Accelerating 3D registration using multi-core CPU and GPU cluster

Shridhar Mohandoss

A Thesis

in

The Department

of

Computer Science

and

Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Software Engineering at

Concordia University

Montreal, Quebec, Canada

August 2012

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Shridhar Mohandoss**

Entitled: **Accelerating 3D registration using multi-core CPU and GPU cluster**

and submitted in partial fulfillment of the requirements for the degree of

M. A. Sc. (SOEN)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____Chair
Dr. N.Tsantalis

_____ Examiner
Dr. T.Fevens

_____ Examiner
Dr. R.Jayakumar

_____ Supervisor
Dr. D.Goswami

_____ Supervisor
Dr. S. P. Mudur

Approved by _____
Chair of Department or Graduate Program Director

_____
Dr. Robin A.L. Drew, Dean
Faculty of Engineering and Computer Science

Date _____

Abstract

Accelerating 3D registration using multi-core CPU and GPU cluster

Shridhar Mohandoss

Three dimensional models (3D) are becoming popular due to different fields like 3D printing, games, graphics and movies. 3D registration is the process of aligning the different range images (3D scan data) obtained from the scanner to create the complete model. Due to the large size of the 3D scan data, registration is a time consuming process. Today most of the computers have a multi-core CPU (Central Processing Unit) and a GPU (Graphical Processing Unit), providing us the opportunity to accelerate the 3D registration process using multi-core CPU and GPU. As a part of our research we have studied the problem of accelerating 3D registration using CPUs and GPUs. While most existing methods focus on using a single GPU to accelerate 3D registration, we have proposed a method to accelerate 3D registration using a cluster of multi-core CPUs and GPUs. To demonstrate the performance of our method we have implemented the 3D registration system on a cluster with 20 CPU cores and 5 GPUs. We observed a speed-up in registration time of up-to 15 times when compared to registration on a single CPU core. To the best of our knowledge, ours is the first attempt to accelerate 3D registration using a multi-core CPU and GPU cluster. Finally, we have compared the performance and the accuracy of our registration system with an open source registration system.

Acknowledgements

I would like to thank my supervisors Dr. Sudhir P. Mudur and Dr. Dhrubajyoti Goswami for their guidance, support and encouragement. I am particularly thankful to Dr. Mudur for all his insightful suggestions which have been a great help through the course of my research. Dr.Goswami's friendly nature and the discussions we had about parallel programming have also been very helpful during my work.

I would like to thank Dr.Radhakrishnan for his guidance not only in academics but also in all aspects of life.

Finally, a special thanks to my parents, family and friends for their love, support and trust.

Table of Contents

List of tables

List of figures

Chapter 1: Introduction

1.1. 3D registration

Three dimensional models (3D) are becoming popular due to different fields like 3D printing, games, graphics and movies. The low-cost availability of 3D scanners, especially Microsoft Kinect[1], has made it easier to obtain 3D data of various objects. Typically, to obtain the complete geometry of an object in 3D, multiple scans are required. So the model will be scanned from several different view-points and different facets of the object will be visible from the scanner. The raw data obtained from the scanning operation consists primarily of three dimensional point clouds called "range-images" [1]. Each of the range images obtained from the scan will be in its own co-ordinate system. These range images need to be merged to construct a "3D model" of the object. The process of transforming the different coordinates associated with different scans, to bring all the range images into a single co-ordinate system is called "registration". If the scanner and object positions for all the scans are recorded using a calibrated scanning set-up, then the registration process is simple. But in general calibration is not possible, especially when using a hand-held scanner or when scanning large objects. So, aligning the range images to a single co-ordinate system is based on the content of the range images. The transformation to register two range images can be easily determined by matching the common features between the two range images. However in most cases noise, scanning resolution and self-occlusion makes it difficult to find the common features. So, most registration techniques assume that a rough initial alignment is known and try to register by using some numerical optimization technique to minimize the

[1]Microsoft Kinect: Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs. The kinect includes a depth sensor, RGB camera, a microphone, etc.

alignment error. As with any numerical optimization technique, the registration process also requires a good initial estimate of the transformation to find the final transformation and to converge quickly. The registration process also assumes a good common overlap between the range images to be registered.

There are several registration algorithms reported in the literature and they can be classified based on multiple factors, like, whether the algorithm assumes an initial approximate transformation or not, number of data-sets registered simultaneously, and the error metric used etc. In this thesis we consider the classification based on the number of data-sets registered simultaneously.

"Pair-wise registration" is the process of registering two range images simultaneously. Most registration methods reported in the literature concentrate on pair-wise registration. To create a complete model, pair-wise registration is chained all over the range images [5]. Figure 1-1 shows how a set of range images will be registered in a pair-wise fashion to create a 3D-model.

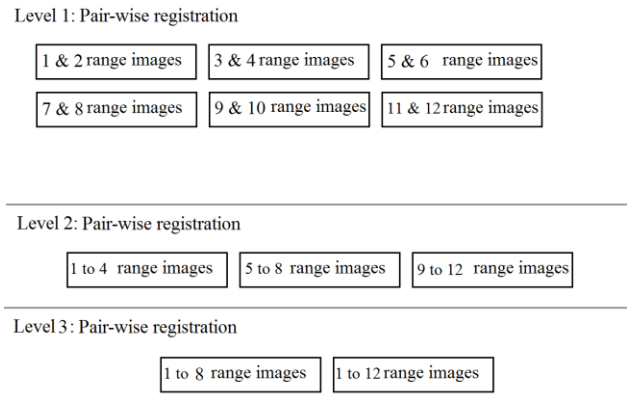

Figure 1-1: Pair-wise registration

One of the main problems with pair-wise registration is that small error in registering each pair of range images can grow cumulatively and result in a larger total error [5, 6].

"Multi-view registration" is the process of registering more than two range images at the same time. Multi-view registration reduces the accumulation of error by distributing the error among all the range images registered simultaneously. To create a complete 3D model, multiple range images with a common overlap are first registered. Then the registered range-images are combined together using pair-wise registration. Figure1-2 shows an example situation of how a set of range images are registered in a multi-view fashion to create a 3D-model.

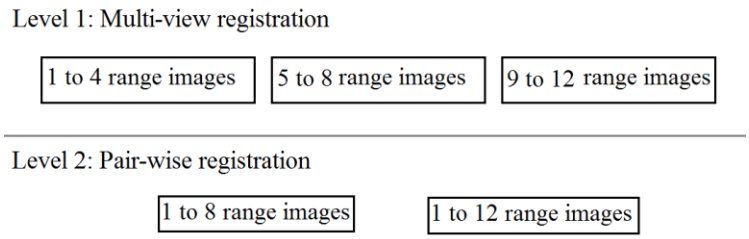

Figure 1-2: Multi view registration

1.2. Need for accelerating 3D registration problem:

"Even though many fast range sensors are available and their computing power is increasing nowadays, 3D registration is still a time-expensive task" [2]. One reason for this is the number of scans to be merged in the generation of the complete model, typically, is in the order of tens or hundreds, and the number of points in each scan could be in the order of thousands to tens of thousands. Typical values from the literature [30] are given in Table 1-1 below. Due to the large amount of time involved, registration has been an offline process. In offline registration all the range images are acquired and saved, then the registration is done as a separate process. One drawback of offline registration is that when the data is noisy registration cannot be done. So re-scanning of the model is needed causing considerable delay in obtaining a model.

Model	Number of points	Number of range images	Source
Bunny	350,000	10	Stanford repository
Thai statue	19,400,000	36	Stanford repository
Lucy	14,027,872	47	Stanford repository
Happy Buddha	543,652	60	Stanford repository
Dragon	566,098	70	Stanford repository
Armadillo	3,390,515	114	Stanford repository
Vase	150,000	10	Scanned in our lab
Pot	265,000	12	Scanned in our lab

Table 1-1: List of models and the total number points in each model.

Due to the continued development of the technology in multi-core processors and low cost Graphics Processing Units or GPUs, it has become possible to accelerate the registration process by using hardware accelerators.

1.3. Hardware accelerators for registration

There are several hardware accelerators like multi-core CPU, GPU, FPGA, and IBM cell processor, etc. available in the market. As GPUs and multi-core CPUs provide the highest computing power-per dollar, consume less space and power compared to the other hardware accelerators [4], we have explored the use of GPUs and multi-core CPUs for accelerating registration in this thesis.

1.3.1. GPU:

A graphics processing unit (GPU) is a special purpose processor designed to perform graphic operations. The emergence of programmable GPUs in early 2000's made it possible to use GPUs for traditional computation handled by the CPUs. The low-cost and specialized computational

capabilities of GPUs attracted many researchers to use GPUs to accelerate various computationally expensive tasks. The field of using GPUs to accelerate the traditional computation is known as general purpose graphics computing (GPGPU). In the early days of GPGPU, graphics API's such as OpenGL and DirectX were the only way to interact with GPUs. So, to perform general purpose computations on GPUs, the computations were expressed as rendering tasks. In essence, GPU was tricked into performing general purpose computations by making those tasks appear as rendering tasks [7]. If anyone wanted to use GPUs for general purpose computing they had to express the computations in graphics programming languages. This convoluted programming model of GPGPU deterred many developers.

In early 2007 Nvidia introduced a new parallel programming model called CUDA for GPGPU on Nvidia GPUs. The CUDA architecture allowed the developers to give general purpose computational tasks to GPUs without the need for expressing the computations as rendering tasks. This greatly simplified GPGPU and promoted wide adoption of Nvidia's CUDA architecture. Furthermore, Nvidia released a series of GPUs with tailored execution units to suit general purpose computational needs.

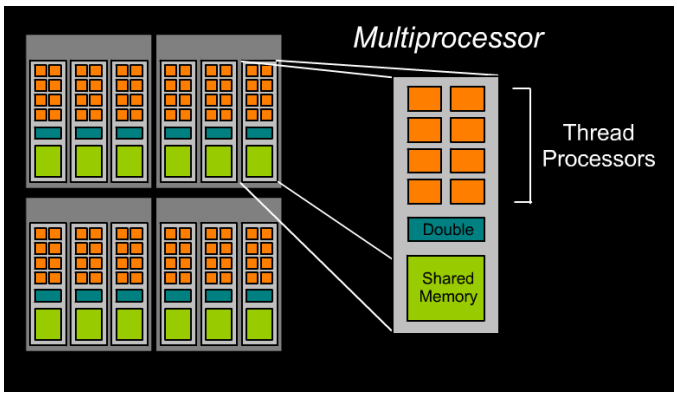

Figure 1-3: Architecture of Nvidia GPU [35]

GPUs in principle are “stream processors” or process a data stream, so they have large number of RISC (Reduced Instruction Set Computer) processing cores. Thus GPUs fall into the SIMD (Single instruction multiple data) class. Figure 1-3 show the architecture of an Nvidia GPU. Here, a single multiprocessor has 8 thread-processors (RISC processing element), one double-precision unit; and shared memory with every RISC unit to enable thread to thread communication within the same multiprocessor. The general purpose computations executed on GPUs are known as “kernels”. CUDA kernels have the following properties:

1. A CUDA kernel is executed as an array of threads. All the threads run the same code and the threads are identified by a thread ID.

2. A kernel is launched as a grid of thread blocks. Thread blocks are executed on multiprocessors, CUDA schedules the thread blocks on the multiprocessors during run-time. CUDA supports up to 65535 thread blocks on a single GPU [35].

A thread is executed in thread processor. CUDA threads are light weight threads and have little creation overhead. So, CUDA uses thousands of threads to hide latency [33].

