A SOFTWARE TEST-BED FOR THE REGISTRATION OF 3D RANGE IMAGES

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Bachelor of Science with Distinction at

The Ohio State University

By

Gerald Dalley

*****

The Ohio State University

2000

Honors Distinction Committee:

Dr. Patrick J. Flynn, Adviser

Dr. Kim Boyer

Approved by

____

Adviser

Department of
Electrical Engineering

© Copyright by

Gerald Dalley

2000

ABSTRACT

Building 3D models of real-world objects by assembling views taken by a range sensor promises to be a more efficient method than manually producing CAD drawings. A series of range images are acquired, then registered, or aligned with each other to a high degree of accuracy. Finally, the range images are merged to form a complete 3D model.

Many techniques have been proposed for solving the registration problem; however, little work has been done to date to compare several registration algorithms with the same sets of data. In this thesis, we examine a software test-bed built for performing such analyses. Within this test-bed, we have implemented several common registration algorithm variants and tested them on four different sets of range images.

Through a set of quality criteria we propose, we show several features of the variants we have implemented. For example, a base Iterative Closest Point (ICP) algorithm performs very well when nearly all of the data is overlapping in the range images. For range image pairs that have large non-overlapping regions, an outlier classifier is required for better results. We also examine the effects of uniform decimation and ICP exit criteria values.

Our test-bed is extensible and allows researchers to add new registration algorithms and variants. Given the distribution of closest point pair distances we observed we propose extending this test-bed with novel approaches to produce better registration results through proper outlier classification.

Dedicated to my wife

ACKNOWLEDGEMENTS

I thank everyone who has made this thesis possible. In particular, I would like to thank my advisor, Prof. Flynn for introducing me to the field of computer vision and taking an interest in me while still an undergraduate. I also thank Prof. Boyer for his service on my honors distinction committee.

I thank those in the Signal Analysis and Machine Perception Lab at The Ohio State University who have answered countless questions of mine and helped prepare me for my future graduate work. Specifically, Rick Campbell has assisted with programming, range image acquisition, and generally making the lab a happy place. Ravi Srikantiah helped me get started with VTK and his wry sense of humor helped make some long nights go more pleasantly. Anand Kalyanaraman, the future Shisen-Sho champion of the world, has assisted me in expanding my mathematical background. While not laughing at his humorous morning emails, Cem Unsalan several times helped me get out mental ruts when I got stuck.

My parents, past professors, and past teachers have all inspired in me a thirst for knowledge and deeper understanding of the world about me.

Most of all, I thank my wife, Dianna, for her patience with my sometimes long hours at the lab and for encouraging me to always perform at my peak.

This research was supported in part by the National Science Foundation under grant EIA-9818212.

VITA

March 20, 1975 .... Born - Rochester, MN, USA

1999-present .... Undergraduate Research Associate, The Ohio State University

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

Studies in 3D Model Building: Prof. Patrick J. Flynn

TABLE OF CONTENTS

Abstract ii

Dedication iii

Acknowledgements iv

Vita v

Table of Contents ii

List of Figures viii

Chapter 1 : Introduction 1

1.1 Range Images 1
1.2 Registration 1
1.3 Mesh Integration 2
1.4 Organization of the Thesis 2

Chapter 2 : Range Image Registration: Prior Work 4

2.1 Iterative Closest Point Registration 4
2.2 Spring Mass 8

Chapter 3 : Rationale and Goals for the Test-Bed 10

3.1 Extensibility 10
3.2 Instrumentality 10
3.3 Reusability 11
3.4 Summary 11

Chapter 4 : Implementation 12

4.1 Integration with Visualization Toolkit (VTK) 13
4.1.1 Overview of VTK 13
4.1.2 `vtkMultiInOutPolyDataFilter` and `vtkFilterAlgorithm` Classes 17
4.2 Basic Registration 18
4.3 Hooks for Variants 20
4.3.1 `vtkICPLoopCriterion Class` 20
4.3.2 `vtkPointCorrespondenceAnalyzer` Class 22
4.3.3 `vtkICPAccelerator` Class 25
4.4 Instrumentation 26
4.4.1 Progress Callback 26
4.4.2 `vtkICPHistoryItem` Class 26
4.4.3 `CCSVFile` Class 27
4.5 Visualization 28

4.5.1 The Test-Bed Graphical User Interface 28
4.5.2 vtkCenterAtOriginAlgorithm Class 30
4.5.3 vtkPointCorrespondenceAlgorithm Class 30
4.5.4 vtkClosestPointsConnectorAlgorithm Class 30

Chapter 5 : Experimental Analysis and Results 34

5.1 Experimental Setup 34
5.1.1 Data Sets and View Pairs 36
5.1.2 Decimation Factor 37
5.1.3 Outlier Classifiers and Parameters....37
5.1.4 Loop Exit Criterion Threshold 38
5.1.5 Experiment Runs 38
5.2 Analysis Methodology 38
5.3 Experimental Results 39
5.3.1 Effects of Outlier Classifier Type and Parameter Settings....42
5.3.2 Effects of Decimation 45
5.3.3 Effects of the Loop Criterion Threshold Value 47
5.4 Summary and Conclusions.... 47

Chapter 6 : Future Work....49

Bibliography 51

Appendix A : Hardware and Software Tools Used 52

Appendix B : Test Naming Conventions....53

LIST OF FIGURES

Figure 1.1: Steps in building a 3D model using a range sensor.................... 3

Figure 1.2: Simple range image. (a) is a 2D array of the z-values for each grid point. The x and y coordinates are implicitly defined by the row and column position of each grid point. (b) is a rendering of the range image.................... 3

Figure 2.1: Tree classifying the most common types of registration algorithms.................... 4

Figure 2.2: Registration error function to be minimized during ICP registration.................... 5

Figure 2.3: Schütz’s method for classifying outlier corresponding point pairs.................... 7

Figure 2.4: Factors for determining convergence and detecting poorly coupled range images.................... 7

Figure 2.5: Zhang's Outlier Classification Scheme 8

Figure 4.1: Test-Bed Application Components....................12

Figure 4.2: Section of a VTK Pipeline for Displaying a Sphere. Arrows represent direction of data flow via `Execute()`....................14

Figure 4.3: Implicit Network Execution. (a) The diagram shows five pipeline process objects. Arrows represent direction of data flow via `Execute()`. The top text section describes the interaction diagram in (b). The lower text section desribes the interaction diagram in (c). Adapted from 94 of [9]. 16

Figure 4.4: vtkMultiInOutPolyDataFilter and vtkFilterAlgorithm Class Diagram....................17

Figure 4.5: Internal pipeline branch structure for the `vtkMultiInOutPolyDataFilter` with a `vtkRegistrationAlgorithm`....................19

Figure 4.6: Class Diagram for `vtkRegistrationAlgorithm` and `vtkICPAlgorithm`....................19

Figure 4.7: Pseudo-Code for `vtkICPAlgorithm::Execute()` 21

Figure 4.8: Class Diagram for Registration Tweaks....................21

Figure 4.9: Loop Criterion Class Diagram....................21

Figure 4.10: Truth Table for vtkLinearThreshold Criterion....................22

Figure 4.11: Required Calculations for a Set of Point Correspondence Analyzers 24

Figure 4.12: Point Correspondence Analyzers Class Diagram....24

Figure 4.13: Equations for vtkCenterOfMassCalculator....25

Figure 4.14: vtkHistoryItem Class Diagram 27

Figure 4.15: Close-up view of the nose for theFacesPat1_000U02-108U02_V70984_0003 test. The lines shown connect closest point pairs....29

Figure 4.16: User manually selecting corresponding point pairs for initial registration of the Buddha_000U01-020U01 tests. The yellow dots represent the points picked and have been enlarged for this document. This figure shows the second point in the 5th pair being selected....32

Figure 4.17: The main GUI window for setting up and analyzing registration tests 32

Figure 4.18: The history dialog give in-depth information about each ICP iteration and allows for visualization of the registration at each iteration. 33

Figure 4.19: Class Diagram for the Visualization `vtkFilterAlgorithms`....33

Figure 5.1: Texture-Mapped Rendering of the Range Images Used in Our Experiments. Labels under each image indicate the data set name followed by the range image name. For the “Angel” and “Buddha” images, this number is approximately equal to the number of rotational degrees. For the “FacesPat1” and “FacesRick1” images, this number is roughly twice that angle. 35

Figure 5.2: Photo of the Minolta Vivid 700 Range Sensor [1]....37

Figure 5.3: Snapshot of the FacesPat1_000U01-144U01_N_0003 test. Demonstrates a catastrophic failure of the registration when large non-overlapping regions exist and no corresponding point pairs are classified as outliers....41

Figure 5.4: Snapshot of the Angel_000U02-060U02_M08873_0003 test. Demonstrates a mis-registration at the edge of the wing in the circled region....41

Figure 5.5: Snapshot of the FacesRick1_000U01-036U01_V01109125_0003 test. Demonstrates good “splotching.” 41

Figure 5.6: Snapshot of FacesRick1_000U01-036U01_M0221825_0003 test. Demonstrates poor “splotching.” 41

Figure 5.7: A typical histogram of the distances between corresponding point pairs for a final registration. Note that the histogram resembles a Gaussian, as hypothesized by [14]. The histogram columns are shaded according to whether the points in that histogram column have been classified as inliers or outliers. This histogram is from the Angel_000U02-040U02_V35492_0003 test....44

Figure 5.8: Hypothesized source of non-Gaussian histograms of distances between corresponding point pairs 46

Figure 5.9: A histogram of the distances between corresponding point pairs for a final registration typical of tests where there are large non-overlapping regions. The histogram columns are shaded according to whether the points in that histogram column have been classified as inliers or outliers. This histogram is from the FacesPat1_000U01-144U01_V01109125_003 test....46

Figure B.1: Naming conventions used for our experiments. See section 5.1 for additional details on the tests performed. 53

CHAPTER 1:

INTRODUCTION

In recent years, there has been a growing interest in techniques for building 3D computer models of real-world objects and scenes without requiring humans to manually produce these models using laborious and error-prone CAD-based approaches. Using "range sensors" instead, users are able to capture 3D images of objects from different viewpoints that may be combined to form the final model of the object or scene. These models then may be used for a variety of purposes such as building 3D maps for robot navigation, providing training data for computer vision experiments, and digitizing historical buildings for restoration planning [12], [14]. Figure 1.1 shows the steps involved in building a model. The following paragraphs discuss the steps in more detail.

1.1 Range Images

The first step in building such models is to obtain a set of range images. A range sensor scans an object or scene, producing a 2D array of distance values called a *range image*. Figure 1.2a represents the distance array. The numbers in each cell of the table represent a distance from the sensor while distances that cannot be quantified appear as crosses. These numbers are used as the values for the z coordinate of points on a polygonal mesh. The x and y coordinates are implicit in the positioning of a grid point in the range image. Figure 1.2b shows a rendered view of the range image in Figure 1.2a.

1.2 Registration

In order to obtain a complete model, these individual range images must be combined into a single data set. Before this final *mesh integration* can be performed, the range images must first be aligned, or *registered*. Just as having humans generate the CAD models is impractical, so is requiring them to perform

the very precise alignments that are needed. Additionally, while some sensors provide positional and orientation data for each range image, these values are generally far too approximate for the purposes of model building. Instead, either human-assisted or this *a priori* sensor-produced registration is used as a first-cut registration. This coarse registration is then refined through an automatic registration scheme.

1.3 Mesh Integration

Once the range images have been registered, they are combined to produce a single surface description. This mesh integration step prunes the redundant data from the input range images and stitches together their surfaces. Mesh integration is another large area of study that will not be discussed in detail in this report.

1.4 Organization of the Thesis

The remainder of this document is organized as follows. Chapter 2 describes the most common methods used for automatic range image registration and the principal approaches that researchers have taken to solve the problem. In Chapter 3, we describe the rationale behind our work in building a software test-bed to comparatively study various range image registration approaches. An overview of the test-bed implementation follows in Chapter 4, and the results and conclusions of comparing several registration methods using this software are given in Chapter 5. Chapter 6 describes additional future work that may be performed to enhance the studies presented here.

