}$$

Where y^* is the constrained distance from point p. The signs of the barycentric coordinates are used to determine the edge to project onto. If the projected point lies opposite of the edge corresponding to the barycentric coordinates' vertex, the coordinate will be negative. If two coordinates are negative, than the closest point corresponds to a vertex. In these cases we can avoid further computation and return a vertex.

1.2 Algorithmic

Our approach was to first use the registration method above to compute the location of the pointer tip in fixed body coordinates in all frames and store this as a point cloud. We then performed a linear search of the mesh on each frame of data to find the closest point on the mesh. We compute the closest point on the triangle on the given iteration, compare this distance to the minimum seen so far, then update if needed. We seek to optimize this search in the next assignment using a k-d tree.

1.3 Software Overview

Eigen(https://eigen.tuxfamily.org/) was used as the central numerical library. There are some common objects used throughout the codebase, described in the following files.

1.3.1 Point Clouds and Points

`pointcloud.h`
`Note: This was reused from PA1,2`

Provides a `PointCloud` object, internally representing the point cloud as an Eigen::Array, were each row is a point. The object provides various functions to easily transform the point cloud and compute the centroid. Tests developed for the class include the computation of the centroid, transformation, and point insertion. A `Point` is an alias for a matrix of size 3 by 1, or `Eigen::Matrix<T,3,1>`.

- `Point at(size_t i)` : Returns the ith point.
- `Point centroid()` : Compute the centroid.

- `PointCloud center()` : Returns a new `PointCloud` with its centroid at $(0, 0, 0)$
- `PointCloud center_self()` : Subtract the centroid from all points.
- `void transform(Eigen::Transform)` : Applies the transformation to each point.

1.3.2 File Parsers

`files.h`

This provides classes used to represent the different types of file inputs and outputs to expect within the assignment. These include `RigidBody` for the representation of the local rigid body reference frames, `SampleReadings` for the incoming frames of LED optical tracker data, and `SurfaceFile` for the surface mesh files.

1.3.3 Iterative Closest Point Functions

`icp.h`

This file contains function definitions for everything needed to compute the distance from a point in space to a surface. This is needed during the ICP algorithm to determine the closest point on the surface. Key functions include:

- `Point project_onto_segment(const Point c, const Point p, const Point q)` : Project a point onto a line
- `Point project_onto_triangle(const Point p, const Point v1, const Point v2, const Point v3)` : Use the project onto a segment method as well as barycentric coordinates to determine which point on a triangle is closest to the given point
- `Point project_onto_surface_naive(const Point p, const SurfaceFile surface)` : Use a simple linear search to determine the closest point on a mesh surface by computing the distance to each triangle in the mesh.

1.3.4 Registration

`registration.h`

This file defines one function,

```
cis::PointCloud
pointer_to_fixed(const cis::RigidBody pointerRef,
const cis::RigidBody fixedRef,
const std::vector<cis::PointCloud> pointerframes,
const std::vector<cis::PointCloud> fixedframes)
```

which takes in the two rigid body representations and uses the horn method defined in `horn.h` from the previous assignment to compute the registration of the pointer tip to the fixed body reference frame in all frames of data provided.

1.3.5 Utilities

`utils.h utils.cpp`

These files contain general purpose functions. The main funciton used here is `split(str, delim)`, which splits a string by a delimiter and returns a vector.

1.4 Verification

We continued to use doctest (`github.com/onqtam/doctest`) to generate test cases within our newly defined classes to assure that they are functional. While the code was implemented, unit tests were used to verify functionality. With Doctest, test cases are outlined for each function at the bottom of their respective header files. The structure is as follows.

```
TEST_CASE("Description") {
[Preamble]
    SUBCASE {
        ex. Test Assertions
        CHECK(LHS == RHS)
        CHECK_THROWS(expr)
    }
    [Conclusion]
}
```

During the execution the preamble is executed for each subcase, followed by the test, followed by the conclusion. To execute all the tests, the argument "test" can be passed to the main executable. Specific tests implementations are outlined below. After unit tests functionality was tested on the debug files.