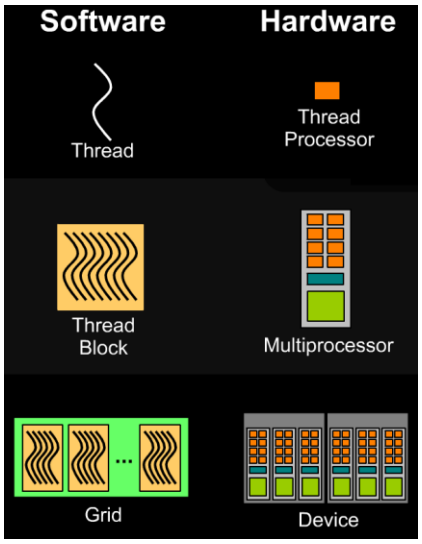

Figure 1-4: Nvidia CUDA's execution model [35]

1.3.2. Multi-core CPU

Over the past 40 years the CPU clock speed has continuously gone up. However increasing CPU clock speed also increases the power consumption. The graph in Figure 1-5 shows the power consumption trend of Intel processors from 1985 to 2000. We can observe the general trend is the power consumption increases by a factor of 2X every four years. The graph in figure 1-6 shows the cost of cooling solutions. The increase in power consumption and the cost of cooling (solution for the increased thermal problem) was a bottleneck in increasing the performance of the processors. Meanwhile the number of transistors in the processor has also been growing over the past years. To make use of the extra transistors and increase the processor performance per watt, the chip makers came up with the design of having multiple cores at a lower clock frequency, instead of a single core processor with higher clock frequency. This has resulted in the emergence of "multi-core processors"

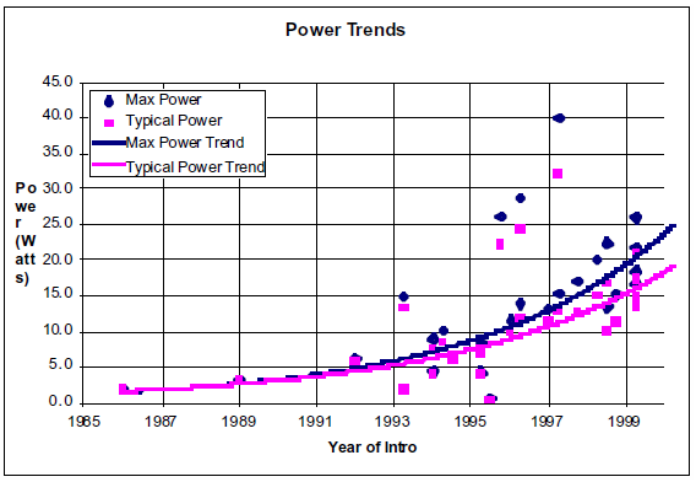

Figure 1-5: Processor performance per watt over the years [34]

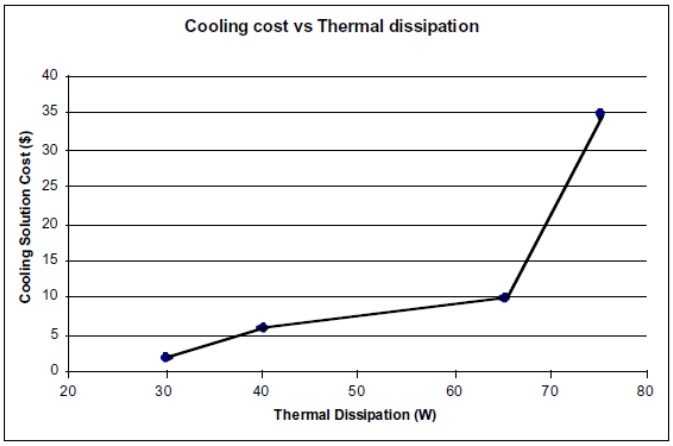

Figure 1-6: Cooling cost VS Thermal dissipation [34]

The multi-core processors can be classified in to homogeneous and heterogeneous multi-core processor. As the name implies, homogeneous multi-core processor's use one core design repeatedly, while heterogeneous multi-core processor's use a mix of different cores. Figure 1-7 shows the architecture of a homogeneous multi-core processor. All the cores are embedded in a single processor chip, with each core having its own levels of cache memory. Since core is a fully functional unit, multi-core processors support MIMD (multiple instruction multiple data) type of parallel processing.

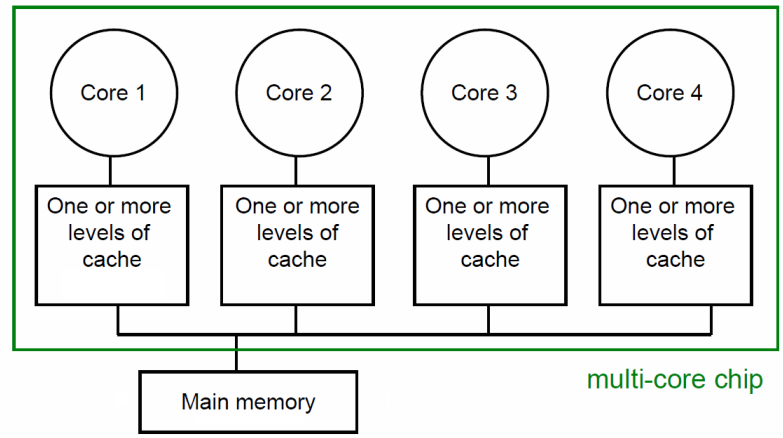

Figure 1-7: Multi-core architecture [34]

Existing parallel programming libraries like MPICH2 and OpenMP can be used for programming any multi-core processors [33, 34]. Intel introduced a new C++ library, known as TBB for programming Intel's multi-core processors. The multi-core processors have the following properties:-

1. Multi-core is a shared memory multiprocessor.

2. Each core can run multiple threads. Since multi-core processors are of MIMD type, the threads can run different code.

3. The number of cores on the multi-core CPU is typically less than 10. And since, the CPU threads have a high thread creation and thread management overhead, to achieve efficiency multi-core CPUs run few threads.

1.4. Comparison of GPUs and multi-core CPUs

The Table 1-2 we created shows the main differences between multi-core CPUs and GPUs.

Feature	**GPU**	**Multi-core CPU**
Type of processor	Stream processor	Shared memory multiprocessor
Architecture	SIMD, so GPUs can handle only data-parallelism	MIMD, so multi-core CPUs can handle both data-parallelism and task parallelism.
Number of processing cores	Up to 512 processing cores	A typical commercially available Multi-core CPU has less than 10 cores.
Number of threads	To achieve high efficiency GPUs run 1000's of threads.	Multi-core CPUs run few threads.
Memory bandwidth	Up to 148 GB/s	Up to 37.0 GB/s
Cache memory	8/16 KB per multiprocessor	1MB to 24MB

Table 1-2: A comparison of GPU and multi-core CPU

1.5. Objectives of this Thesis:

In [26] a pair-wise and a multi-view registration algorithm called "distance based registration" has been proposed. That algorithm was run on a single core CPU. In this thesis we have extended the "distance based registration" algorithm to explore the use of multi-core CPUs and GPUs to accelerate the registration process. In particular, we have examined the following four aspects:

a) Devising a functionality distribution scheme for a single computer with one multi-core CPU and one GPU, to accelerate registration. To devise the proper functionality distribution, we experimentally compared three different distribution schemes.

b) Thoroughly exploring the use of a cluster of multiple multi-core CPU and multiple GPUs for accelerating the registration algorithm. This novel exploration is not done by others, to the best

of our knowledge. To distribute the registration process we redundantly create a map of all the computations in each node of the cluster. Then based on the map we divide the computations equally to all the nodes. We have also compared the accelerated registration with the base line registration on single CPU core and measured the speed-up

c) Experimentally comparing the efficiency and speed-up of pair-wise registration with multi-view registration using various combinations of multi core CPUs and GPUs.

d) Experimentally comparing the accuracy and speed-up of our method with a state of the art GPU accelerated registration algorithm called KinFu. [35]

1.6. Outline of the Thesis:

The remaining Chapters of the thesis are organized as follows:

In chapter-2, we provide a review of the related work in 3D range image registration and accelerating range image registration using hardware accelerators. In chapter-3, we present our extensions of distance based registration algorithm to accelerate pair-wise registration and multi-view registration using a single GPU and a single multi-core CPU. In chapter-4, we present our method to accelerate pair-wise registration and multi-view registration, using multiple multi-core CPUs and multiple GPUs. In chapter-5 we present our experimental results comparing pair-wise and multi-view registration. Finally in chapter-6, we conclude by summarizing the benefits and pay-offs and suggest avenues for future work. In summary, this thesis is an experimental and implementation oriented study of accelerating pair-wise and multi-view registration using multi-core CPUs and GPUs.

Chapter 2: A Brief review of registration techniques.

2.1. Feature Based Registration

The early attempts to register the range images involved matching common features in the overlapping range images to find the transformation between the range images. This type of registration is known as feature based registration. Feature based registration generally has two steps.

- First, creating a representation of the surface using some of the "objects features" such as surface normal, texture, and curvature etc.
- Second, matching the selected features to get the transformation parameters.

The various feature based registration approaches differ mainly in the techniques used to represent the surface. Yamany et. al [8] represented an object using a set of surface signatures. The signatures computed at each 3D point encode the surface curvature seen from that 3D point. The key idea is to use the curvature information and create a reduced representation of the surface at certain points. Since the curvature information is an invariant of transformation the transformation parameters can be obtained by matching the curvature information. The signature image is generated as follows:

As shown in Figure-2-1, for each point P, denoted by its 3D coordinates and the normal UP, each other point Pi on the surface can be related to P by two parameters:

1. The distance di = || P - Pi ||
2. The angle $\alpha_i = cos^{-1}\left(\frac{\vec{U}_P.(P-P_i)}{||P-P_i||}\right)$

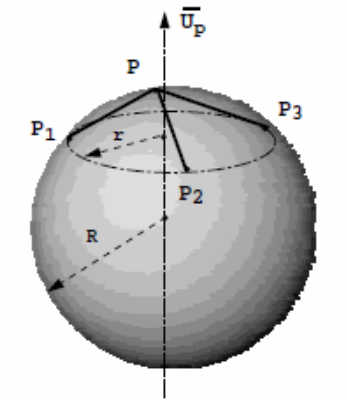

Figure 2-1: A sphere showing the calculation of surface curvature. [8]

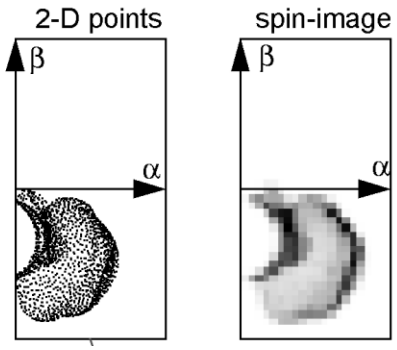

Figure 2-2: 2D representation of 3D points and the spin image created from the 2D representation. [7]

Hebert [7] represented the surface shape using a dense collection of 3D points and surface normals. In addition, each surface point was associated with a descriptive image (called spin image) that encodes global properties of the surface. The points, normals and associated images together make up the surface representation. Two cylindrical coordinates can be defined with respect to a selected 3D point and its normal: the radial coordinate 'a', defined as the perpendicular distance to the line through the surface normal, and the elevation coordinate 'b', defined as the signed perpendicular distance to the tangent plane defined by vertex normal and position.

A spin image is created by representing the points around a 3D point using a and b. This is the equivalent of projecting the 3D-points on to a 2D plane. Next, bilinear interpolation is used to smooth the projected points. The result can be thought of as an image where dark areas in the image correspond to areas that contain many projected points. This procedure is repeated for a set of selected 3D points and a set of "spin images" are created. Spin images created for different range images are matched to find the matching point pairs, from which the transformation parameters are extracted.