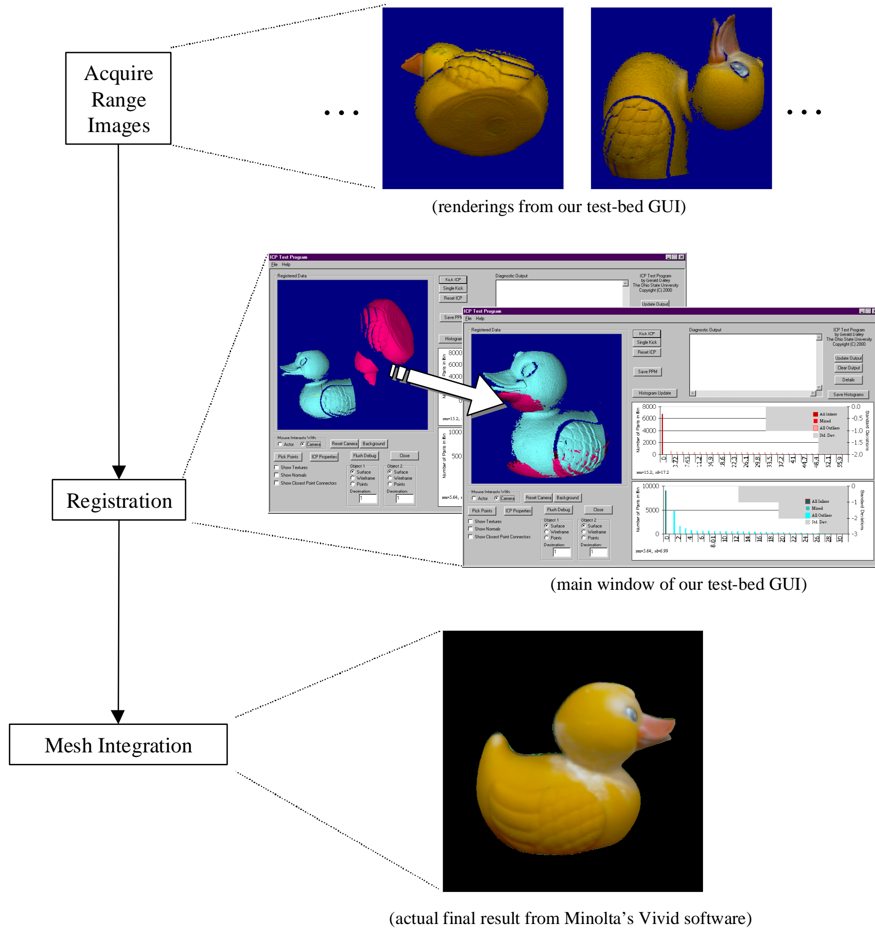
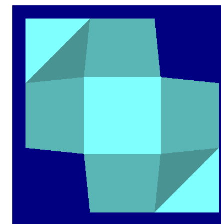


Figure 1.1: Steps in building a 3D model using a range sensor

-1	-1	-1	
-1	0	0	-1
-1	0	0	-1
	-1	-1	-1

(a)



(b)

Figure 1.2: Simple range image. (a) is a 2D array of the z-values for each grid point. The x and y coordinates are implicitly defined by the row and column position of each grid point. (b) is a rendering of the range image.

CHAPTER 2:

RANGE IMAGE REGISTRATION: PRIOR WORK

There are three main classes of algorithms commonly used to register range image pairs, shown as leaves in the tree in Figure 2.1. Early work in the field generally concentrated on extracting and aligning major features of the range images, and some researchers use feature-based methods for registration [12]. Besl and Jain introduced the *Iterative Closest Point* (ICP) method in [4]. This method iteratively minimizes the distance between the range images by finding corresponding point pairs. A significant number of researchers use Besl's algorithm in one form or another [12]. A second large subgroup of researchers use a "spring-mass" based system which iteratively performs a series of damped oscillations on corresponding point pairs based on spring-mass physics systems [6], [14]. In this chapter, we will describe the ICP algorithm and its variants in more detail.

2.1 Iterative Closest Point Registration

If a range image, P, is being registered to a data set X by using a rotation matrix $\mathbf{R}$ and a translation vector $\overline{T}$, then the ICP registration process attempts to minimize the function given in Figure 2.2. [4] uses

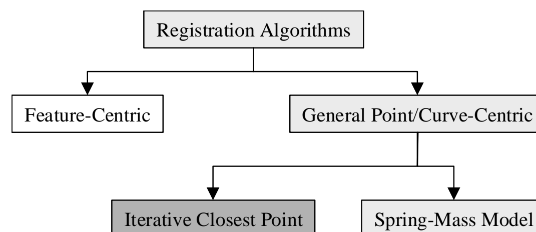


Figure 2.1: Tree classifying the most common types of registration algorithms

$P = \{\overline{p}_i\} \equiv$ Points in the range image being registered

$X = \{\overline{x}_i\} \equiv$ Points in the reference range image to which P is registering. For a given point, $\overline{p}_i$, $\overline{x}_i$ is the closest point in data set X to the point $\overline{p}_i$. $\overline{p}_i$ and $\overline{x}_i$ are called a corresponding point pair (not necessarily 1:1).

$N_P \equiv$ Number of points in data set P

$\mathbf{R} \equiv$ Registration rotation matrix

$\overline{T} \equiv$ Registration translation vector

$$f(\mathbf{R}, \overline{T}) = \frac{1}{N_P} \sum_{i=1}^{N_P} \left\| \overline{x}_i - \mathbf{R} \cdot \overline{p}_i - \overline{T} \right\|^2$$

Figure 2.2: Registration error function to be minimized during ICP registration

a quaternion-based approach to find the values for $\mathbf{R}$ and $\overline{T}$ that minimize this function. These calculations are performed iteratively using the following basic algorithm:

1. Given each point $\overline{p}_i$, find the closest point, $\overline{x}_i$, in X to $\overline{p}_i$.
2. Compute the registration ($\mathbf{R}$, $\overline{T}$) such that $f(\mathbf{R}, \overline{T})$ is minimized.
3. Apply the registration to P.
4. Go to step 1 if the difference in the registration errors, $f_n(\mathbf{R}, \overline{T}) - f_{n-1}(\mathbf{R}, \overline{T})$, has not dropped below some threshold.

This process is guaranteed to converge to some local minimum for any starting registration when P is a subset of X.

This base algorithm has been modified in many ways by various researchers in attempts to improve the speed and/or quality of the registrations it produces. For example, in [4] itself Besl and Jain recommend accelerating the iterative process by using a parabolic interpolant to the last three iterations. Others have made recommendations on weighting the point pairs, on classification of outlier corresponding pairs, on more robust methods of detecting convergence, and on methods of detecting descents into non-global registration minima.

Turk and Levoy [12] enhance the registration by adding a confidence weighting factor to each point pair. Points whose normal is directed away from the range sensor are given a lower confidence value, as are points at the edge of the mesh. These confidence values are used to weigh the summed terms of the minimization function and enhance Besl's algorithm to be more robust to sensor errors.

The original algorithm requires that a range image be registered to a surface that is a superset of the range image. Unfortunately, when building new models from range images, no model already exists to which a range image may registered. Instead, we desire to register multiple range images so that just their *overlapping* regions are aligned. Schütz, Jost, and Hügli propose a simple heuristic method of determining which corresponding point pairs belong to overlapping regions and which are actually not corresponding points [11]. They theorize that point pairs whose distance is much greater than the separation of the centers of mass of two partially registered range images must be outliers. As shown in Figure 2.3, they calculate a binary weighting factor for each point pair. Those pairs whose two points are separated by more than a specified value d are considered outliers and given a weight of zero. Those point pairs whose distance is less than d are considered inliers and have a weight of one. In this way, those points that are highly separated and likely to belong to non-overlapping regions are excluded from the registration calculations.

Schütz also expands the ICP algorithm by enhancing the convergence determination and validating that too many point pairs have not been classified as outliers using the equations in Figure 2.4. To determine convergence, they require the change in the mean distance μ and the standard deviation, σ to be below user-specified threshold values. Additionally, if the "surface coupling" given by ε drops below 30 to 50%, the validity of the ICP step is in question. Thus a better estimation of convergence is obtained and a program can detect when it has begun excluding too many point pairs.

Zhang has implemented a more extensive and sophisticated set of modifications to ICP for the purposes of robot navigation [14]. In addition to aligning both points and curves, he has developed a more theoretically-based method of classifying outlier corresponding point pairs. Central to his method is the assumption that for sufficiently registered range image pairs, the distances between points in the corresponding point pairs will be Gaussian distributed. This Gaussian is due to mis-registration and noise

$$w_i = \begin{cases} 1 & \text{if } d_i < (c \cdot s \cdot r)^2 \\ 0 & \text{otherwise} \end{cases}$$

$$d_i = \left\| \overline{p}_i - \overline{x}_i \right\|^2$$

$s \equiv$ Range scanner sample distance

$r \equiv$ Subsampling factor

$c \equiv$ Empirically determined threshold based on the separation of the centers of mass of the data sets

Figure 2.3: Schütz's method for classifying outlier corresponding point pairs.

$$\mu = \frac{1}{N_P} \sum_{N_P} d_i , \quad \sigma = \sqrt{\frac{1}{N_P - 1} \sum_{N_P} (d_i - \mu)^2} , \quad \varepsilon = 100 \frac{\sum w_i}{N_P}$$

Figure 2.4: Factors for determining convergence and detecting poorly coupled range images.

in the input data from quantization and sensor error. A range image is considered to be sufficiently registered when the mean point pair distance is on the order of the range image sampling distance.

A different number of standard deviations from the mean are considered inliers depending upon how close the registration is to this expected error (see Figure 2.5). For example, suppose that the expected error is 2mm, but the actual mean distance error is 5mm and the standard deviation is 4mm. According to the algorithm in the Figure 2.5, this means that we should include as inliers all pairs whose distance is less than the mean distance plus two standard deviations (2mm < 5mm < 3*2mm). All pairs whose distance between its two points is greater than 13mm are then classified as outliers and do not enter in the registration calculations.

If the registration is outside this range, then a fallback method is used to classify the outliers. A histogram of the distances is generated which is examined for the first valley beyond the highest peak. This valley is found by searching for the first histogram point 60% below the peak that is less than the previous

$D \equiv$ Expected error (proportional to the range image sampling distance)
$D_{max} \equiv$ Maximum distance between points in a corresponding point pair for the pair to be considered an inlier
$\xi \equiv$ First valley after largest peak in the histogram of corresponding point pair distances

if $\mu < D$
$D_{max} = \mu + 3\sigma$ Registration is quite good
elseif $\mu < 3D$
$D_{max} = \mu + 2\sigma$ Registration is still good
elseif $\mu < 6D$
$D_{max} = \mu + \sigma$ Registration is not too bad
else
$D_{max} = \xi$ Registration is really bad

Figure 2.5: Zhang's Outlier Classification Scheme

histogram point. All point pairs whose distance falls beyond this valley are classified as outliers and removed from the current iteration's registration calculations. outlines this classification scheme.

The ICP algorithm originally presented by Besl [4] provides a means for iteratively performing a least squares registration. This registration is guaranteed to converge when there are no errors in the data and when a range image is being registered to an existing model that wholly contains the surface described by the range image. Researchers have expanded on this base algorithm in attempts to provide better performance and to make it more robust to errors and non-overlapping regions.

2.2 Spring Mass

Instead of the ICP algorithm, Eggert et al. [6] use a spring-mass model to perform registration. Although the implementation of the project documented in this thesis focuses on ICP, certain concepts from this non-ICP work can be applied to ICP. In the spring-mass approach, registration begins by finding the corresponding point pairs, as is done in ICP. Between each pair, a damped spring is logically set up whose undamped force is proportional to the distance between the points. These individual forces are combined to calculate the overall forces and torques on the rigid range image meshes. Motion is then calculated for a small time step. This motion is applied and the force calculations are revisited without performing the computationally expensive point correspondence search again. The oscillating motions are

iteratively performed until they cease to move the data by a significant distance. Once these oscillations settle down, the corresponding point pair search is performed again. With the new correspondence data, a new set of spring-mass calculations are performed until they converge. The process of finding corresponding point pairs and performing damped oscillations is repeated until the overall motion is sufficiently small. At this point, the range images are registered. One feature of Eggert's approach that can be applied to ICP is that during the nearest neighbor search, matching points are required to be both close to each other and to have similar smoothed normals.

CHAPTER 3:

RATIONALE AND GOALS FOR THE TEST-BED

Many of the papers discussing new variants to the ICP algorithm and other registration algorithms have only been tested on a small set of objects. Additionally, few algorithms have been tested on the same set of data. Broader comparative studies to date have been limited in scope, such as Lorusso's studies on four different methods of calculating the ICP registration values approach [8]. This lack of comparative data makes it difficult to judge the true strengths and weaknesses of each one. A test-bed platform facilitates the making of effective comparisons between algorithm variants if properly designed. The following goals serve as the basis for such a design:

1. Extensibility
2. Instrumentality
3. Reusability

3.1 Extensibility

The extensibility of the platform is measured by the ease with which it may be extended to include other registration algorithms and algorithm variants. This ease translates into decreased development time for new variants. Since there are so many algorithms, variants, and sub-variants for registration, this feature is important.