1.4.1 Point cloud to point cloud registration (horn.h)

In our submission for the past assignments, we were told that plugging in a specific point cloud, transform it, then confirm that our point cloud to point cloud registration method gets a similar transformation is not enough. A more thorough test would consist of testing over a variety of randomly generated point clouds and well as randomly generated transformation. Consequently, we did so in this assignment to verify that our Horn method is functional. We completed the steps above a total of 4 iterations and had success on every test case.

1.4.2 Closest Point Calculations (icp.h)

Within the `icp.h` file, we define all methods related to projecting a point onto a line and onto a triangle using barycentric coordinates. The goal of testing was to ensure in every projection case is accounted for. Sample points were picked such that the correct projection would like inside the triangle, on each edge of the triangle, on each vertex of the triangle, and each exterior space formed by the extension of the triangle edges. Within the "Segment Projection" test cases, we compare the expected results of a projection onto a line segment with the actual results.

1.4.3 New Data files (files.h)

In this assignment, we were given rigid body point clouds in their own reference frame, frames of data of the point clouds relative to an optical tracker, and a surface mesh file. We designed classes within `files.h` to store this formatted data. Our testing methods ensured that the chosen data structures functioned properly.

For the rigid body data, we simply parsed a sample data file and ensured that the proper point cloud and tip was stored. For the surface data, we parsed a sample data file and checked that points were interpreted correctly. We also check that the points are bounded for the vertex and triangle indcies via asserts.

For the sample readings data, we generated a sample data file including dummy data points to ensure that the parser correctly assigned points to the correct bins for the pointer data $a_{i,k}$ and the fixed body data $b_{i,k}$. This was done by simply ensuring that the points are equal.

1.5 Results

After our tests, we compared our results with those of the debug files provided. The results are as follows.

```
A
  d_k average error: 0.00682843
  c_k average error: 0
  Difference average error: 0.00153333

B
  d_k average error: 0.006
  c_k average error: 0
  Difference average error: 0.003

C
  d_k average error: 0.00655228
  c_k average error: 0
  Difference average error: 0.00226666

D
  d_k average error: 0.0084379
  c_k average error: 0
  Difference average error: 0.00286667

E
  d_k average error: 0.0113807
  c_k average error: 0
  Difference average error: 0.00299996

F
  d_k average error: 0.00949509
  c_k average error: 0
  Difference average error: 0.00393332
```

All of our outputs match the provided answers.

1.6 Discussion

Within the assignment, we were able to successfully demonstrate a functional method to both register pointer coordinates to the fixed rigid body frame as well as compute the closest point in a mesh to these points after completing a naive registration to the CT reference frame. This assignment is simply a bridge into the next assignment where the ultimate goal is to actually determine the true registration between the fixed body frame and the CT reference frame. Our goal is to implement a more efficient data structure such as the k-d tree to better identify the closest point in the mesh in the next assignment as well as actually implement the ICP method to determine the registration.

1.7 Usage

The codebase is built with cmake. Inside the project directory,

```
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release .. && make
```

The executable is also included.

```
e.x:

bin/cisicp INPUT/Problem3MeshFile.sur INPUT/Problem3-BodyA.txt
    INPUT/Problem3-BodyB.txt INPUT/PA3-G-Debug-SampleReadingsTest.txt
    OUTPUT/PA3-G-Output.txt

bin/cisicp test
```

The CLI usage information is printed if no arguments are passed.

This program uses Eigen (eigen.tuxfamily.org) for linear algebra and numerical methods as well as doctest (github.com/onqtam/doctest) to implement test cases.

1.8 Contributions

Ravi primarily worked on the triangle projection, and Doran mainly on the registration. Both contributed to developing the methods and testing.