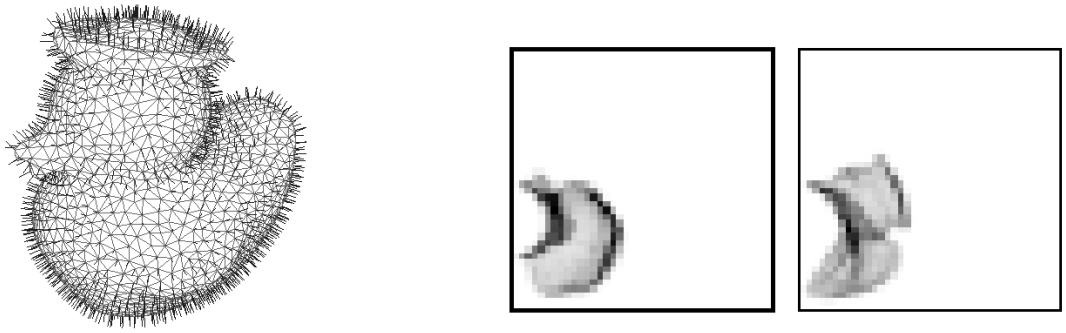

Figure 2-3: 3D points with surface normal and the spin images extracted from 3D points. [7]

Chua [10] creates a representation of the surface by associating a set of selected 3D-points with point signatures. The point signature is created in the following way. For a given 3D point p, a sphere of radius r is placed, centered at p. The intersection of the sphere with the object surface is a 3Dspace curve, C whose orientation can be defined by the "normal" vector, n1, a "reference" vector, n2, and the vector cross-product of n1 and n2. The 3D point, normal and the 3D space curve together makeup the point signature.

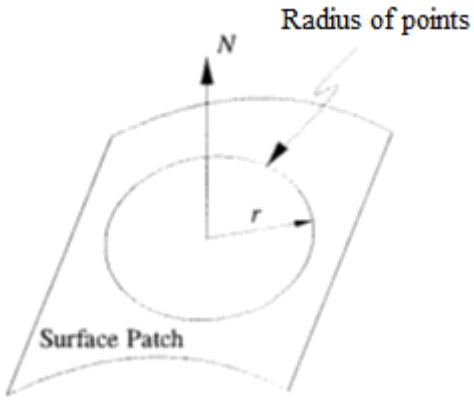

Figure 2-4: A surface showing point signature calculation. [10]

Feature based registration requires a very large number of features to be matched to extract a good transformation between the range images. Computing and matching of a large set of features is a time consuming task. The other main draw-back of feature based registration is that

it cannot handle the case when it is not possible to identify adequate number of matching features between the range images. Generally there is lot of noise in the range images caused due to lighting conditions during scanning process, so in many cases it is not possible to find adjoint features between the range images.

2.2. ICP Based Registration

An approach named "Iterative closest point (ICP)" was proposed by Besl and McKay [11]. Many of the problems of feature-based approaches were overcome by the Iterative Closest Point (ICP) approach [5]. ICP assumes a rough initial transformation between the range images, and tries to register by iteratively minimizing the alignment error between the range images. By assuming an initial transformation is known, ICP considers the registration of the range images as a numerical optimization problem. ICP based registrations have three steps.

1 Finding corresponding points: This step involves finding matching point pairs. ICP assumes that for a point P_i in the reference range image, the closest point in the other range images is a matching pair.

2 Error computation: ICP takes the sum of squared distances between the matching point-pair's to be the alignment error.

3 Error minimization: The final step of ICP is to use a numerical optimization technique to reduce the alignment error. ICP employed the least squares method to minimize the alignment error between the range images.

Searching the closest point at each iteration is a computationally expensive task. Nishino [12] and Schmitt [13] proposed to accelerate the searching (step-1), using some common search optimization techniques like kd-tree, z-buffering, or a closest-point caching.

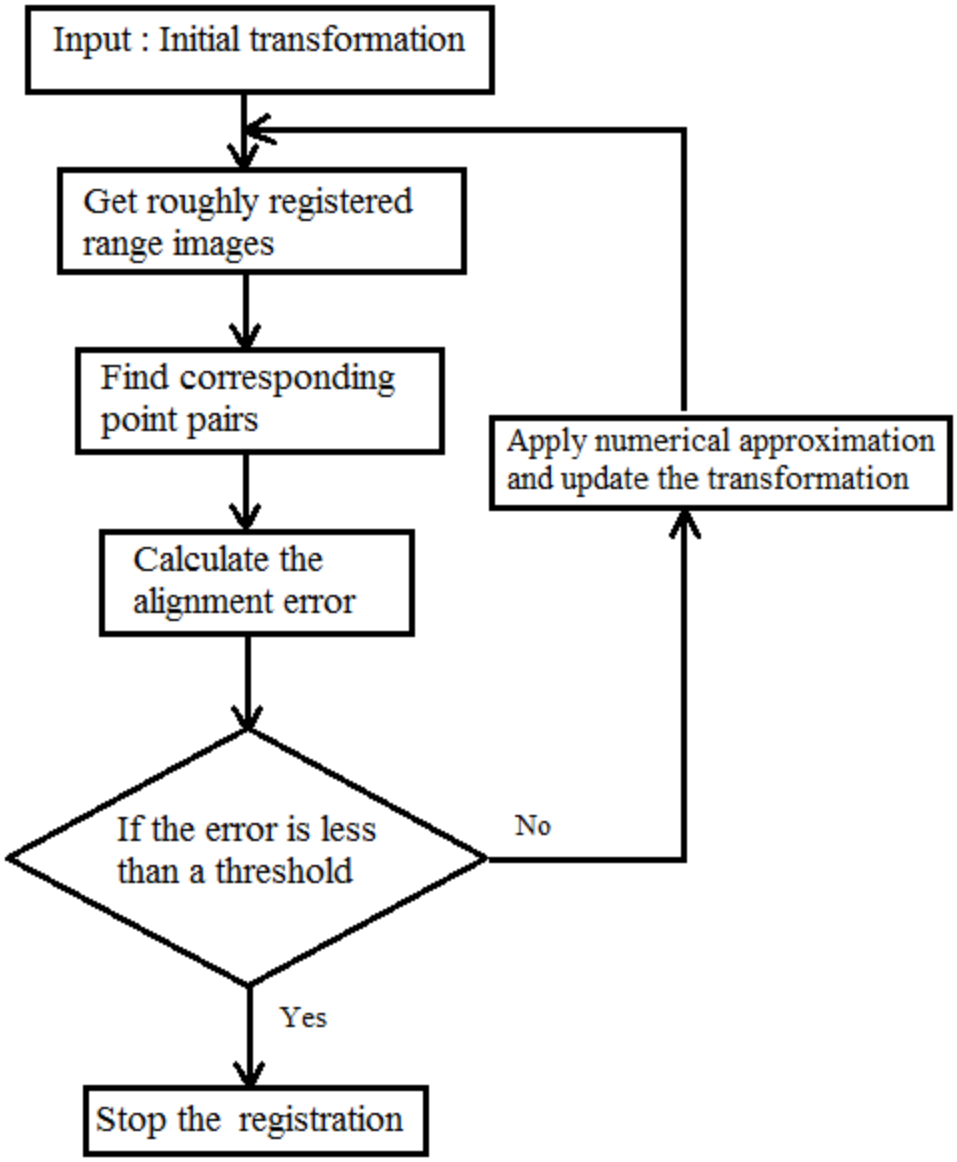

Figure 2-5: ICP algorithm shown as a flow chart.

2.2.1. Pair-wise registration (ICP variants):

Since the introduction of ICP many variations have been proposed to the basic ICP concept. These variations differ in the techniques used for finding the corresponding point pairs, error

metric, and the procedure for minimizing the error metric. Some of the important variations are discussed in this section.

Blais [14] and Neugebauer [15] proposed a fast variation to the ICP. They established point correspondences by projecting the source control point "p" (refer to Figure 2-7) onto a destination surface from the point of view of the destination "O_Q". This approach is commonly known as Point-to-Projection variation of ICP. Since it eliminates the searching process this method is fast. However, the disadvantage is that the result of registration is not as accurate as those of the others [3].

Medioni [29] proposed a more accurate variation of ICP commonly known as point-to-(tangent)-plane variation of ICP. The destination control point "q'" (refer to Figure 2-6) is the projection of the source control point "p" onto the tangent plane at a destination surface point "q"; the destination surface point "q" is the intersection of the normal vector of the source control point "p" and the destination surface. But, finding the intersection on the destination surface is also computationally expensive. Rusinkiewicz [3] accelerated searching by first searching the closest point, and finding the intersecting surface (or the triangle) from its neighboring triangles. S.Y. Park [16], combined the advantages of point-to-plane and point to-projection techniques for fast control point searching, and proposed an accurate and fast point-to-plane registration technique.

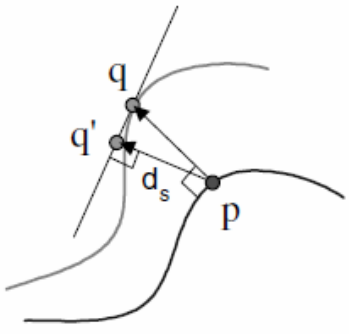

Figure 2-6: Point-to-Plane variation of ICP [16]

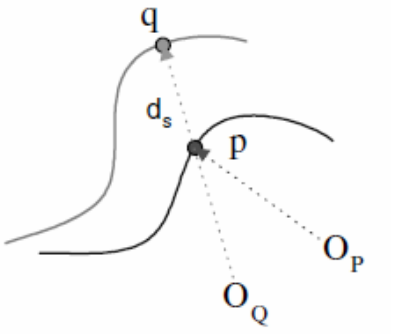

Figure 2-7: Point-to-Projection variation of ICP [16]

Holt [17] extended the basic ICP by using color information. He used color and point information to find the corresponding point pairs. Johnson [18] goes further with the color ICP approach by using the texture information. They assumed that using geometrical information of a point along with color and texture information they will be able to establish point correspondences. But, since the texture of two range images from two different views may be different due to lighting conditions, the corresponding points may have different color and texture values, which will make the final result inaccurate during registration.

Dorai [19] formulated a new error metric to handle noisy data. They proposed a new error metric by establishing dependencies between the orientation of a surface, noise in the sensed surface data, and the accuracy of surface normal estimation. Their error metric commonly known as" minimum variance estimator" can handle the noise case very well.

2.2.2. Multi-view registration (ICP variants):

The approaches in the previous section focused on registration of two data range images using ICP technique at the core. But the problem with pair-wise registration is that the small error in each registration step can accumulate and result in a larger total error [5, 6]. Multi-view

registration is the process of aligning more than two range images simultaneously. Multi-view registration reduces the accumulation of error by minimizing the error among all the range images registered simultaneously. But the problem of multi-view registration is not that widely addressed. In this section we present some of the important multi-view registration techniques.

Neugebauer [15] proposed a method for multi-view registration. They defined an error metric and a minimization technique to simultaneously register multiple data sets. But the complexity of their error metric increases the processing time.

Pulli [5] proposed an approach to register the range images using pair-wise registration first. Then use the pair-wise alignments as constraints that the multi-view step enforces. By this they evenly diffuse the pair-wise registration errors. The main advantages of their method are it has less accumulation error and it is less likely to get struck in a local minimum.

Figure 2-8: A model created using pair-wise registration showing a large final error. [5]

Figure 2-9: A model created using multi-view registration with a lesser error. [5]

Gregory [20] also proposed a similar method to register the range images using pair-wise registrations first followed by multi-view registration. They defined registration as two sub problems:

1) Local problem of pair-wise registration on neighboring views.

2) Global problem of distribution of accumulated errors.

Their main contribution was that they proposed a framework for distributing the accumulated errors. They achieved this by defining the global problem as an optimization over the graph of neighboring views registered using pair-wise registrations. They showed how the graph can be decomposed into a set of multi-view registrations such that the optimal transformation parameters for each range image can be obtained.

Bhakar et.al [26] proposed a new error metric for multi-view registration. The main advantage of their error metric was that they completely avoided the need for establishing point correspondence during the iteration's ICP. Because they avoided the point correspondences step, their method is faster than other multi-view registration techniques. They also employed Levenberg-Marquardt algorithm for the error minimization step. Since Levenberg-Marquardt algorithm is known to have better convergence properties than other algorithms, the resulting method requires less number of iterations to register the range images. This method is also known as "distance based registration".