3.2 Instrumentality

After implementing the test-bed, it must be used to gather data for evaluating the relative fitness of various algorithm variants. At a minimum, the test-bed must be able to readily yield the following information given a set of variants and input data:

1. Final registration error
2. Registration transformations
3. Number of ICP iterations
4. Total execution time

The final transforms and registration error provide the means for evaluating the validity and quality of a particular registration test. The registration error serves as an objective value, and the transformations allow for visualization of the results for subjective human verification of the registration. The total number of ICP iterations required and the total execution time provide the means to evaluate the relative performance of registration tests.

3.3 Reusability

Given a highly instrumentable and extensible set of software for supporting the registration of range images, the test-bed software should be engineered to be reusable in larger applications. For example, an application that merges range images should be able to use the registration software built for the test-bed as a pre-processing step.

3.4 Summary

To better evaluate the relative strengths and weaknesses of the various registration methods available, we have implemented a test-bed environment. In addition to being able to collect raw results data, this test-bed can serve longer-term purposes by its extensibility and reusability features. In the next chapter, we will examine the design and implementation of the test-bed software.

CHAPTER 4:

IMPLEMENTATION

In order to fulfill the design goals of extensibility, instrumentality, and reusability, we used object-oriented design methodologies and tools. In particular, the *Strategy Pattern* described in [7] was used extensively. The bulk of the test-bed code was written in C++ using the Visualization Toolkit (VTK) version 2.4 [3], [9], [10]. Additional tools such as those used in visual modeling of the software design and those used in the development of the user interfaces are mentioned in Appendix A.

Our test-bed applications perform the functions shown in Figure 4.1. The GUI application reads in range images that were produced by our range sensor, passes them through the registration process, then performs the required visualization functions. After performing the registration on the input range images, our batch-processing application outputs timing and registration information.

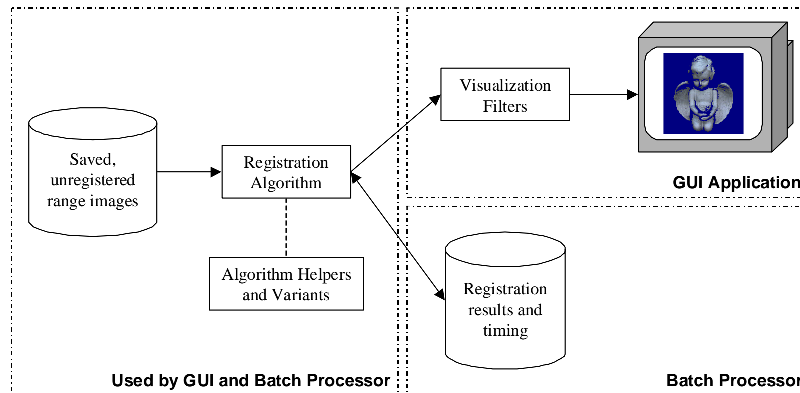


Figure 4.1: Test-Bed Application Components

Section 4.1 introduces VTK and describes in general how the toolkit was used. Section 4.2 describes the classes written to support basic range image registration. Section 4.3 shows hooks for variants of the ICP algorithm, and Section 4.4 details the instrumentation hooks. Section 4.5 describes the visualization support and gives screenshots of the most important windows in the graphical user interface (GUI) application.

4.1 Integration with Visualization Toolkit (VTK)

A large body of multi-dimensional data-set manipulation and visualization software has already been created. Our test-bed software is built upon the Visualization Toolkit (VTK). To the base toolkit, we added classes to assist in the automatic registration process, and we built classes to aid in the visualization of the registration results. In this section we will give an overview of VTK, then we will explain in greater detail the classes which we have written to support registration.

4.1.1 Overview of VTK

VTK is an open source software system supported by Kitware that supports the processing and visualization of 2D, 3D, and high-dimensional data [3], [9], [10]. The toolkit ships with over 500 C++ classes that can be used as-is or extended in either C++ or through several popular scripting languages.

At the core of a VTK application is the implicitly executed data pipeline. In general, there are two types of objects in a pipeline: data objects and process objects. Data objects store the data to be processed and visualized such as sets of polygons or implicit 3D data. Process objects may supply, use, or modify the content of these data objects.

A graphical view of a portion of a simple pipeline is shown in Figure 4.2. At the head of a pipeline is a source, such as the `vtkSphereSource` shown in the figure. A source outputs some sort of data object such as the `vtkPolyData` that holds 3D polygonal data sets. Between the source and the rendering portion of the pipeline, any number of filters may be used. The figure shown contains a `vtkTransformFilter` that performs a 4x4 rigid matrix transformation on the input data. The rendering portion of the pipeline contains various data objects and process objects to set up lighting

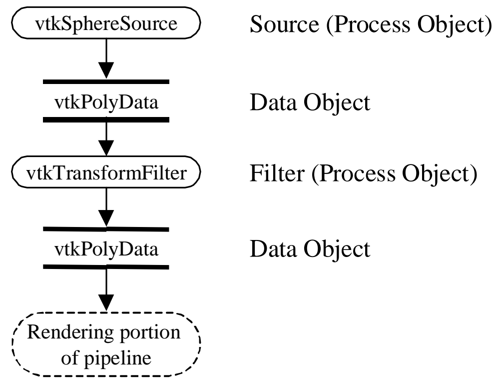


Figure 4.2: Section of a VTK Pipeline for Displaying a Sphere. Arrows represent direction of data flow via `Execute()`.

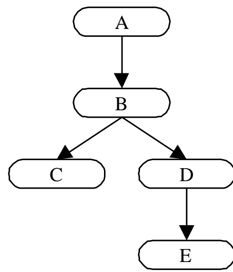
conditions, perform the 3D-to-2D mapping, etc. More complex pipelines, such as the ones used in the range image registration test-bed follow this same basic structure, but may contain more filters, and may have several pipelines logically in parallel.

VTK pipelines operate "implicitly," meaning that downstream process objects request the newest data whenever it is required, and the upstream objects implicitly perform only whatever execution is necessary to guarantee current results. This process requires two steps, an update step and an execution step. The update step consists of downstream process objects requesting current information from upstream objects. The execute step happens when upstream objects process their data and produce new data in their output for the downstream objects to use.

For example, consider Figure 4.3 adapted from the discussion on pages 92-97 of [9]. If `A` is modified and then the output from `E` is requested, then the update step is performed on the chain `E-D-B-A`. When `A` is updated, it recognizes that it must regenerate its results and calls `Execute()` on itself, updating its output. This updated output means that `B` must now regenerate its output through a call to `Execute()`. This execution step continues for `D` and `E`.

In the second example, `C` is modified and has its output requested. The chain `C-B-A` has updates requested, but since nothing has changed for `A` or `B`, their `Execute()` methods are not called. Since `C` has changed, its `Execute()` method is called, producing current output.

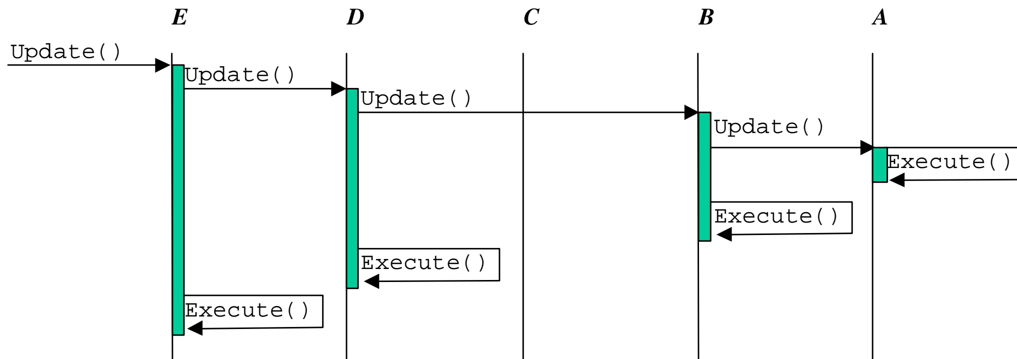
An explicit execution model requires an independent object, called an executive, to keep track of when which portions of the pipeline need to be updated and explicitly cause those updates to occur. Although explicit execution can be more flexible, it often suffers from complex executive implementations. The registration test-bed software uses the implicit execution scheme favored by VTK, but has limited hooks for explicit execution where necessary for instrumentation and algorithm evaluation purposes.



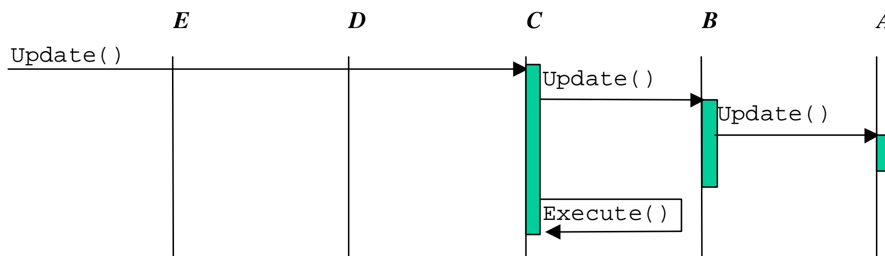
Execution of *E*: Full Execution
1. `A` parameter modified
2. `E` output requested
3. Chain `E-D-B-A` back propagates `Update()` method
4. Chain `A-B-D-E` executes via `Execute()` method

Execution of *C*: Partial Execution
1. `C` parameter modified
2. `C` output requested
3. Chain `C-B-A` back propagates `Update()` method
4. No modifications made to `A` or `B` so `Execute()` method not called for them
5. `C` executes via `Execute()` method

(a)



(b)



(c)

Figure 4.3: Implicit Network Execution. (a) The diagram shows five pipeline process objects. Arrows represent direction of data flow via `Execute()`. The top text section describes the interaction diagram in (b). The lower text section desribes the interaction diagram in (c). Adapted from 94 of [9].

4.1.2 `vtkMultiInOutPolyDataFilter` and `vtkFilterAlgorithm` Classes

Nearly all of the built-in VTK filter classes have a single input data object and a single output object. A few support an arbitrary number of inputs, but these still only have a single output object. The task of registration requires sets of input-output data object pairs. The input objects are the range images before registration, and the output objects are copies of the inputs transformed so they are registered with each other. To support this functionality, we have created the `vtkMultiInOutPolyDataFilter` class (see Figure 4.4).

The `vtkMultiInOutPolyDataFilter` handles the simultaneous updates and execution of all of its "pipeline branches." A pipeline branch is an input-output data object pair, plus any internal filters used to produce the output. When its `Execute()` method is called to make the outputs current, it delegates the execution to its `vtkFilterAlgorithm`, employing the *Strategy Pattern* [7].

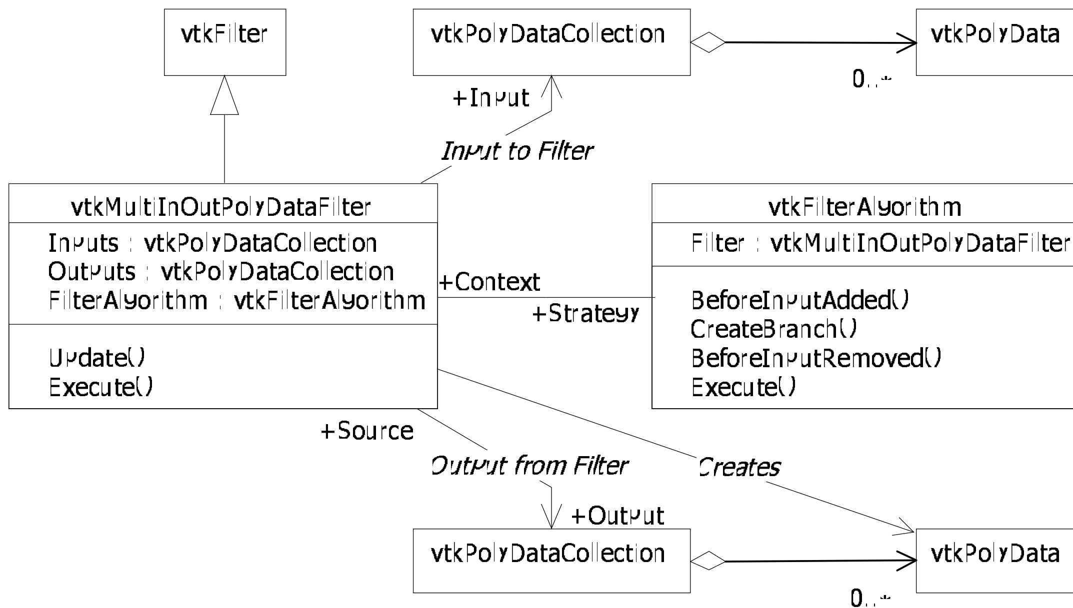


Figure 4.4: vtkMultiInOutPolyDataFilter and vtkFilterAlgorithm Class Diagram

The `vtkFilterAlgorithm` base class (see Figure 4.4) defines the following responsibilities with respect to its associated `vtkMultiInOutPolyDataFilter`:

1. Creation of the output data objects used internally by the `vtkMultiInOutPolyDataFilter`
2. Regulating internal pipeline branches when new inputs are added or removed
3. Performing the necessary work to move data into the output objects

These responsibilities have been separated out from the `vtkMultiInOutPolyDataFilter` class to allow greater dynamic flexibility when experimenting with the pipeline setup. For example, suppose that the test program begins by using one particular registration algorithm and then the user wishes to test another algorithm on the same set of input data. With the existing design, the new `vtkFilterAlgorithm` subclass can be substituted in for the old registration algorithm without disturbing the structure of the pipeline. If the `vtkMultiInOutPolyDataFilter` and `vtkFilterAlgorithm` classes were merged, then the new filter would need to have all of its inputs added back in manually and all of its outputs reconnected to the rest of the pipeline.