2.3. Accelerating registration

There have been several attempts to accelerate or speed up registration using different parallel hardware like GPUs, multi-core CPUs and IBM's Cell Broadband Engine (CBE). Qiu et.al [21] proposed a GPU based "nearest neighbor search" (NNS). To evaluate the performance of their GPU based NNS they ran the error metric of the ICP algorithm on a GPU. Park et. al [22] implemented a Point-to-Plane variation of the ICP algorithm in GPU. Kitaaki et al.[23] presented a GPU implementation of a variation of ICP. They used a GPU to compute the point correspondence part of the algorithm. Park [2] presented an implementation of Point-to-

Projection ICP technique. They implemented all the computation of the registration on a GPU. S.Rusinkiewicz proposed a real-time model creation process using dual-core CPU [27].

Scharfe et. al [24] accelerated the registration process using the IBM's Cell Broadband Engine (CBE). They showed how to take advantage of the vector processors by reducing branching operations, loop unrolling and use the limited storage available in the processing cores. They achieved a speed up of 4 times by using all the six cores in the CBE.

Recently a group of researchers from Microsoft presented a fast registration of range images obtained from Microsoft Kinect. Kinect is a sensor that supplies 30 range images per second. A user can create a complete 3D model of an object by moving the Kinect slowly around the object. They used a single GPU to accelerate the Point-to-Plane variation of the ICP to register the obtained range images [25]. The important part of their ICP is choosing a good initial transformation for the range images that is close to the final transformation. Since, Microsoft Kinect supplies continuous range images at a high frequency they use the final transformation of the previous frame to be the starting transformation of the next frame. This is based on the assumption that the camera moves only a small distance between the two frames.

Table 2-1 shows an overview of the different attempts to speed up the registration process.

Work	**Hardware accelerator**	**Registration technique**	**Pair-wise/ Multi-view**	**Contribution**
Qiu [21]	GPU	ICP	Pair-wise	Registration algorithm using one GPU.
Qiu [21]	Multi-core CPU	ICP	Pair-wise	Registration algorithm using multi-core CPU.
Park [22]	GPU	Projection point variation of ICP	Multi-view	Registration algorithm using one GPU.
Kitaaki [23]	GPU	HM - ICP	Pair-wise	Registration algorithm using one GPU.
Park [22]	GPU	Projection point variation of ICP	Pair-wise	Registration algorithm using one GPU.
Scharfe [24]	IBM cell processor	Point-to-Plane variation of ICP	Pair-wise	Registration algorithm using one GPU.
Rusinkiewicz [27]	Multi-core CPU	Point-to-Plane variation of ICP	Pair-wise	Model creation process on multi-core CPU.
Microsoft KinectFusion [25]	GPU	Point-to-Plane variation of ICP	Pair-wise	Registration algorithm using one GPU.
KinFU [35]	GPU	Point-to-Plane variation of ICP	Pair-wise	Registration algorithm using one GPU.
Our work	Multipple GPUs and multiple multi-core CPU,s	Distance-field variation of ICP	Pair-wise and Multi-view	Registration algorithm using multiple GPUs and multi-core CPUs

Table 2-1: Attempts to speed up the registration process.

2.4 Focus of our research

All the previous works in accelerating registration has been concentrated in using a single GPU or a single multi-core CPU to accelerate the pair-wise registration. Our research has been driven by the two following ideas.

- Today all the modern computers have both a multi-core CPU and a GPU. Accelerating the registration process can benefit by using both the multi-core CPU and GPU. So, we have devised a functionality distribution[2] scheme to accelerate pair-wise and multi-view registration "distance based registration" algorithm using both multi-core CPUs and GPUs.

- It is clear from the literature [5, 6] that multi-view registration is more accurate than pair-wise registration. Yet, only one attempt has been made to accelerate multi-view registration and no research has been done in comparing pair-wise with multi-view registration in the context of multi-core CPU and GPU. So, we have made a thorough comparison of accelerating pair-wise registration with multi-view registration.

[2] Functionality distribution: In the context of scheduling in a heterogeneous environment, it is the process of dividing a problem in to a set of smaller problems and scheduling the sub-problem to the most suitable type of processor to achieve the best possible performance.

Chapter 3: Functionality distribution

3D registration can be viewed as a group of sub-problems that can be solved by different algorithms. To accelerate 3D registration using both multi-core CPUs and GPUs, it is important to schedule the right sub-problem to the right processor to best exploit the architecture of the processor. Functionality distribution can be used to properly schedule the sub-problems of 3D registration.

In the first part (Sections 3.1 and 3.2) of this chapter we introduce pair-wise and multi-view distance based registration techniques. In the second part (Section 3.3), we present different distribution schemes for accelerating the distance based registration technique using multi-core CPU and GPU. Finally we select the best distribution scheme.

3.1. Distance based registration

For registering the range images, the important step is to create a representation of the range images using features that are independent of orientation. The key idea is to define a "distance field" around the range images, and create a discrete representation of the range images using distance field values, defined on a 3D grid surrounding the range images. This representation of the range images can be then matched to extract the transformation parameters and is called distance based registration.

3.1.1. Distance value

To create a distance field, we consider uniform 3D grids within the bounding box enclosing the range images. For every grid point, we find the nearest 3D point in the range image, and record the distance as the distance field value of the grid point. Let's call the grid points around the reference image model-grid-points and the grid points around other range images as "data-grid-points".

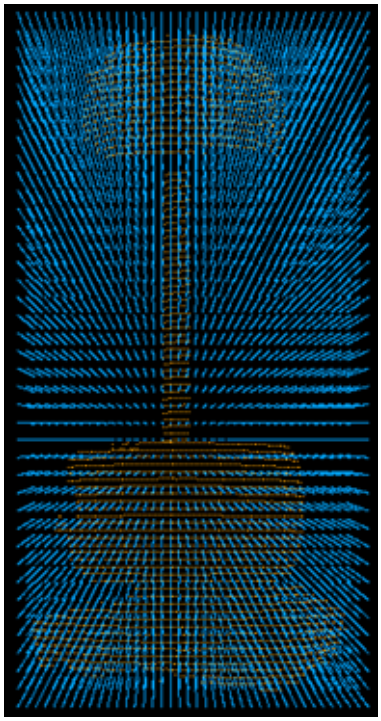

Figure 3-1: Data-points (orange) and grid-points (blue).

Even though we store the distance field values, only at the grid points, the distance field values at any other point inside the bounding box can be approximated using tri-linear interpolation. Tri-linear interpolation of distance values is discussed below.

Given 8 grid-points C_{000}, C_{001} … C_{111} and their respective distances values $V[x_0,y_0,z_0]$, $V[x_0,y_0,z_1]$ … $V[x_1,y_1,z_1]$. The distance value of any point C inside the bounds of the 8 grid-points can be interpolated using the equations below. Where (x,y,z) is the location of point C and (x_0,y_0,z_0), (x_0,y_0,z_1) … (x_1,y_1,z_1) are the locations of grid-points C_{000}, C_{001} … C_{111} respectively.

$$x_d = (x - x_0)/(x_1 - x_0) \qquad z_d = (z - z_0)/(z_1 - z_0)$$
$$y_d = (y - y_0)/(y_1 - y_0)$$

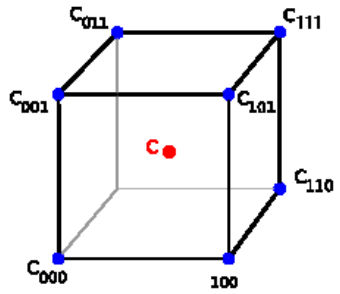

Figure 3-2: data-point "c" with 8 grid-points.

$$
\begin{aligned}
i_1 &= V[x_0, y_0, z_0](1 - z_d) + V[x_0, y_0, z_1]z_d \\
i_2 &= V[x_0, y_1, z_0](1 - z_d) + V[x_0, y_1, z_1]z_d \\
j_1 &= V[x_1, y_0, z_0](1 - z_d) + V[x_1, y_0, z_1]z_d \\
j_2 &= V[x_1, y_1, z_0](1 - z_d) + V[x_1, y_1, z_1]z_d \\
w_1 &= i_1(1 - y_d) + i_2 y_d \\
w_2 &= j_1(1 - y_d) + j_2 y_d
\end{aligned}
$$

$$IV = w_1(1 - x_d) + w_2 x_d.$$

Here, IV is the interpolated distance value for the point C shown in figure 3-2.

3.1.2. Error metric

The key assumption in error computation is if two range images are well registered, then the distance values of the data-grid-points should be same as the corresponding model-grid-point[3]. To find the corresponding model-grid-point, the data-grid-point is transformed to the reference co-ordinate system using the transformation parameters. The distance value of the transformed data-grid-point in the reference co-ordinate system can be computed by tri-linear interpolation. The difference between the distance value of data-grid-point and the transformed data-grid-point gives the alignment error at the particular grid-point. The sum of errors at all the grid points gives the alignment error between the two range images.

[3] A corresponding model-grid-point is obtained simply by transforming the data-grid-point with the transformation parameter T. At the start of registration, T is the supplied initial transformation; T is continuously updated as registration proceeds.

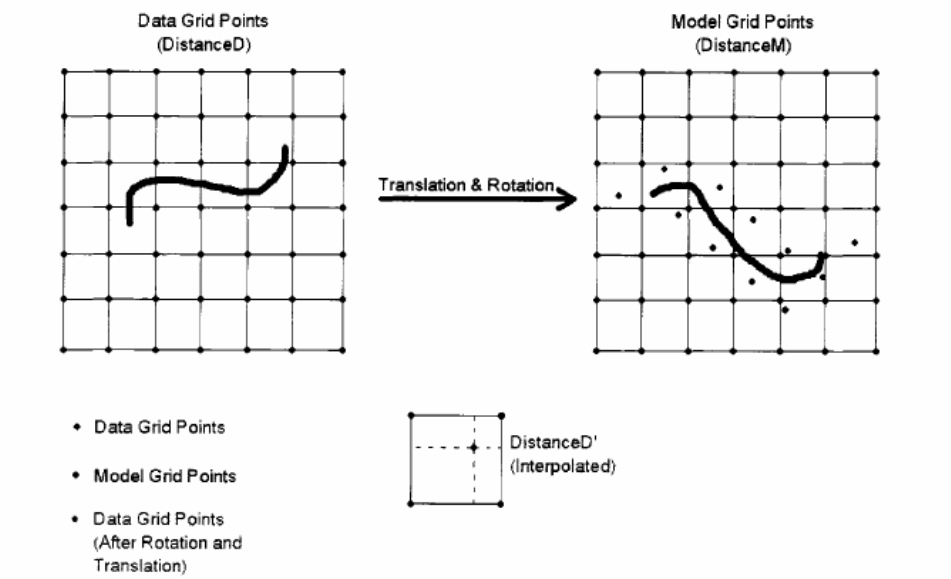

Figure 3-3: A 2D representation of error calculation.