4.2 Basic Registration

To actually perform the range image registration, we have created a subclass of `vtkFilter-Algorithm` called `vtkRegistrationAlgorithm` (see Figure 4.5). This base class creates internal pipeline branches that perform rigid transformations for each of the inputs to the associated `vtkMultiInOutPolyDataFilter` (see Figure 4.6). Although this base class performs little actual work in and of itself, it acts as a reusable base class from which a wide variety of `vtkFilterAlgorithms` may be created that simultaneously perform rigid transformations on a series of inputs.

Currently, the most interesting subclass of `vtkRegistrationAlgorithm` is `vtkICP-Algorithm`. When executed, this algorithm uses Besl's Iterative Closest Point (ICP) algorithm described in section 2.1 to set the internal pipeline transformations in its associated `vtkMultiInOutPolyData-Filter`.

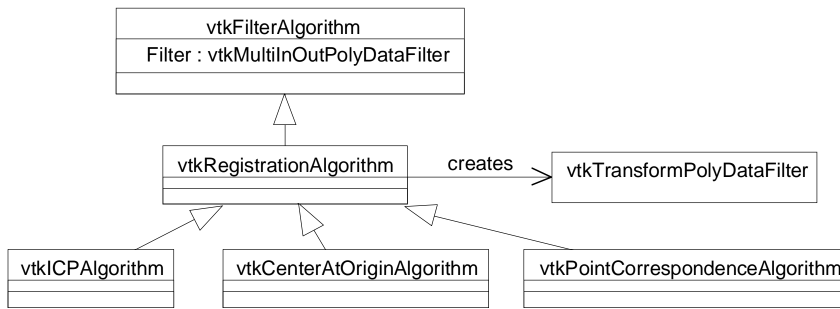


Figure 4.5: Internal pipeline branch structure for the `vtkMultiInOutPolyDataFilter` with a `vtkRegistrationAlgorithm`

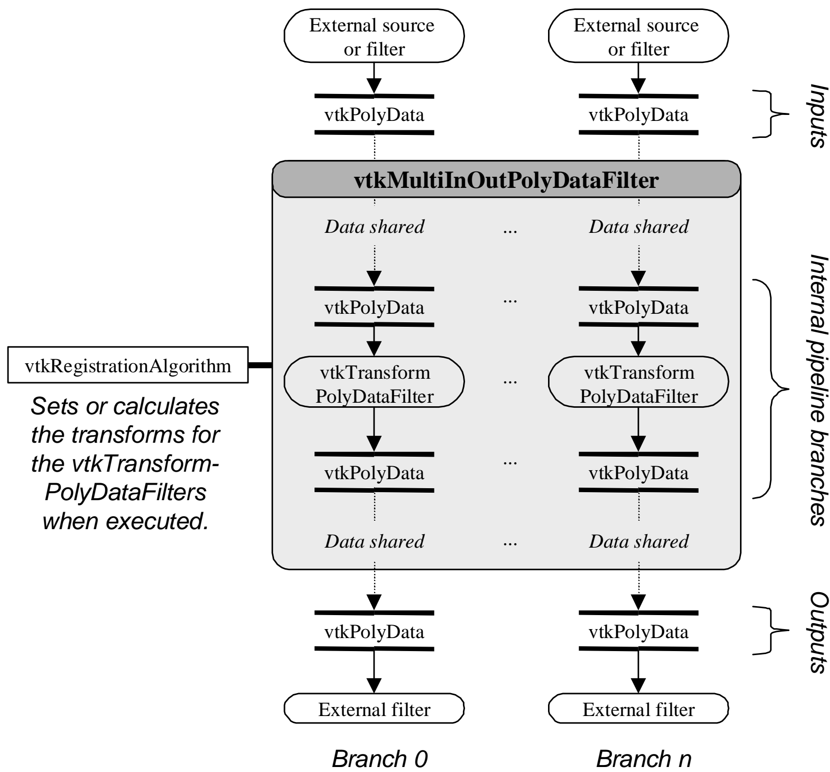


Figure 4.6: Class Diagram for `vtkRegistrationAlgorithm` and `vtkICPAlgorithm`

4.3 Hooks for Variants

With the purpose of developing the test-bed being to evaluate different range image registration methods, several hooks for algorithm variants have been added to the `vtkICPAlgorithm` class. These hooks are grouped into three major categories: loop exit criteria, point correspondence analyzers, and registration acceleration. Figure 4.7 gives pseudo-code for the execution loop of `vtkICPAlgorithm`, indicating how each of the three variant hooks are integrated into the loop. Figure 4.8 gives a class diagram of the interfaces that the variants use.

4.3.1 `vtkICPLoopCriterion Class`

The loop exit criteria are implemented as subclasses of `vtkICPLoopCriterion`. There is always exactly one criterion used per `vtkICPAlgorithm` instance, and that criterion is responsible for indicating the validity and convergence of the current ICP iteration. Implementations of `vtkICPLoopCriterion` may declare an ICP iteration sequence either because the point correspondence analyzers have introduced fluctuations in the registration error descent or they may use some more sophisticated methods to perhaps detect descent into a local, non-global minimum.

At the present time, only one subclass, `vtkLinearThresholdCriterion`, has been implemented (see Figure 4.9). This class examines the present iteration and previous ICP iteration to determine validity and convergence, as shown in Figure 4.10. If the change in the registration errors is less than the `Threshold` value, then the current iteration is deemed to have converged. Thresholds closer to zero generally require more ICP iterations and yield registrations with a smaller mean squared distance error value. This class also determines validity by verifying that the previous iteration's registration error is greater than or equal to the error for the current iteration.

```
while the loop criterion indicates that more ICP
      iterations are necessary
begin
  calculate the point correspondences for all branches
  calculate the registration for each branch using Besl's algorithm
  perform registration acceleration to tweak the
      just-calculated registrations
  apply the tweaked registrations simultaneously
end while
```

Figure 4.7: Pseudo-Code for `vtkICPAlgorithm::Execute()`

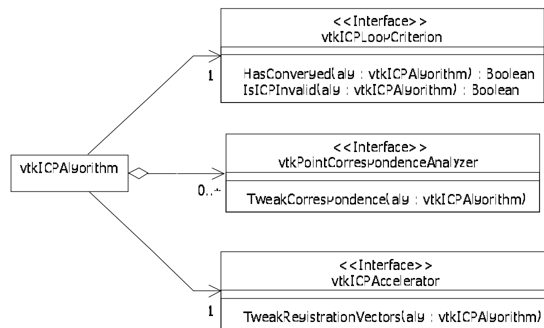


Figure 4.8: Class Diagram for Registration Tweaks

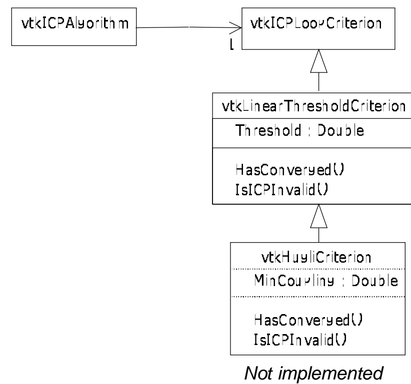


Figure 4.9: Loop Criterion Class Diagram

$$\varepsilon_c \equiv \text{Error for current ICP iteration}$$
$$\varepsilon_p \equiv \text{Error for previous ICP iteration}$$
$$\tau \equiv \text{Convergence threshold}$$
$$\Delta\varepsilon = \varepsilon_p - \varepsilon_c$$
$$(\text{converged, valid}) = \begin{cases} (\text{T,T}) & \text{if } \Delta\varepsilon = 0 \\ (\text{T,F}) & \text{if } 0 < \Delta\varepsilon < \tau \\ (\text{F,T}) & \text{if } \tau \le \Delta\varepsilon \\ (\text{F,F}) & \text{if } \Delta\varepsilon < 0 \end{cases}$$

Figure 4.10: Truth Table for vtkLinearThreshold Criterion

4.3.2 `vtkPointCorrespondenceAnalyzer` Class

The point correspondence analyzers are implemented as subclasses of `vtkPointCorrespondenceAnalyzer`, and there may be any number of analyzers sequentially used by a particular `vtkICPAlgorithm` instance. These analyzers are responsible for determining specific information about corresponding point pairs, as shown in Figure 4.11. At the present time, the analyzers shown in Figure 4.12 have been implemented.

4.3.2.1 `vtkClosestPointFinder` Class

By default, a `vtkClosestPointFinder` object is used as the first analyzer for a `vtkICPAlgorithm`. This object creates or replaces the `ClosestPoint.Branch`, `ClosestPoint.ID`, `ClosestPoint.Distance`, and `ClosestPoint.PairWeight` point fields for all branches. To determine these values, the object loops through all points in all branches. Inside this double loop, it calculates the distance from the given point to all other points in all other branches. The other point that has the smallest distance is selected as the closest point. The pair weight is simply set to `1.0` for every point. More sophisticated implementations could use more efficient nearest-neighbor search techniques such as kd-tree based searches.

4.3.2.2 `vtkMaxDistancePointWeightAnalyzer` Class

Once the `ClosestPoint.Branch`, `ClosestPoint.ID`, `ClosestPoint.Distance`, and - `ClosestPoint.PairWeight` point fields have been calculated, a `vtkMaxDistancePoint-WeightAnalyzer` object modifies the `ClosestPoint.PairWeight` field. Based on the method introduced in [11], it identifies outliers as those point pairs that are separated by "too much" space. As discussed in section 2.1, this method attempts to solve the problem of partially overlapping data sets. The original paper suggests that the distance threshold be linearly proportional to the center of mass separation of the point set and its corresponding closest points, but leaves out details on how to properly choose the proportionality constant. By default, in the test-bed setup and visualization application, the constant is set to the average estimated triangle edge length. Any point pairs whose separation is greater than this constant times the center of mass separation are considered outliers and have their weight set to zero. If the `CenterOfMass` and `ClosestPointsCenterOfMass` data set fields have not been set for a branch, this class internally uses an instance of `vtkCenterOfMassCalculator` to perform the calculation (see below).

Data About Each Point in Each Internal Pipeline Branch	
ClosestPoint.Branch	ID of the branch that has the point closest to a given point
ClosestPoint.ID	ID of the point that is closest to a given point
ClosestPoint.Distance	Distance between a given point and its closest point not in its own branch
ClosestPoint.PairWeight	"Weight" or "mass" of the point pair when viewed from the perspective of the given point
Data About Each Internal Pipeline Branch	
CenterOfMass	Center of mass of the weighted points in a particular internal pipeline branch
ClosestPointsCenterOfMass	Center of mass of the points closest to those in a particular branch
TotalWeight	Total mass of a particular branch

Figure 4.11: Required Calculations for a Set of Point Correspondence Analyzers

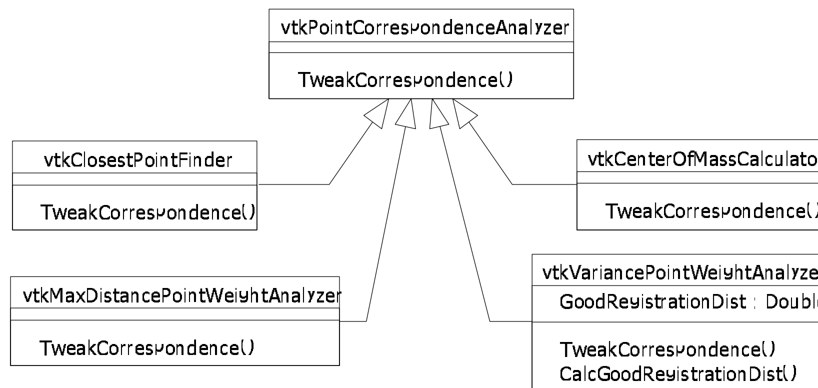


Figure 4.12: Point Correspondence Analyzers Class Diagram

4.3.2.3 `vtkVariancePointWeightAnalyzer` Class

A `vtkVariancePointWeightAnalyzer` object performs the same function as a `vtkMax-DistancePointWeightAnalyzer` object: estimating which points are from overlapping sections of the original object. It uses the more statistically sophisticated method introduced on pages 123-125 of [14].

4.3.2.4 `vtkCenterOfMassCalculator` Class

The `vtkCenterOfMassCalculator` class is responsible for calculating the center of mass of a range image. It does so using the standard method shown in Figure 4.13.

If the `ClosestPoint.PairWeight` point fields do not exist, all points are given a weight of `1.0`. If the `ClosestPoint.Branch` or `ClosestPoint.ID` point fields do not exist, then the center of mass of the closest points is not calculated. The results are placed in the `CenterOfMass` and `Closest-PointsCenterOfMass` data set fields.

4.3.3 `vtkICPAccelerator` Class

Registration acceleration can be provided through concrete implementations of the `vtkICP-Accelerator` class. If not accelerator is provided to a `vtkICPAlgorithm`, then the registration vectors generated by the base ICP algorithm are used directly to calculate the registration transforms. At

$$w_i \in \text{ClosestPoints.PairWeight}$$
$$\overline{p}_i \in \text{Points in range image}$$
$$\overline{p}_i \in \text{Points in ClosestPoints.Branch : ClosestPoints.ID}$$
$$\overline{\text{CenterOfMass}} = \sum_{i=1}^{\text{num points in range image}} w_i \cdot \overline{p}_i$$
$$\overline{\text{ClosestPointsCenterOfMass}} = \sum_{i=1}^{\text{num closest points in range image}} w_i \cdot \overline{q}_i$$

Figure 4.13: Equations for vtkCenterOfMassCalculator

the present time, no concrete accelerators have been implemented.

4.4 Instrumentation

Registration evaluation is facilitated by the instrumentation built into the test-bed software. This instrumentation can provide runtime approximation about how much work remains to complete the registration. More importantly, key information is stored for each ICP iteration to allow more thorough analysis. Coupled with this instrumentation is a serialization mechanism that can be used with the test-bed applications and may also be easily processed by shell scripts and loaded directly by popular spreadsheet programs.

4.4.1 Progress Callback

Primarily to facilitate the inclusion of GUI progress meters for long operations, VTK has a built-in method for filters to report an approximate fraction of the work they have completed while executing their `Update` methods. For example, if a filter's `Progess` is `0.75`, then the filter estimates that 75% of its work has been completed. If the user has supplied one, the filter may periodically call a `ProgressMethod` to report changes in the progress estimate. The `vtkICPAlgorithm` class takes advantage of this mechanism by calling the `ProgressMethod` after each ICP iteration. The current test-bed implementation utilizes this mechanism for progress bars, as originally intended, though its use could easily be extended to provide additional information such as timing the length of each individual ICP iteration through a callback.

4.4.2 `vtkICPHistoryItem` Class

The primary means for analysis of the registration steps is through examining a stack of `vtkICP-HistoryItem` instances associated with a `vtkICPAlgorithm` (see Figure 4.14). For every ICP iteration, a `vtkICPHistoryItem` is created. As new key portions of data become available, they are placed into the history item. Certain information, such as whether the ICP iteration was valid or had

converged is stored as single elements in the history item. Other information such as the registration error and center of mass data is conceptually stored in an array, with one element per internal pipeline branch.

A `vtkICPAlgorithm` object can rollback to the state specified by a `vtkICPHistoryItem` via its `RollbackToHistory` method. Additionally, whenever any input data object to the `vtkMultiInOutPolyDataFilter` is modified, the existing registration history can no longer be considered valid. Upon execution, the `vtkICPAlgorithm` maintains the previous final registration, but flushes the old history and begins a new history stack.

4.4.3 `CCSVFile` Class

In order to allow the batch processing of registration tests and saving of their results, a relatively simple serialization mechanism is used. Under the current architecture, a parent object acts as a builder for its sub-objects using token-based commands in a Comma-Separated Variable (CSV) file. For example, if a `vtkICPAlgorithm` object is being built and two successive cell values, "`GenCentersOfMass`" and "`Y`", are read, then the owning object would make the following C++ call: `algBeingBuilt->GenCentersOfMassOn()`. A `CCSVFile` utility class performs the CSV parsing and encoding.

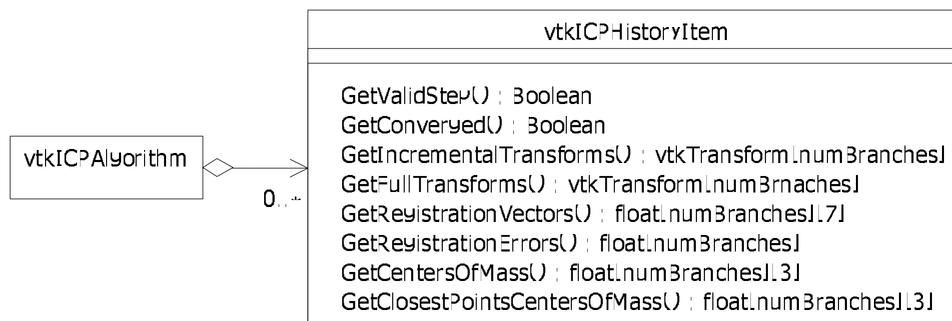


Figure 4.14: vtkHistoryItem Class Diagram

4.5 Visualization

For sensor data, there is no precisely known “ground truth” range image registration. Additionally, each algorithm variant that has a different method of classifying outliers produces different numerical registration error values for exactly the same transform on a particular set of range images. Because of this, naïve objective tests that simply compare registration errors cannot be used alone. Specifically, the registration needs to be viewed subjectively to determine what types of high-level errors it produces and/or avoids. For example, consider a common occurrence when registering human face range images. Although the reported registration error may have been quite low, features such as the nose or ears would be so mismatched that the registration could not be considered valid, even though a local minimum was apparently reached (see Figure 4.15). To account for these difficulties, we developed a graphical user interface (GUI) for the test-bed and several visualization support classes.

4.5.1 The Test-Bed Graphical User Interface

Although VTK has been instrumental in the implementation of the registration algorithm, its true strength is in visualization of data that have been processed. We have created a visualization application that performs the following basic tasks:

1. Initial registration
2. Configuration of registration variants
3. Reading and writing of the CSV configuration and output files
4. Analysis of the registration history

The actual registration may be performed directly using the visualization program, or a configuration file may be output for later batch processing. At the present time, the batch programs must be used if timing information is required.

The initial registration may be specified by one of two methods. First, the user may manually manipulate each range image object with the mouse, using all six degrees of freedom. This method generally results in extremely poor initial registrations. For better initial registrations, the user may open a dialog that allows the user to pick corresponding point pairs (see Figure 4.16). Once all of the pairs are chosen, the dialog is closed, and the ICP algorithm is run with a single iteration to register that small set of corresponding points.

Other dialog boxes may be used to configure the registration variants. For example, the user may specify which point correspondence analyzers to use and their parameter values. Once an initial registration and the registration parameters have been specified, the GUI may be used to output a configuration CSV file of the type described in section 4.4. These files may also be read back into the GUI at any time for visualization or modification purposes.

In addition to assisting with the setting up of test files, the test-bed may be used to visually evaluate the

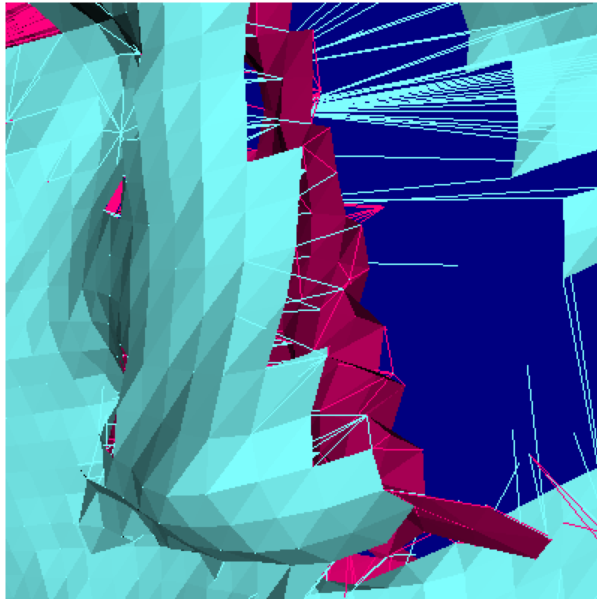


Figure 4.15: Close-up view of the nose for theFacesPat1_000U02-108U02_V70984_0003 test. The lines shown connect closest point pairs.

results of a registration. Through its main GUI dialog shown in Figure 4.17, many visualization options are available such as modifying the rendering style, mouse-based view interaction, and inclusion of data such as lines connecting closest point pairs. A histogram of the distances between corresponding point pairs is also available, as is a window that outputs the state of the current registration as MATLAB source code. A separate dialog may also be launched which allows for detailed numerical and graphical inspection of each step in the registration process (see Figure 4.18). These visualization facilities are enhanced through several additional classes that we developed.

4.5.2 `vtkCenterAtOriginAlgorithm` Class

The built-in mouse interaction in VTK manipulates the scene based on the scene's 3-D center. Unfortunately, the ranges images being registered may be placed in arbitrary places in 3-D space. The `vtkCenterAtOriginAlgorithm` class solves this mismatch by calculating the center of mass of an entire set of inputs and transforming that center to the origin (see Figure 4.19). In the test-bed GUI, this algorithm is applied at the end of the pipeline so as to preserve the registered range images in space close to their original location.

4.5.3 `vtkPointCorrespondenceAlgorithm` Class

The `vtkPointCorrespondenceAlgorithm` class is a relatively simple subclass of the base `vtkFilterAlgorithm` (see Figure 4.19). Just as the `vtkICPAlgorithm` does, it stores a list of `vtkPointCorrespondenceAnalyzers` that attach extra data fields to points in the pipelines upon execution. Unlike the `vtkICPAlgorithm` however, the output data objects do preserve this extra information. This class is generally used in conjunction with another such as the `vtkClosestPoints-ConnectorAlgorithm`.

4.5.4 `vtkClosestPointsConnectorAlgorithm` Class

When the `ClosestPoint.Branch` and `ClosestPoint.ID` point fields have been specified, a `vtkClosestPointsConnectorAlgorithm` object (see Figure 4.19)may be used to construct lines

that connect closest point pairs, such as those shown in Figure 4.15. This ability facilitates understanding which points have been selected as outliers by a point correspondence analyzer and better gauging local registration errors visually.

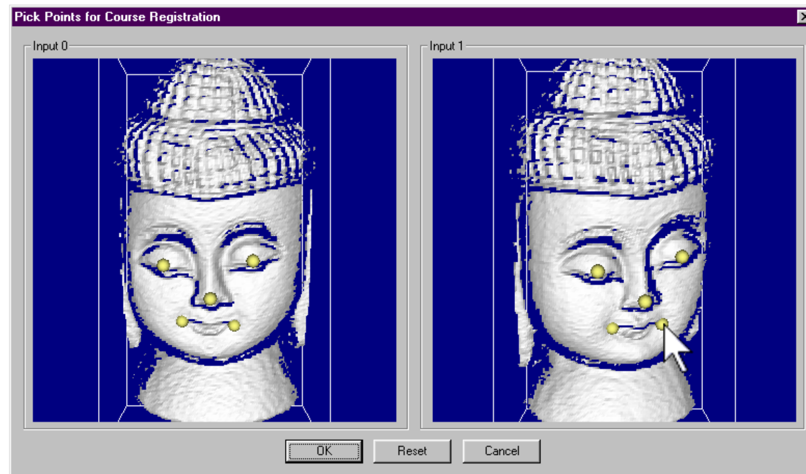


Figure 4.16: User manually selecting corresponding point pairs for initial registration of the Buddha_000U01-020U01 tests. The yellow dots represent the points picked and have been enlarged for this document. This figure shows the second point in the 5th pair being selected.