3.1.3. Transformation

We use quaternions to represent the rotational parameters. Quaternion is a 4 dimensional vector-space, so it uses four terms: q1, q2, q3 and q4 to represent a rotation. A data-grid-point (x, y, z) can be rotated by applying the quaternion rotational matrix R.

$$R = \begin{matrix} 1 - 2*q2^2 - 2*q3^2 & 2*q1*q2 - 2*q3*q4 & 2*q1*q3 + 2*q2*q4 \\ 2*q1*q2 + 2*q3*q4 & 1 - 2*q1^2 - 2*q3^2 & 2*q2*q3 - 2*q1*q4 \\ 2*q1*q3 - 2*q2*q4 & 2*q2*q3 + 2*q1*q4 & 1 - 2*q1^2 - 2*q2^2 \end{matrix}$$

If, T is the translation vector,

$$T = \begin{matrix} T_x \\ T_y \\ T_z \end{matrix}$$

Then the transformed data-grid-point D (x', y', z') is given by

(x', y', z') = (x, y, z) * R + T

$$x' = x - 2(q_2^2 * x) - 2(q_3^2 * x) + 2(q_1 * q_2 * y) + 2(q_3 * q_4 * y) + 2(q_1 * q_3 * z) - 2(q_2 * q_4 * z) + T_x$$

$$y' = 2(q_1 * q_2 * x) - 2(q_3 * q_4 * x) + y - 2(q_1^2 * y) - 2(q_3^2 * y) + 2(q_2 * q_3 * z) + 2(q_1 * q_4 * z) + T_y$$

$$z' = 2(q_1 * q_3 * x) + 2(q_2 * q_4 * x) + 2(q_2 * q_3 * y) - 2(q_1 * q_4 * y) + z - 2(q_1^2 z) - 2(q_2^2 z) + T_z$$

3.1.4. Error function

To interpolate the distance values of transformed data-grid-points in reference co-ordinate system, we find 8 grid points surrounding the transformed data-grid-points. Not all the transformed data-grid-points will fall inside the bounds of the reference co-ordinate system. The transformed data-grid-points that fall outside the bounds of the reference co-ordinate system will be ignored during error computation. Let the distance field value at a data-grid-point (x, y, z) be D(x, y, z), and the distance field value of transformed data-grid-point (x', y', z') be D (x', y', z'). Then the alignment error E is given by

$$E = \sum_i [D(x, y, z) - D'(x', y', z')]^2 = \sum_i e_i^2$$

Where, 'i' is the total number of transformed data-grid-points that fall inside the bounds of the reference co-ordinate system.

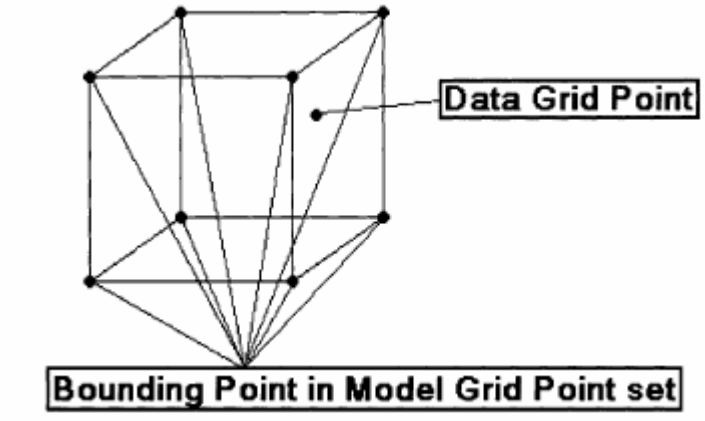

Figure 3-4: Data-grid point surrounded by the model grid-point.

3.1.5. Error minimization

Distance based registration uses the Levenberg-Marquardt algorithm [28] to minimize the sum of the squared differences in distance field values between the data-gird-point and the transformed data-grid-point. The Levenberg-Marquardt algorithm is an iterative algorithm. It minimizes the error during iteration by updating the initial transformation parameters.

The update value Δm is calculated by solving the equation:

$$\Delta m = (A + \lambda I)^{-1} B$$

Where, 'B' is the weighted gradient vector of the error function,

'A' is the Hessian matrix of the error function,

'λ' is a time-varying stabilization parameter,

And 'I' is an identity matrix.

For each data-grid-point, we calculate the hessian matrix (a_i) and the weighted gradient vector (b_i), and sum all of them together.

$A = \sum a_i$

$B = \sum b_i$

The weighted gradient vector is given by the equation:

$$b_i = -2 * e * \text{Jacobian}(e_i) = -2 * e_i * \begin{matrix} \frac{\partial e_i}{\partial q_1} \\ \frac{\partial e_i}{\partial q_2} \\ \frac{\partial e_i}{\partial q_3} \\ \frac{\partial e_i}{\partial q_4} \\ \frac{\partial e_i}{\partial T_x} \\ \frac{\partial e_i}{\partial T_y} \\ \frac{\partial e_i}{\partial T_z} \end{matrix}$$

The Hessian Matrix

$$\text{Hessian}\,(e_i) = \begin{matrix} \frac{\partial^2 e_i}{\partial q_1^2} & \frac{\partial^2 e_i}{\partial q_1\,\partial q_2} & \frac{\partial^2 e_i}{\partial q_1\,\partial q_3} & \frac{\partial^2 e_i}{\partial q_1\,\partial q_4} & \frac{\partial^2 e_i}{\partial q_1\,\partial T_x} & \frac{\partial^2 e_i}{\partial q_1\,\partial T_y} & \frac{\partial^2 e_i}{\partial q_1\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial q_2\,\partial q_1} & \frac{\partial^2 e_i}{\partial q_2^2} & \frac{\partial^2 e_i}{\partial q_2\,\partial q_3} & \frac{\partial^2 e_i}{\partial q_2\,\partial q_4} & \frac{\partial^2 e_i}{\partial q_2\,\partial T_x} & \frac{\partial^2 e_i}{\partial q_2\,\partial T_y} & \frac{\partial^2 e_i}{\partial q_2\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial q_3\,\partial q_1} & \frac{\partial^2 e_i}{\partial q_3\,\partial q_2} & \frac{\partial^2 e_i}{\partial q_3^2} & \frac{\partial^2 e_i}{\partial q_3\,\partial q_4} & \frac{\partial^2 e_i}{\partial q_3\,\partial T_x} & \frac{\partial^2 e_i}{\partial q_3\,\partial T_y} & \frac{\partial^2 e_i}{\partial q_3\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial q_4\,\partial q_1} & \frac{\partial^2 e_i}{\partial q_4\,\partial q_2} & \frac{\partial^2 e_i}{\partial q_4\,\partial q_3} & \frac{\partial^2 e_i}{\partial q_4^2} & \frac{\partial^2 e_i}{\partial q_4\,\partial T_x} & \frac{\partial^2 e_i}{\partial q_4\,\partial T_y} & \frac{\partial^2 e_i}{\partial q_4\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial T_x\,\partial q_1} & \frac{\partial^2 e_i}{\partial T_x\,\partial q_2} & \frac{\partial^2 e_i}{\partial T_x\,\partial q_3} & \frac{\partial^2 e_i}{\partial T_x\,\partial q_4} & \frac{\partial^2 e_i}{\partial T_x^2} & \frac{\partial^2 e_i}{\partial T_x\,\partial T_y} & \frac{\partial^2 e_i}{\partial T_x\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial T_y\,\partial q_1} & \frac{\partial^2 e_i}{\partial T_y\,\partial q_2} & \frac{\partial^2 e_i}{\partial T_y\,\partial q_3} & \frac{\partial^2 e_i}{\partial T_y\,\partial q_4} & \frac{\partial^2 e_i}{\partial T_y\,\partial T_x} & \frac{\partial^2 e_i}{\partial T_y^2} & \frac{\partial^2 e_i}{\partial T_y\,\partial T_z} \\ \frac{\partial^2 e_i}{\partial T_z\,\partial q_1} & \frac{\partial^2 e_i}{\partial T_z\,\partial q_2} & \frac{\partial^2 e_i}{\partial T_z\,\partial q_3} & \frac{\partial^2 e_i}{\partial T_z\,\partial q_4} & \frac{\partial^2 e_i}{\partial T_z\,\partial T_x} & \frac{\partial^2 e_i}{\partial T_z\,\partial T_y} & \frac{\partial^2 e_i}{\partial T_z^2} \end{matrix}$$

For simplifying the computation, we only compute an approximate Hessian Matrix

$$a_i = \begin{matrix} \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_2} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_3} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_4} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial T_x} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial T_y} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial T_z} \\ \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial q_2} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial q_3} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial q_4} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial T_x} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial T_y} & \frac{\partial e_i}{\partial q_2}\cdot\frac{\partial e_i}{\partial T_z} \\ \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} \\ \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} \\ \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} \\ \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} \\ \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} & \frac{\partial e_i}{\partial q_1}\cdot\frac{\partial e_i}{\partial q_1} \end{matrix}$$

We calculate all the derivatives in the following way:

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D\left(x'_i, y'_i, z'_i\right)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D\left(x'_i, y'_i, z'_i\right)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D\left(x'_i, y'_i, z'_i\right)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial e_i}{\partial q_1} = \frac{\partial x'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} + \frac{\partial y'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial y'_i} + \frac{\partial z'_i}{\partial q_1}.\frac{\partial D(x'_i, y'_i, z'_i)}{\partial z'_i}$$

$$\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} = \frac{\partial D\left(x'_i + \Delta x, y'_i, z'_i\right) - \partial D(x'_i - \Delta x, y'_i, z'_i)}{2 * \Delta x}$$

$$\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} = \frac{\partial D\left(x'_i + \Delta x, y'_i, z'_i\right) - \partial D(x'_i - \Delta x, y'_i, z'_i)}{2 * \Delta x}$$

$$\frac{\partial D(x'_i, y'_i, z'_i)}{\partial x'_i} = \frac{\partial D\left(x'_i + \Delta x, y'_i, z'_i\right) - \partial D(x'_i - \Delta x, y'_i, z'_i)}{2 * \Delta x}$$

$$\frac{\partial x'_i}{\partial q_1} = 2(q_2 * \text{y}) + 2(q_3 * \text{z})$$

$$\frac{\partial x'_i}{\partial q_2} = 2(q_1 * \text{y}) - 2(q_4 * \text{z}) - 4(q_2 * \text{x})$$

$$\frac{\partial x'_i}{\partial q_3} = 2(q_4 * \text{y}) + 2(q_1 * \text{z}) - 4(q_3 * \text{x})$$

$$\frac{\partial x'_i}{\partial q_4} = 2(q_3 * \text{y}) + 2(q_2 * \text{z})$$

$$\frac{\partial y'_i}{\partial q_1} = 2(q_2 * \text{x}) + 2(q_4 * \text{z}) - 4(q_1 * \text{y})$$

$$\frac{\partial y'_i}{\partial q_2} = 2(q_1 * \text{x}) + 2(q_3 * \text{z})$$

$$\frac{\partial y'_i}{\partial q_3} = 2(q_2 * \text{z}) - 2(q_4 * \text{x}) - 4(q_3 * \text{y})$$

$$\frac{\partial y'_i}{\partial q_4} = 2(q_1 * \text{z}) - 2(q_3 * \text{x})$$

$$\frac{\partial z'_i}{\partial q_1} = 2(q_3 * \text{x}) - 2(q_4 * \text{y}) - 4(q_1 * \text{z})$$

$$\frac{\partial z'_i}{\partial q_2} = 2(q_4 * \text{x}) + 2(q_3 * \text{y}) - 4(q_2 * \text{z})$$

$$\frac{\partial z'_i}{\partial q_3} = 2(q_1 * \text{x}) + 2(q_2 * \text{y})$$

$$\frac{\partial z'_i}{\partial q_4} = 2(q_2 * \text{x}) - 2(q_1 * \text{y})$$

$$\frac{\partial x'_i}{\partial T_x} = \frac{\partial y'_i}{\partial T_y} = \frac{\partial z'_i}{\partial T_z} = 1$$

$$\frac{\partial y'_i}{\partial T_x} = \frac{\partial z'_i}{\partial T_y} = \frac{\partial x'_i}{\partial T_z} = 0$$

$$\frac{\partial z'_i}{\partial T_x} = \frac{\partial x'_i}{\partial T_y} = \frac{\partial y'_i}{\partial T_z} = 0$$

3.1.6. Pair-wise registration algorithm

In summary, the complete distance based registration algorithm for registering two range images consists of the following steps

Step-1: Start the registration with initial transformation parameters T.