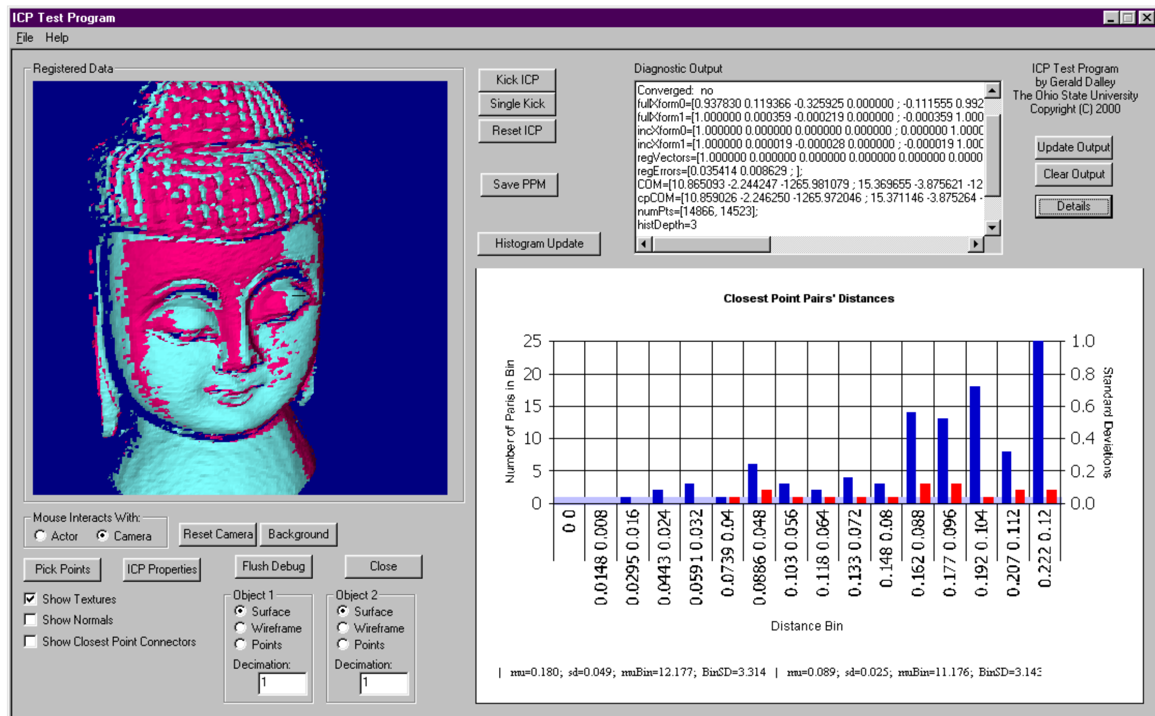


Figure 4.17: The main GUI window for setting up and analyzing registration tests

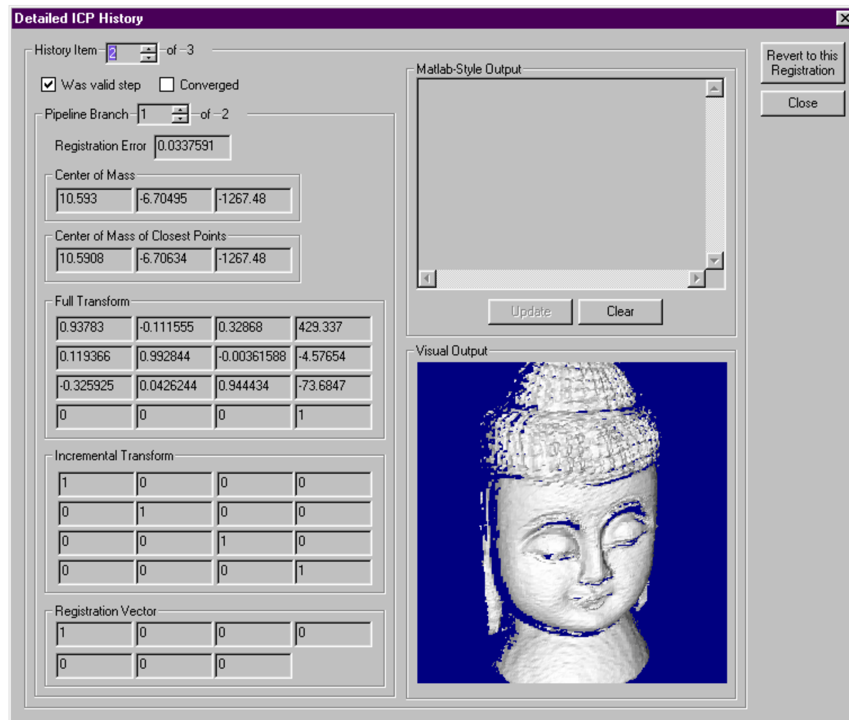


Figure 4.18: The history dialog give in-depth information about each ICP iteration and allows for visualization of the registration at each iteration.

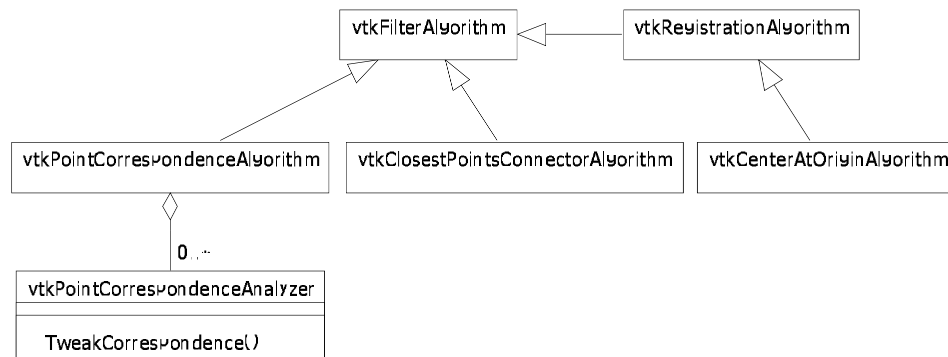


Figure 4.19: Class Diagram for the Visualization `vtkFilterAlgorithms`

CHAPTER 5:

EXPERIMENTAL ANALYSIS AND RESULTS

In the absence of actual "ground truth" registrations against which comparisons may be made, we used a combination of qualitative and quantitative measures to analyze the experimental results. Qualitatively, we examined a sampling of the test data to verify trends and give meaning to the numerical results. We used the numerical data to guide this sampling process and produce additional statistics on the registration results.

In Section 5.1, we describe the ways in which we have setup our experiments, and in Section 5.2 we propose several ways to analyze the quality of a range image registration. Section 5.3 gives the key results that we found through our experiments. At the end of this chapter, in Section 5.4 we will summarize our work and give our conclusions.

5.1 Experimental Setup

In setting up our experiments, we selected several views from four real-world objects, as shown in Figure 5.1. From this set of views, pairs from an object were selected, then we used the GUI to produce an initial registration, as discussed in the Section 4.5. For the tests we performed, we attempted to produce "good", realistic initial registrations that visually appeared to only need a small amount of fine registration.

Once the initial registration was created, we saved the base test configuration file to disk. This configuration file was then duplicated and modified for each registration variant permutation desired. The variables upon which these permutations are based for our tests are:

- **Data set**

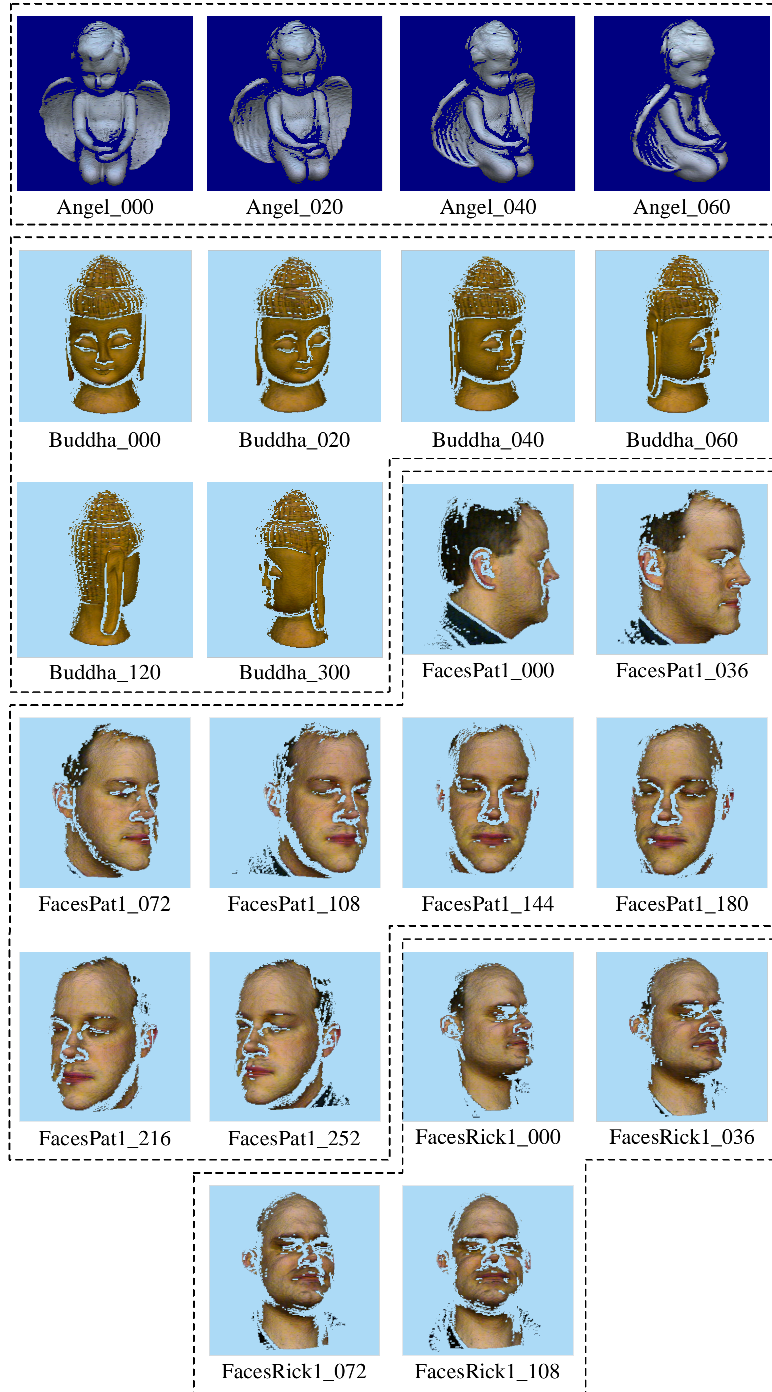


Figure 5.1: Texture-Mapped Rendering of the Range Images Used in Our Experiments. Labels under each image indicate the data set name followed by the range image name. For the "Angel" and "Buddha" images, this number is approximately equal to the number of rotational degrees. For the "FacesPat1" and "FacesRick1" images, this number is roughly twice that angle.

- **View Pair**
- **Decimation Factor:**
- **Outlier Classifier**
- **Outlier Classifier Parameter**
- **Loop Exit Criterion Threshold**

Our tests are named according to which values for these parameters were chosen. For details on the naming conventions we used, see Appendix B. Throughout this thesis, we have followed this convention when labeling figures and referring to specific experiments. The remaining subsections will give details on the permutations chosen. The final subsection of this section summarizes the permutations and briefly describes our registration testing environment.

5.1.1 Data Sets and View Pairs

For this paper, we used range images from the "Angel", "Buddha", "FacesPat1", and "FacesPat2" data sets. All of the range images on which we gathered data are shown in Figure 5.1. Most of the tests included the 000 image as one of the two images in the pair. The narrowest view angle used in these tests was 20 degrees and the widest view angle was approximately 126 degrees. Note that for the "FacesPat1" and "FacesRick1" data sets, the numbers used to name the range images are actually double the approximate angle (e.g. the 126 degree view of "FacesPat1" is labeled as 252).

The "Angel" data set was supplied with our Minolta Vivid 700 range sensor, shown in Figure 5.2. The most important feature of this data set is that the Angel_000 view is, for all intents and purposes, a superset of all of the other views. The large wings on either side of the body block the camera's view of the side and back of the angel.

The next data set, "Buddha," was also supplied with our sensor. The Buddha head is quite round and has some very nice 3D texture in the hair. The ears served as one of the locations that was easiest to find misregistrations.

Figure 5.2: Photo of the Minolta Vivid 700 Range Sensor [1]

We acquired "FacesPat1" and "FacesRick1" with our sensor. Both of these proved to be smoother data sets than the Buddha images, however they provided more prominent noses and more complex ear structures.

All of the above range images plus many others are available in our range image database located in the OSU SAMPL web site at http://sampl.eng.ohio-state.edu/.

5.1.2 Decimation Factor

We tested our experiments at different uniform decimation factors, including 1, 2, 4, 8, 16, and 32. Larger decimation factors indicate smaller tested images. For example, a 200x200 range image decimated by a factor of 4 becomes a 50x50 range image where every fourth row and column from the original is selected.

5.1.3 Outlier Classifiers and Parameters

Three main point weight analyzer variants were used for our tests to perform outlier classification. First, the base algorithm introduced by [4] that classifies no points as outliers was used. It requires no parameters. The `vtkMaxDistPointWeightAnalyzer` and the `vtkVariancePointWeight-Analyzer` based on [11] and [14], respectively, were also used. These latter two were provided the same

set of values for their `ExpectedError` and `GoodRegistrationDistance` values, respectively. We used a base value of `3.5492mm` for each of these, plus several multiples of this value. This base value was obtained from the mean edge length of several initial data sets at a decimation factor of `2`.

5.1.4 Loop Exit Criterion Threshold

As described in section 4.3, the loop exit criterion threshold is the maximum difference between successive registration errors required to consider an ICP sequence to have converged. We chose the loop exit criterion values of `0.3`, `0.03`, and `0.003` somewhat arbitrarily because they yielded good results with initial tests. Except when analyzing the effects of the criterion value, the tightest threshold, `0.003` was generally used for qualitative tests.

5.1.5 Experiment Runs

After generating the test configuration files, four 450MHz Pentium II™ machines with 256MB of RAM were used to perform batch experiment runs. The vast majority of the tests were performed on three machines configured specifically for performing the tests and had minimal software installed. We collected data from a total of 7,699 tests for analysis, which consists of four different input objects with a total of 18 view pairs. For each of these tests, we recorded the configuration, the registration history described in the section 4.4, and the total execution time. From the results files, we compiled a database of the final registration information, the execution time, and the number of iterations required for each test.

5.2 Analysis Methodology

After performing batch run tests of our registration experiments, we collected the results and examined key individual tests in the GUI. Most of the examinations performed in the GUI were made to characterize how the outlier classifier choices affect the quality of the registration. Typically, the no-outlier results were viewed as a baseline, then results from the two classifiers were viewed. The following criteria were generally used to judge the quality of a registration in the GUI:

1. Are there any gross registration errors?

2. Are there any mismatched edges?
3. Are there "splotchy" sections?

Gross registration errors consist of registrations that are completely wrong. For example, Figure 5.3 shows two face range images where the nose from one is pointing out the ear of the other. Occasionally, obvious registration errors such as these correspond to very low registration error values because, even though the registration is incorrect, the registration produces a low mean squared error.

If the entire registration is not obviously incorrect, we next looked for key feature areas such as ears and noses on the face images because they were easy to examine with polygonal rendering. Generally these feature areas would have edges in one or more range images allowing us to more easily see between the two range images. Often, we instructed the GUI to draw lines connecting the corresponding point pairs via a `vtkClosestPointsConnector` filter such as was shown in Figure 4.15. Figure 5.4 shows a case where the angel's right wing is misregistered. The pink edge of the wing intersects the cyan edge instead of being aligned with it.

Finally, if there were no problems found in these feature areas, we examined large areas with relatively constant curvature. Given a perfect registration of range images that have Gaussian noise, we expected that the two surfaces would cross over each other often, creating a "splotchy" surface as in Figure 5.5. A worse registration would not have this characteristic because the two surfaces would be too far away to have this interleaving, as shown in Figure 5.6. Thus, we generally considered slightly splotchy surfaces to have a better registration than registrations displaying large expanses of non-interweaving range image sections.

Using these criteria, we qualitatively analyzed the registration results to determine which algorithm variants worked the best, and under what conditions. Additional quantitative results were analyzed. Key findings from those analyses are found in section 5.3.

5.3 Experimental Results

Upon examining our results, we determined the key effects of each of the experiment permutation variables. We have broken down the results of these tests into the following categories:

1. Effects of Outlier Classifier Type and Parameter Settings

2. Effects of Decimation
3. Effects of the Loop Criterion Threshold Value

For each of these effects sections detailed below, we will highlight key similarities and differences with the data sets we tested.

Figure 5.3: Snapshot of the FacesPat1_000U01-144U01_N_0003 test. Demonstrates a catastrophic failure of the registration when large non-overlapping regions exist and no corresponding point pairs are classified as outliers

Figure 5.4: Snapshot of the Angel_000U02-060U02_M08873_0003 test. Demonstrates a mis-registration at the edge of the wing in the circled region.

Figure 5.5: Snapshot of the FacesRick1_000U01-036U01_V01109125_0003 test. Demonstrates good "splotching."

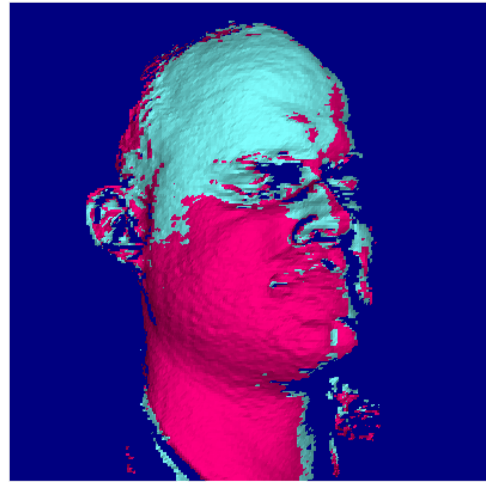


Figure 5.6: Snapshot of FacesRick1_000U01-036U01_M0221825_0003 test. Demonstrates poor "splotching."

5.3.1 Effects of Outlier Classifier Type and Parameter Settings

One aspect that significantly affects the number of iterations and the total execution time of a test is whether the ICP process is deemed invalid before it converges. When no classifier is used, 84% of the tests converged before an ICP iteration resulted in a worse registration than the previous iteration. When either of the two outlier classifiers were used, this rate drops dramatically to approximately 25%. We hypothesize that the following is happening:

1. The outlier classifier marks outlier point pairs based on a registration.
2. The next registration is calculated and applied given the set of inlier pairs. This registration results in a small movement of the range image.
3. The outlier classifier marks outlier points, but this time it selects a significantly different set of pairs as outliers.
4. The next registration is calculated and applied on the new set of inlier pairs. The different set of inliers results in a registration error greater than that calculated in step #2.

We have found that often if we do not check that each ICP iteration's error value is less than the previous iteration's, the ICP cycle enters an infinite loop, apparently jittering between different sets of inliers and outlier alternately pulling the registration different directions upon different ICP iterations.

Regardless of whether the ICP sequence was deemed to have converged, we found the effects of using an outlier classifier to be generally as expected. When nearly the entire surface from the range image being registered overlaps the other range image, we found that not using any classifier produced the best results. We found this situation to be the case for the "Angel" test set as well as for range image pairs that were only separated by a small angle.

The "Angel" test set approximated Besl's original requirement of always registering a subset of an object to the object in the following way. The base range image to which the other range images were registered was the frontal view of the angel. This view captures the face, wings, and front of the body, only missing some of the sides of the body that form oblique angles to the camera. All other views could not see much more of the original data because the wings blocked the sides and the back.

For the other data sets, as the amount of non-overlapping data increases, the classifiers become more important. For most of the tests examined, we found that the `vtkVariancePointWeightAnalyzer` performed the best, though for many tests, it was only slightly better than the `vtkMaxDistancePoint-WeightAnalyzer`.

A major deficiency in the algorithms we tested was that they generally had difficulty registering certain human-identifiable features of high curvature changes and edges of range images. For example, the tips of the ear lobes in the face tests would generally be closely registered for the most successful tests, but often the curves at those tips did not match correctly. We expect that including the smoothed normals as [6] suggests would assist in obtaining better results.

One rather startling discovery we made was that the `vtkVariancePointWeightAnalyzer` was performing very well, but sometimes for the wrong reasons. [14] claims that the histogram of the Euclidean distances between corresponding point pairs of a moderately well registered range image pair is distributed as a Gaussian curve. Often we found this to be the case, as is shown Figure 5.7. On the other hand, for the "FacesPat1" and "FacesRick1" images, the right leg of the histograms tend to be elevated, especially when large non-overlapping regions are present.

Consider the curves in Figure 5.8. Both have Gaussian noise and are perfectly registered. If we examine the overlapping region, the section that looks like a carat (^), we can plot a histogram of the distances between corresponding point pairs. This histogram would be a Gaussian curve due to the noise in the data. If we examine the straight segment on the dashed curve, then every point on that segment will match with the left tip of the dotted carat section. This produces a histogram resembling a pulse function. When these two histograms are added, we get the lower histogram shown in the figure. In this case, the standard Gaussian fit leads us to erroneously base our outlier classification by the arithmetic mean rather than the peak of the Gaussian section.

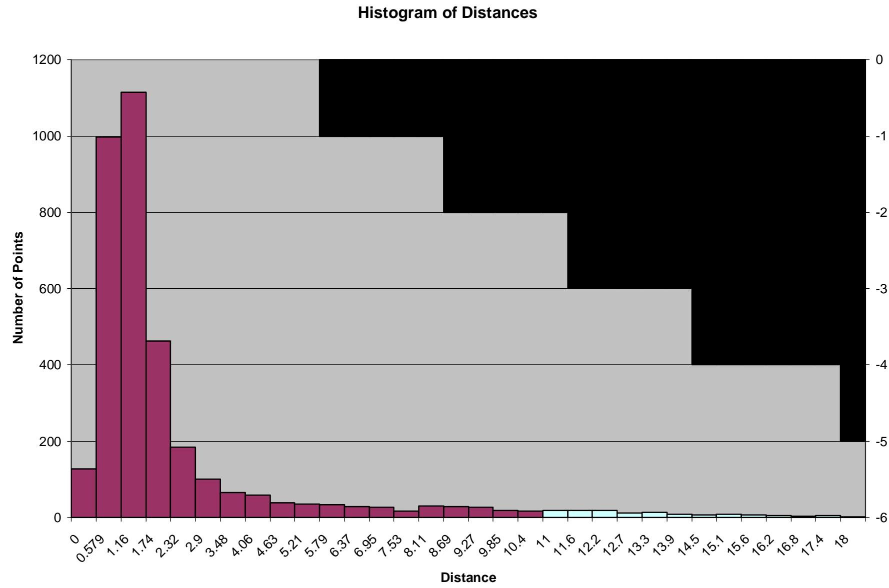


Figure 5.7: A typical histogram of the distances between corresponding point pairs for a final registration. Note that the histogram resembles a Gaussian, as hypothesized by [14]. The histogram columns are shaded according to whether the points in that histogram column have been classified as inliers or outliers. This histogram is from the Angel_000U02-040U02_V35492_0003 test.

In our "FacesPat1" and "FacesRick1" tests, we found that the above situation happened nearly all of the time. Additionally, a typical arithmetic mean of the corresponding point pair distances was near 30mm. With an expected error parameter setting ranging from 0.1109125mm to 7.0984mm, this mean causes the registration to be considered so poor by [14] that the fallback classifier method must be used. As shown in Figure 5.9, this effectively results in all point pairs corresponding to the right of the main peak in the histogram to be classified as outliers. These results demonstrate a theoretical deficiency in Zhang's outlier classifier when there are large non-overlapping sections. Further experiments are required to determine whether using a more accurate histogram curve fit would allow us to more correctly extract the inlier corresponding point pairs.

5.3.2 Effects of Decimation

In general, we found that lower decimation factors produced better results, though they required greater execution time. In particular, the simplistic brute-force nearest-neighbor search used for finding closest point pairs proved particularly slow for undecimated data sets due to its $O(N^2)$ complexity. For example, the `Buddha_300U01-000U01_N_0003` test took approximately 1 minute, 45 seconds to load into the GUI. The bulk of this time was spent performing the nearest neighbor search. After breaking up the input points into uniformly sized bins to improve search speed, we saw this loading time decrease to 40 seconds. In order to not skew the timing results for later tests, the original $O(N^2)$ algorithm was used for all tests and the enhanced search was only used for the GUI visualization. We believe that a more sophisticated search such as a kd-tree based search would yield much faster results.

As for the registration quality, when each test is viewed with the decimation factor used to perform the registration, the results tend to be quite good. Unfortunately, the uniform decimation is sub-optimal for preserving the original shape. If a registration performed on decimated data is viewed with undecimated data, the more subtle features of the shape show up and expose mis-registration problems. We found that because the initial course hand registrations were close enough to the correct registration that decimation factors above 2 produced unsatisfactory results. Higher decimation factors generally resulted in fine registrations that were qualitatively worse than the original course registration, when viewing the results with undecimated data.