Step-2: Compute the distance value $D(x, y, z)$ for all the data-grid-points.

Step-3: Compute the alignment error

a. Transform each data-grid-point (x, y, z) using T, and find its corresponding transformed-data-grid-point (x', y', z') in the reference co-ordinate system.
b. Compute the interpolated distance value $D'(x', y', z')$, for all the transformed-data-grid-points.
c. Compute the error E (T) $=\sum_i [D(x, y, z) - D'(x', y', z')]^2$.

Step-4: Compute the update value Δm

a. Pick a modest value for $\lambda = 0.01$.
b. Solve the system of equation $\Delta m = (A + \lambda I)^{-1} B$, and compute the update value Δm.
c. Compute the error E $(T + \Delta m)$,

   If E $(T + \Delta m)$ > E (T), increase λ by a factor of 10 (or any other factor)

If E (T + Δm) < E (T), decrease λ by a factor of 10, update the transformation parameter T = T + Δm

Step-5: If the number of iterations is larger than a threshold or if the error E is smaller than a certain threshold, stop the registration process. Else go back to step 3.

For more information the reader is referred to [26].

3.1.7. Multi-view registration

Multi-view registration registers multiple range images at the same time by minimizing the alignment in the common overlapping region between all the range images. So a large common overlap is required for multi-view registration to yield good results. Ran Wang extended the error function (in section 3.1.4) for multi-view registration.

Let's call the grid-points around the reference range image as model-grid-points, and the grid-points around other range images as data-grid-points. But, this time the reverse transformation is applied to transform all the model-grid-points to different data co-ordinate systems[4], and only if the transformed model-grid-points falls inside the bounds of all the data-co-ordinate systems he computes the error. By rejecting a model-grid-point that when transformed does not fall inside the bounds of all the data co-ordinate systems, he was able to select the model-grid-points that fall in the common overlapping region of all the range images. To minimize the error he employed the Levenberg-Marquardt algorithm.

[4] Data co-ordinate systems: The co-ordinate systems of all the range images except the reference range image.

3.1.7.1. Error metric

A model-grid-point (x, y, z) can be transformed to data-coordinate system by applying the reverse transformation parameters. If, T is the translation vector,

$$T = \begin{matrix} T_x \\ T_y \\ T_z \end{matrix}$$

Then the transformed model-grid-point (x', y', z') is given by

$$\begin{matrix} x' \\ y', \\ z' \end{matrix} = [(x, y, z) - T] * R^{-1}$$

The distance value for a transformed model-grid-point in the data co-ordinate system can be computed using tri-linear interpolation. If, $D(x, y, z)$is the distance-value of a model-grid point in reference co-ordinate system and $D'(x, y, z)$ is the distance-value of model-grid point in the data co-ordinate system; then our error function for multi-view registration is defined below:

$$E = \sum_k \sum_i [D_{ik}(x, y, z) - D'_{ik}(x', y', z')]^2 = \sum_i \sum_k e_{ik}^2$$

Where, 'k' is the number of data co-ordinate systems and 'I' is the number of model-grid-points that fall inside the bounds of all the data co-ordinate systems.

Similarly, we also calculate 'k' update transformations by computing 'k' hessian matrices and 'k' weighted gradient vectors.

$$\Delta m_k = (A_k + \lambda I)^{-1} B_k$$

$$A_k = \sum a_{ik}$$

$$B_k = \sum b_{ik}$$

3.1.8. Multi-view registration algorithm

In summary, the complete distance based registration algorithm for registering multiple range images simultaneously is below

Step-1: Start the registration with initial transformation parameters T.

Step-2: Compute the distance value $D(x, y, z)$ for the grid-points in different co-ordinate systems.

Step-3: Compute the alignment error

a. Transform each model-grid-point (x, y, z) using T^{-1}, and find its corresponding transformed-model-grid-point (x', y', z') in all the data co-ordinate systems.

b. If the model-grid-point when transformed falls inside the bounds of all the data co-ordinate systems compute the interpolated distance value $D'(x', y', z')$ in different data co-ordinate systems, for the transformed-model-grid-points.

c. Compute the error $E(T) = \sum_k \sum_i [D_{ik}(x, y, z) - D'_{ik}(x', y', z')]^2$.

Step-4: Compute the update value Δm_k for each co-ordinate system

a. Pick a modest value for $\lambda = 0.01$.

b. Solve the system of equation $\Delta m_k = (A_k + \lambda I)^{-1} B_k$, and compute the update value Δm_k.

c. Compute the error $E(T^{-1} + \Delta m)$,

If $E(T^{-1} + \Delta m) > E(T^{-1})$, increase λ by a factor of 10 (or any other factor)

If $E(T^{-1} + \Delta m) < E(T^{-1})$, decrease λ by a factor of 10, update the transformation parameter $T^{-1} = T^{-1} + \Delta m$

Step-5: If the number of iterations is larger than a threshold or if the error E is smaller than a certain threshold, stop the registration process. Else go back to step 3.

3.2. Accelerating registration

The distance value computation involves searching the closest 3D points in the range image for every grid point. Generally only the grid points closer to the range image is used in the error calculation and error minimization steps. Using the grid-points closer to the range image reduces the computation in the error calculation & error minimization steps, and also improves the quality of registration by avoiding the unwanted errors introduced by considering grid-point that are far from the range image. The traditional method to choose the grid-points closer to the range image is one of the following:

1. Compute the distance for all the grid-points and choose grid-points with distance values below a certain threshold.
2. Use the texture information to identify some key feature points and select the grid-points around the feature points.

Both the above methods to choose the grid-points are not efficient. In method-1 the distance values are unnecessarily calculated for the grid-points that will not be included in the subsequent steps. Method-2 demands the use of some image processing technique to identify the key feature points. Ran Wang in [26] showed that using image processing techniques for 3D registration does not give good results because of the noise introduced by lighting. So, we introduce a new distance computation technique that includes selection of the grid-points close to the range image.

Our distance computation technique is based on the fact that, since, the grid points around the range image are uniform; given a 3D point from the range image the grid points surrounding the given point can be found without searching. We compute the distance values using the following technique.

1. For every 3D point ‘Pi’ in the range image, find the surrounding grid-points.

   If (xm, ym, zm) are the lower bound values of the grid, (xd, yd, zd) are the step sizes in x, y & z axis, and (px, py, pz) is the 3D point, then one of the surrounding grid points is given by

$$\left\{ \frac{(x-xm)}{xd} , \frac{(y-ym)}{yd} , \frac{(z-zm)}{zd} \right\}$$

   The other surrounding grid points can be found from this grid-point.

2. Compute the Euclidean distance between ‘Pi’ and the surrounding grid-points.
3. If the distance between ‘Pi’ and any of the surrounding grids point, is smaller than the smallest distance computed between that grid point and any other 3D point other than ‘Pi’, replace the previously computed smallest distance with the newly computed smallest distance.

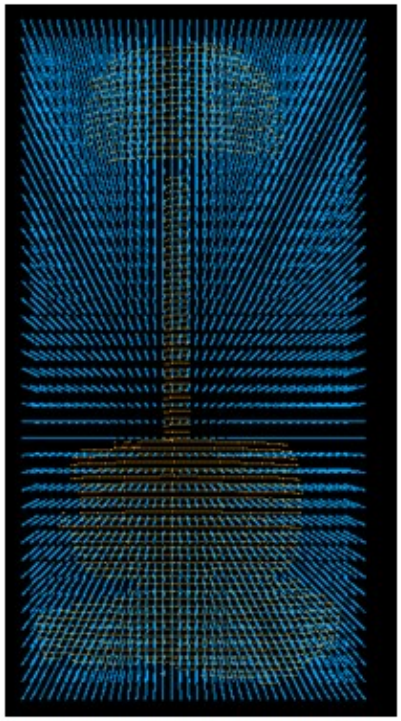

Figure 3-5: Data-points (orange) and grid-points (blue)

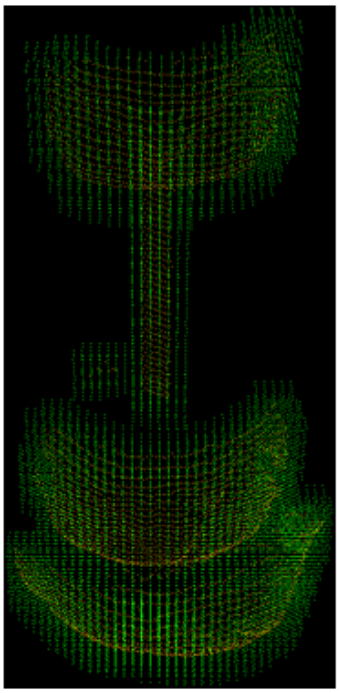

Figure 3-6: Data-points (orange) and grid-points for which distance value is computed. (Green)

Our distance computation technique selects the grid-points close to the range images without computing the distance for all the grid-points or using image processing techniques. Let's call the selected grid points (shown in figure 3-6) for which distance is computed as influenced grid points.

3.3. Distribution schemes

To formulate a distribution scheme we divide the registration process into two sub-problems

- Distance value computation (step-2 in Section 3.1.6)
- Error computation and minimization (step-3 and step-4 in Section 3.1.6)

Now we present the distribution schemes and our experiments to evaluate the different distribution schemes.

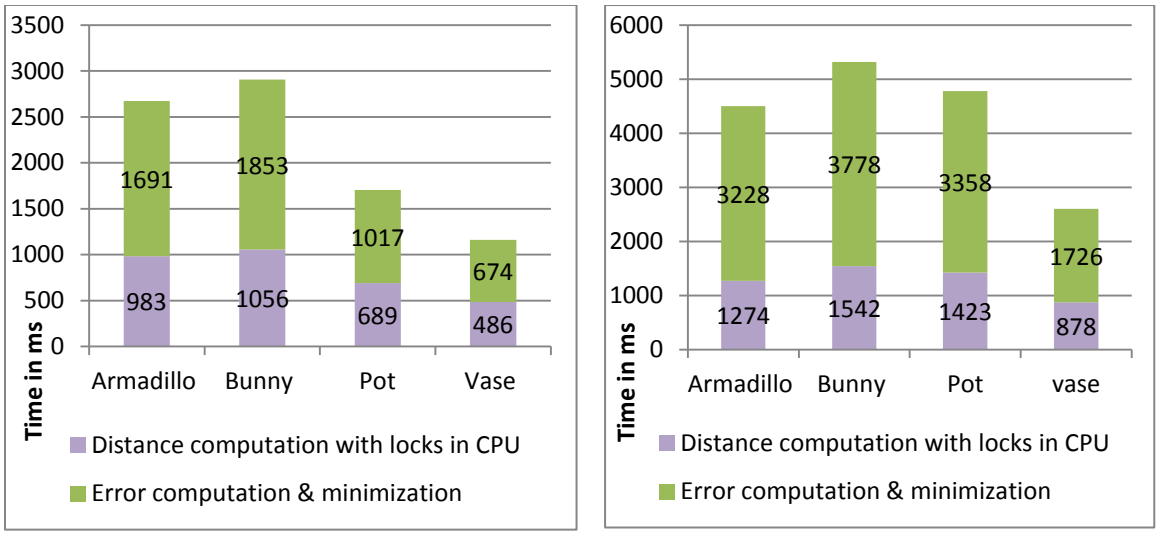

Figure 3-7: Pair-wise (left) and multi-view (right) registration time using 1 CPU core.

3.3.1. Registration using one GPU

In this section, we present our technique to run both the sub-problems on a GPU and the time taken for registering the same data-sets used in Section 3.3 using a GPU.

To run the distance computation on the GPU, we spawn a CUDA thread for each data-point, each thread computes the Euclidean distance between the data-point and its surrounding grid-points, then stores the distance for each grid-point if the current computed distance is smaller than the previously computed distance. To store the distance we allocated 1 floating-point space for every grid-point. Since, multiple threads can run concurrently on GPUs, each thread has to acquire a lock for each grid-point to store the distance.

To accelerate the error calculation step of pair-wise registration on the GPU, we spawn a CUDA thread for every influenced data-grid-point. Each thread transforms one data-grid-point to the mode co-ordinate system, interpolates the distance value of the transformed data-grid-point in the model co-ordinate system and computes the error (shown in Section 3.2).

To run the error calculation step of multi-view registration on the GPU, we spawn a CUDA thread for every influenced model-grid-point. Each thread transforms one model-grid-point to all

the data co-ordinate system. If the transformed-model-grid-point falls inside the bounds of all the data co-ordinate systems, then the thread interpolates the distance values in the data co-ordinate systems and computes the error (shown in Section 3.2).

To map the error minimization step of pair-wise registration on the GPU, we spawn a CUDA thread for every influenced data-grid-point. Each thread calculates the partial derivatives of E with respect to the transformation parameters for one influenced grid point.

To run the error minimization step of multi-view registration on the GPU, we spawn a CUDA thread for every influenced model-grid-point. Each thread calculates the N_r-1 sets of partial derivatives (one for each data-co-ordinate systems) of E with respect to the transformation parameters for one influenced model-grid-point.

The graph below shows the total time taken for registering the data-sets in Section 3.1 on Nvidia Tesla C010 GPU. The graph shows the time taken for the sub-problems and the time spent waiting because of locks.

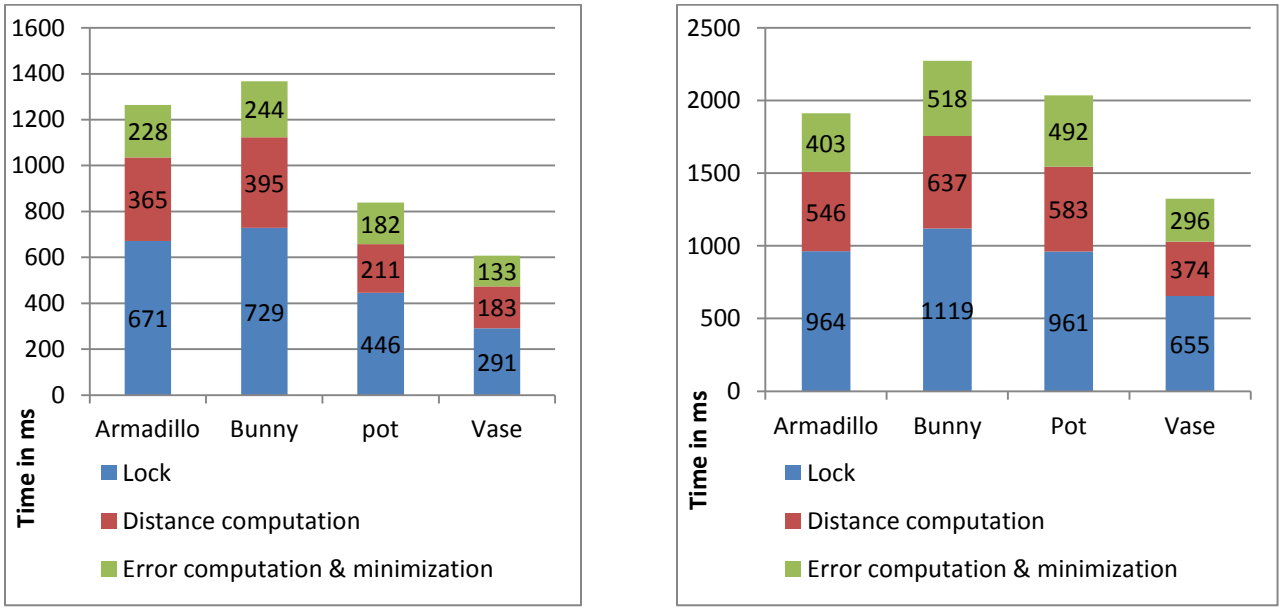

Figure 3-8: Pair-wise (left) and multi-view (right) registration time using 1 GPU.

From the previous graph, we can observe that a significant time is spent waiting for the locks. To reduce the time spent waiting for the locks we tried allocating different number of floating-point spaces and found allocating 10 spaces to give the best performance. To store the distance for a grid point, a thread will randomly acquire the lock to store it in one of the space. By this we reduced the thread's waiting time, but after the threads complete the distance computation, the smallest distance of the 10 should be found to get the distance-field-value of a grid-point.

The graph below shows the time taken for registering the same data-sets with the improved distance calculation. The graph shows the time taken for the sub-problems and the time spent waiting because of locks.

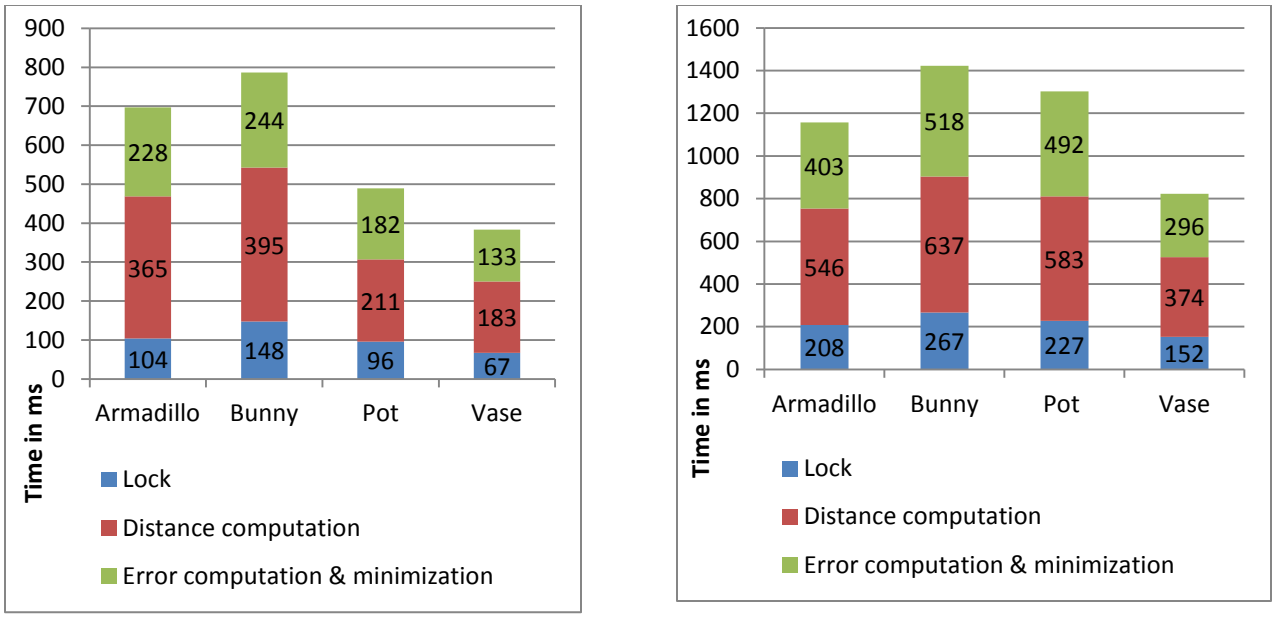

Figure 3-9: Pair-wise (left) and multi-view (right) registration time of the improved method using 1 GPU.

By comparing the graphs in figure 3-8 with the figure 3-9 we can observe, the improvement reduces the waiting time of the threads.

3.3.2. Registration using one multicore CPU

In this section, we present our technique to run both the sub-problems on the multi-core CPU and discuss the time taken for registering the same data-sets used in section 1 using the multi-core CPU.

To run the distance calculation on the multi-core CPU, we created a logical thread (using TBB) for each data-point; the logical thread computes the Euclidean distance between the data-point and its surrounding grid-points, and stores the distance for each grid-point if the current computed distance is smaller than the previously computed distance.

To run the error calculation step of pair-wise registration on the multi-core CPU, we created a logical thread (using TBB) for every influenced data-grid-point. Each thread transforms one data-grid-point to the mode co-ordinate system, interpolates the distance value of the transformed data-grid-point in the model co-ordinate system and computes the error (shown in Section 3.2).

To run the error calculation step of multi-view registration on the multi-core CPU, we created a logical thread (using TBB) for every influenced model-grid-point. Each thread transforms one model-grid-point to all the data co-ordinate system, If the transformed-model-grid-point falls inside the bounds of all the data co-ordinate systems, then the thread interpolates the distance values in the data co-ordinate systems and computes the error (shown in Section 3.2).

To run the error minimization step of pair-wise registration on the four CPU cores, we created a logical thread (using TBB) for every influenced data-grid-point. Each thread calculates the partial derivatives of E with respect to the transformation parameters for one influenced grid point.

To run the error minimization step of multi-view registration on the multi-core CPU, we created a logical thread (using TBB) for every influenced model-grid-point. Each thread calculates the N_r-1 sets of partial derivatives (one for each data-co-ordinate systems) of E with respect to the transformation parameters for one influenced model-grid-point.

To reduce the number of physical threads we grouped 1000 logical threads to one physical thread for all the sub-problems.

The graph below shows the time taken for registering the same data-sets used in the section on the multi-core Intel Xenon (E5540) CPU. The graph shows the time taken for the two sub-problems in different colors.

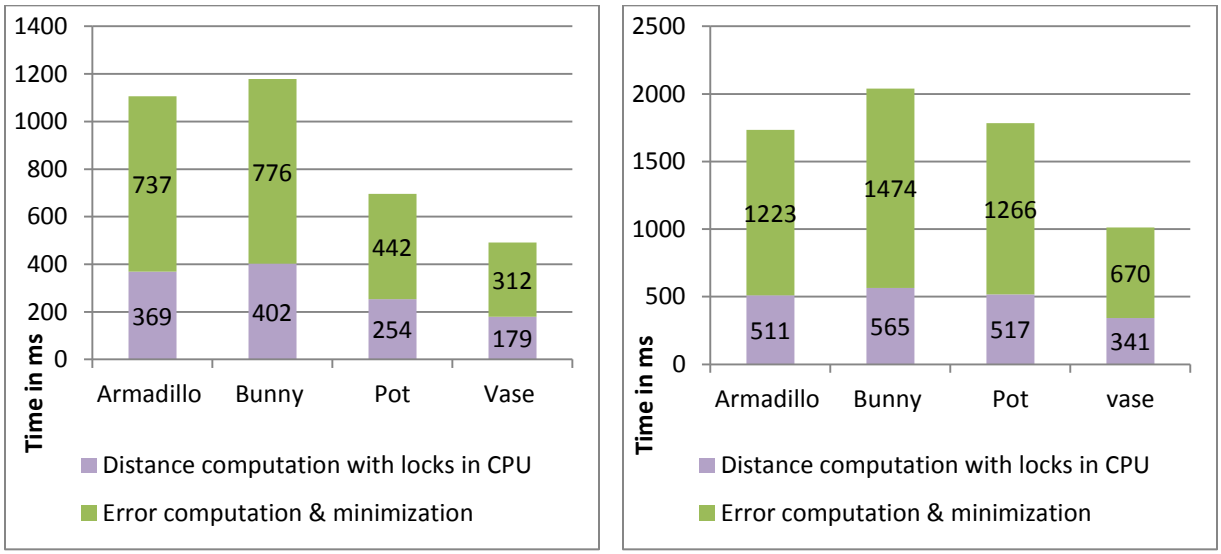

Figure 3-10: Pair-wise (left) and multi-view (right) registration time using 4 CPU core.

3.3.3. Accelerating registration using one GPU and one multicore CPU

From the experiments shown in sections 3.3.3 and 3.3.2, we can observe distance calculation is best accelerated by multi-core CPU, the error calculation and the minimization steps are best accelerated by GPU.