We did not find any strong correlation between the decimation factor and how the outlier classifier parameters affected whether an ICP iteration terminated successfully or became invalid

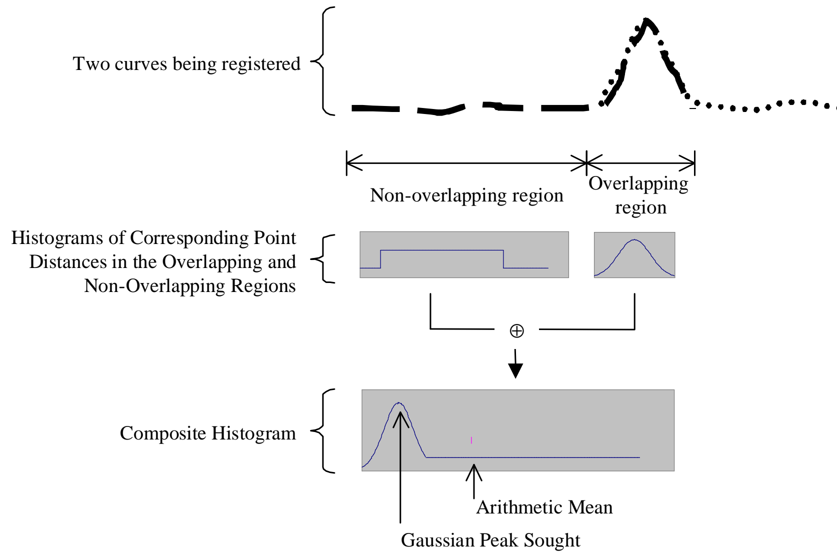


Figure 5.8: Hypothesized source of non-Gaussian histograms of distances between corresponding point pairs

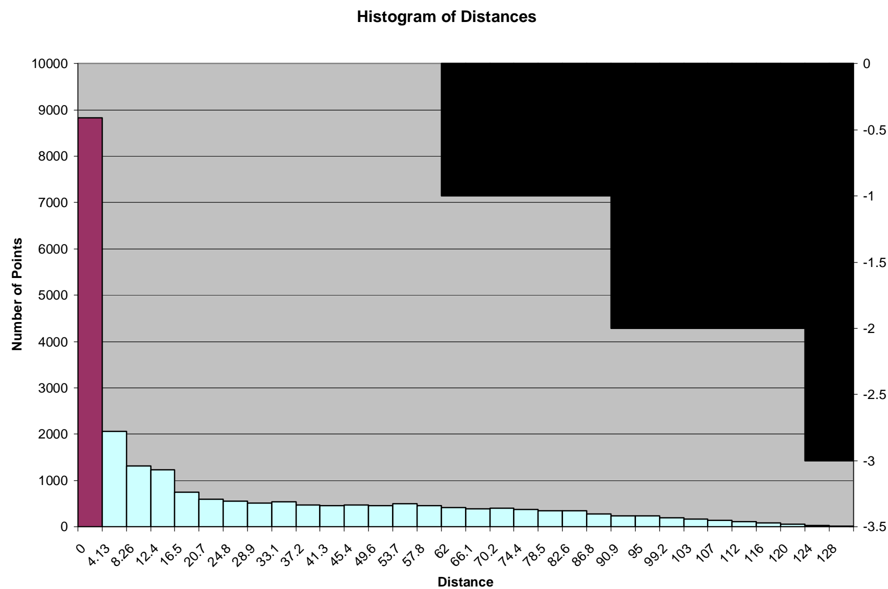


Figure 5.9: A histogram of the distances between corresponding point pairs for a final registration typical of tests where there are large non-overlapping regions. The histogram columns are shaded according to whether the points in that histogram column have been classified as inliers or outliers. This histogram is from the FacesPat1_000U01-144U01_V01109125_003 test.

5.3.3 Effects of the Loop Criterion Threshold Value

Modifying the loop criterion produced predictable results. As tighter criteria were used, we noticed greater refinement in the registration path being followed. If the ICP algorithm was moving toward an incorrect registration, a tighter criterion simply allowed it to move in closer to that incorrect registration. Additionally, the tighter the criterion, the greater chances there were that the ICP sequence would become invalidated.

5.4 Summary and Conclusions

In this chapter, we have described our test setup and the generation of the 7,699 tests for our experiments. We have proposed a method of analyzing the quality of range image registrations, and given our analysis of our experiments using this method.

In that analysis, we found that for range image pairs that approximate Besl's requirement of full-overlapping, using no outlier classifier generally yielded the best results. For those pairs that had significant non-overlapping regions, both of the classifiers generally yielded good results, with the classifier based on Zhang's work performing slightly better in most cases than the one based on the work of Schütz.

We also found that although decimated data could be registered, those registrations tend to only be "good" in the context of their decimation. Once the range image pair is viewed undecimated, the registration imperfections readily manifest themselves. On the flip-side, due to our rather brute-force approach to finding closest point pairs, decimation had an extremely significant impact on the execution time required.

Additionally, we found that modifying the threshold for determining convergence of an ICP sequence had predictable results. As that threshold is lowered, the sequence simply goes further along the path it is following unless it first encounters numerical or algorithmic instabilities.

Through performing the tests and analyses, we benefited from the design goals that we followed. The hooks for instrumentation allowed us to gather the required data from our experiments. Further, we were

able to add new variants as necessary to our test-bed due to its extensible nature. Internally, we reused code as we developed classes such as the `vtkClosestPointsConnector` that shares the point correspondence filters with the `vtkICPAlgorithm` class, and as we built multiple applications that shared the library classes discussed in this thesis. In the next chapter, we will examine ways in which the work we have done may be expanded even further and ways in which we may build upon the observations we have made here.

CHAPTER 6:

FUTURE WORK

While developing and using the test-bed, we encountered numerous areas in which our work may be extended and enhanced in the future. Some of these ways include further improving the test-bed software infrastructure, implementing and testing additional existing registration algorithms and variants, and developing novel techniques and approaches to the registration problem.

When we began developing the batch processing application, we encountered an infrastructure issue: serialization and deserialization of the registration pipeline configuration and results. In this context, serialization is the process of converting a complex set of objects into a stream of bytes typically stored on disk. Deserialization is the inverse process. We used the CSV-based approach introduced in subsection 4.4.3 due to its simplicity and our ready access to tools such as Microsoft Excel™ and PERL that could easily process CSV files. Unfortunately, this approach required hand-coding of each serialization and deserialization routine. As support grows in VTK, we expect to be able to enhance the software to support automatic recognition of new registration and helper classes in libraries and automated serialization and deserialization.

Related to this infrastructure work, there are many other registration algorithms and variants that we have neither implemented nor specifically tested. For example, Turk's non-binary point weight classifier [13] may prove to yield better results around points that have a high probability of sensor error. The spring-mass approach used by Eggert [6] and others may prove to be much faster than a pure ICP approach. By including not only the position, but also the normal in the corresponding point distances used for the registration, our results may improve significantly around features of high curvature.

Finally, we would like to further characterize the distribution of errors due to sensor noise. We expect that we will find the hypotheses presented in subsection 5.3.1 to be correct. The distribution of

corresponding point distances may be approximated by a Gaussian curve from the overlapping regions of the range image plus a step function from non-overlapping regions when the object has strong globally smooth curvature. If this theory holds true, we believe that we can develop more robust point classifiers that can more effectively extract the corresponding point pairs that actually belong to overlapping region in the range images.

BIBLIOGRAPHY

[1] "Minolta Corporation, ISD – 3D Scanner". *Minolta*. http://www.minoltausa.com/vivid/index.asp (31 July 2000).

[2] *Signal Analysis and Machine Perception Lab*. The Ohio State University. http://sampl.eng.ohio-state.edu/ (31 July, 2000).

[3] *VTK Home*. http://www.kitware.com/vtk.html (31 July, 2000).

[4] P.J. Besl and N.D. McKay. "A Method for Registration of 3-D Shapes". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239-256, Feb. 1992.

[5] Y. Chen and G.G. Medioni. "Object Modeling by Registration of Multiple Rnage Images". *IVC*, 10(3):145-155, 1992.

[6] D.W. Eggert, A.W. Fitzgibbon, and R.B. Fisher. "Simultaneous Registration of Multiple Rnage Views for use in Reverse Engineering of CAD Models". *CVIU*, 69(3):253-272, March 1998.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub. Co., 1995.

[8] A. Lorusso, D.W. Eggert, and R.B. Fisher. "Estimating 3-D Rigid-Body Transformations: A Comparison of Four Major Algorithms". *MVA*, 9(5-6):272-290, 1997.

[9] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Prentice Hall PTR, 2nd Edition, 1998.

[10] W. Schroeder and K. Martin. *The vtk User's Guide*. Kitware, Inc., June 1999.

[11] C. Schütz, T. Jost, and H. Hügli. "Semi-Automatic 3D Object Digitizing System Using Range Images". *Proceedings of Asian Conference on Computer Vision*, Jan. 1998.

[12] I. Stamos and P. Allen. "3D Model Construction Using Range and Image Data". *Proceedings of CVPR00*. I:531-536. 13-15 June 2000.

[13] G. Turk and M. Levoy. "Zippered Polygon Meshes from Range Images". *Proceedings of SIGGRAPH* '94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics Proceedings, Annual Conference Series*, 1994. *ACM SIGGRAPH*, pp. 311-318.

[14] Z.Y.Zhang. "Iterative Point Matching for Registration of Free-Form Curves and Surfaces". *IJCV*, 13(2):119-152, Oct. 1994.

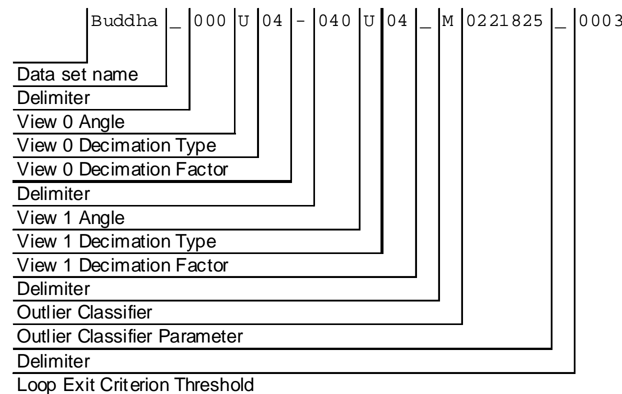
APPENDIX A:

HARDWARE AND SOFTWARE TOOLS USED

- Hardware Tools
    - 450Mhz Pentium II with 256MB memory
    - Minolta Vivid 700 Non-Contact 3D Digitizer (http://www.minolta.com/)
- Software Tools
    - Microsoft Visual Modeler™
    - OSU range image database (http://sampl.eng.ohio-state.edu/)
    - Microsoft Excel™ for numerical analysis
- Languages and APIs
    - C++
    - VTK™ (http://www.kitware.com/vtk.html)
    - MFC™ for the GUI
    - PERL for support scripts (http://www.cpan.org/)
    - HTML for debugging output

APPENDIX B:

TEST NAMING CONVENTIONS



- **Data set name:** Human readable logical name of the data set used. In this paper, the data sets are "Angel", "Buddha", "FacesPat1", and "FacesPat2"are used.
- **Delimiter:** A combination of underscores and dashes are used as delimiter characters to make machine and human parsing of the test names easier.
- **View Angle**: Indicates an approximate view angle about the vertical axis. For example a test with view angles of 000 and 040 would contain two range image views separated by approximately 40 degrees. The narrowest view angle used in these tests was 20 degrees and the widest view angle was approximately 126 degrees. Note that for the "FacesPat1" and "FacesRick1" data sets, the names of the view angles are actually double the value described here (e.g. the 126 degree view of "FacesPat1" is labelled as 252).
- **View Decimation Type:** For a given view, indicates what type of decimation was used. For the tests we are reporting here, only uniform decimation was used, indicated by a "U".
- **View Decimation Factor:** Indicates how much decimation was used. In the example above, "04" indicates that every fourth row and column of the original range image was used.
- **Outlier Classifier:** Indicates which outlier classifier was used. "N" stands for none, "M" for vtkMaxDistancePointWeightAnalyzer, and "V" for vtkVariancePointWeightAnalyzer.
- **Outlier Classifier Parameter:** If the outlier classifer was not "N", this gives the classifier parameter. To interpret these values, insert a decimal point after the first digit. For example, "M0221825" indicates that a vtkMaxDistancePointWeightAnalyzer was used with the ExpectedError value set to 0.221825.
- **Loop Exit Criterion Threshold:** Indicates the maximum difference between successive registration errors required to consider an ICP sequence to have converged. As with the outlier classifier parameter, a decimal point is logically inserted after the first digit, so "0003" is interpreted as 0.003.

Figure B.1: Naming conventions used for our experiments. See section 5.1 for additional details on the tests performed.