GPUs have 100's of light-weight cores this makes a GPU suitable for running large number (thousands or tens of thousands) of threads concurrently. But while accelerating distance computation, running large number of threads concurrently leads to thread locks and thread waits, our experiments have showed that the threads waits were a significant bottleneck to speed up the distance computation. On the other hand multi-core CPUs have fewer number (2 to 8) of more powerful cores, which makes it suitable to run fewer threads concurrently. Running fewer threads results in lesser thread waits. This makes the multi-core CPUs suitable for accelerating the distance computation part of registration.

Error computation and minimization are data-parallel in nature. As GPUs are specially designed to handle data-parallel problems, GPUs are suitable for accelerating the error computation and minimization part of registration.

The figure 3-11 shows the algorithm for registration using one GPU and one multi-core CPU.

Multi-core CPU	GPU
Step 1: Create $\mathbf{N_m + N_{p1} + N_{p2} + ... + N_{pn}}$ logical threads, each thread will compute the distance between the data-point & surrounding grid-points, and stores the distance for each grid-point if the current computed value is smaller than the previously computed value.	Step 2: Create $\mathbf{N_m}$ threads on the GPU; each thread projects one model grid-point on to the data field and computes the alignment error for one grid-point. Step 3: Create $\mathbf{N_m}$ threads on the GPU; each thread calculates the partial derivatives of E with respect to the transformation parameters for one influenced grid-point.

Figure 3-11: Hybrid distribution scheme using GPU and multi-core CPU.

The graph below shows the time taken for registering the same data-sets used in the Section 3.3.1 using the Intel Xenon (E5540) CPU and Nvidia Tesla C010 GPU. The graph shows the time taken for the two sub-problems in different colors.

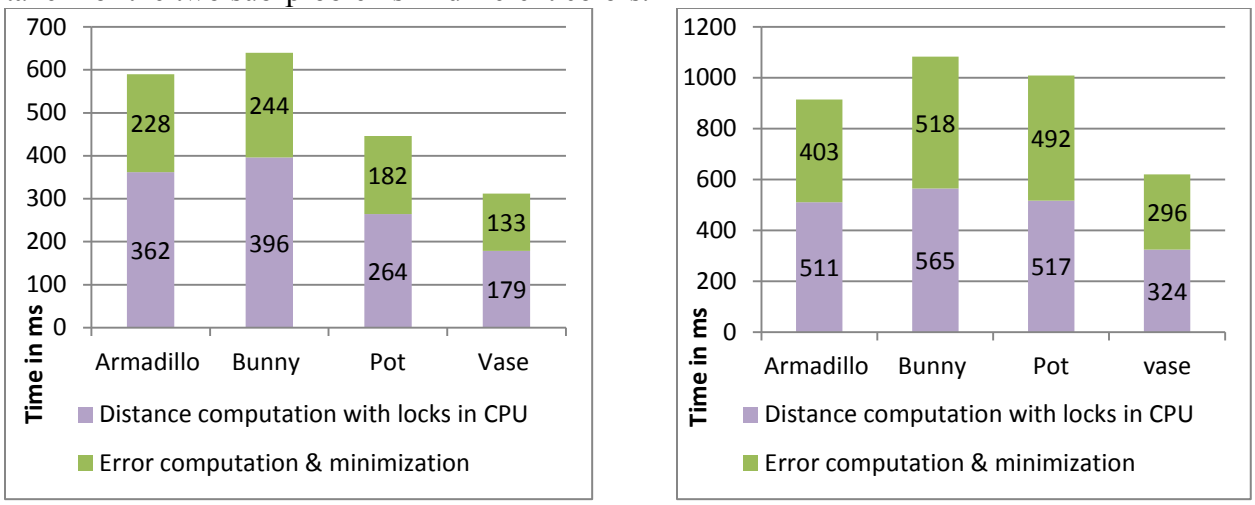

Figure 3-12: Pair-wise (left) and multi-view (right) registration time using 4 CPU cores and 1 GPU.

3.4. Comparison of the distribution schemes

The graph below presents a comparison of the speed-up achieved by the different distribution schemes. The speed up was measured by comparing with the baseline performance obtained on a single core CPU (discussed in Section-3.3.1). From the graphs we can observe that the hybrid multi-core CPU and GPU functionality distribution scheme (discussed in Section-3.3.4) gives the highest speed-up.

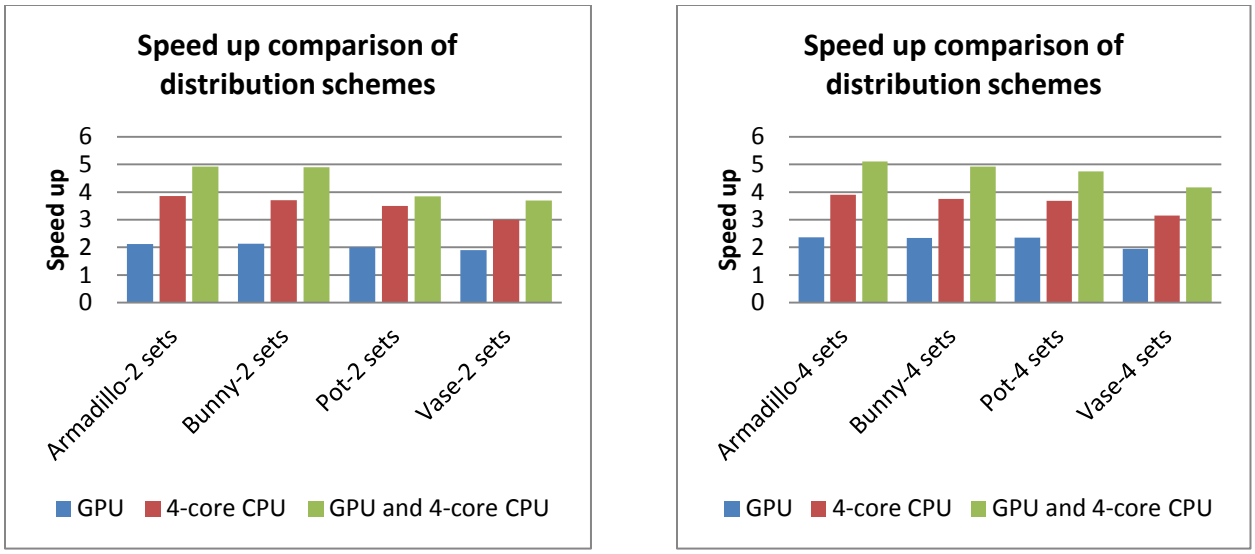

Figure 3-13: Comparison of speed-up of the distribution schemes. Pair-wise is shown in left and multi-view registration is shown in right.

3.5. Conclusion

In this chapter, we devised a functionality distribution scheme (discussed in Section 3.3.4) for a single computer with one multi-core CPU and one GPU to accelerate the distance based registration. To devise the proper functionality distribution, we experimentally compared three methods to accelerate distance based registration. Experiments conducted using four different data sets (two obtained from Stanford and two scanned in our lab) have shown 3.6 times speed-up for pair-wise registration and 4.2 times speed-up for multi-view registration is possible with the hybrid functionality distribution. While a speed up of only 1.9 for pair-wise registration and 2.3 for multi-view registration is possible with 1 GPU.

Chapter 4: Accelerating registration using a multi-core CPU and GPU cluster.

In the previous chapter we presented our functionality distribution scheme to accelerate registration. In this chapter, we present our method to further accelerate the registration process using a cluster of GPUs and multi-core CPUs. Distributing the registration process across multiple GPUs and multiple multi-core CPU cluster can be split into two steps:

1. Dividing the registration process between a GPU and multi-core CPU in a single node: Since the GPU and multi-core CPU based acceleration discussed in the previous chapter (section-3.3.3.) gave us the best results, we adopted the same method to divide the registration process between the GPU and multi-core CPU in a single node.
2. Dividing the registration process between the nodes in the cluster: We developed two methods for dividing the registration process between the nodes. In both the methods, one node is used as a master-node for co-ordination and the slave-nodes does the actual computation. The methods are described in Sections 4.2 and 4.3.

Figure 4-1 shows the architecture of our cluster. While nodes with more than one GPUs are possible our experiments were limited by the hardware set-up in our lab. However, we do not see any major obstacle in extending our algorithms to clusters with multi-GPU nodes.

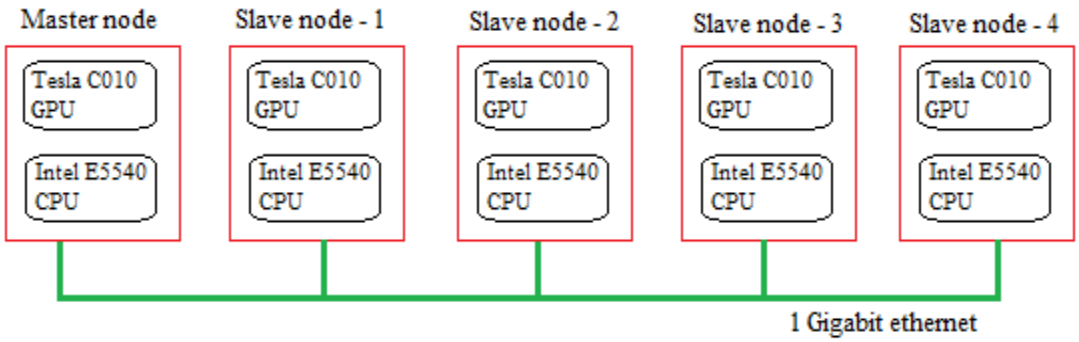

Figure 4-1: Architecture of our cluster.

4.1. Selection of grid-points

To improve the performance of our algorithm, instead of using all the grid-points for error computation and minimizing the error, we select a random subset of the grid-points and carry out error computation and minimization. Our experiments showed that a subset of approximately 40% of the grid-points are required to achieve good results.

Since only a subset of grid-points will be used for error minimization and computation, it will be a waste to compute the distance-values for all the grid points. So we delay the distance computation until we know all the grid-points that will participate in error computation and minimization. This is done as shown below.

1. For every model-point 'Pi' in the range image, find and record the surrounding grid-points as discussed in the previous chapter (Section-3.2).
2. Randomly select 40% of the recorded model-grid-points and compute the distance values.
3. Transform the model-grid-point to data co-ordinate systems and find the data-grid-points surrounding the transformed model-grid-points.
4. Compute the distance values for the data-grid points, as discussed in the previous chapter (Section-3.1.1).

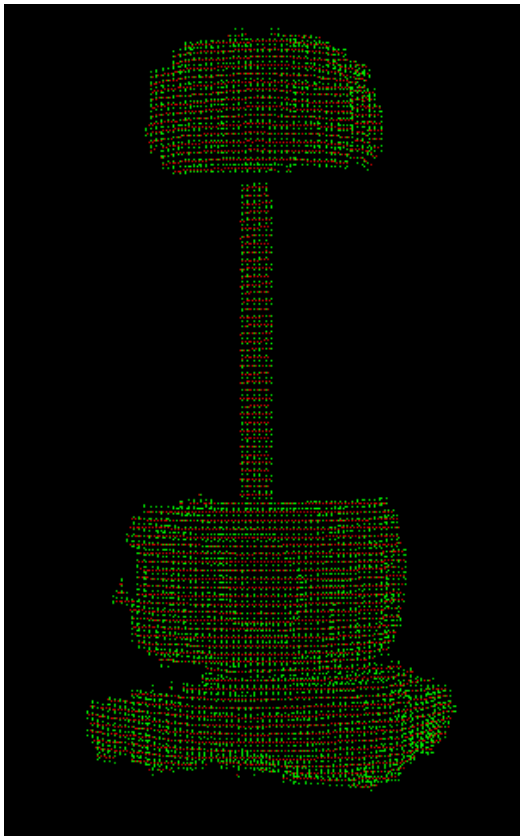

Figure 4 -2: Model-points (shown in orange color) and all the surrounding model-grid-points (shown in green)

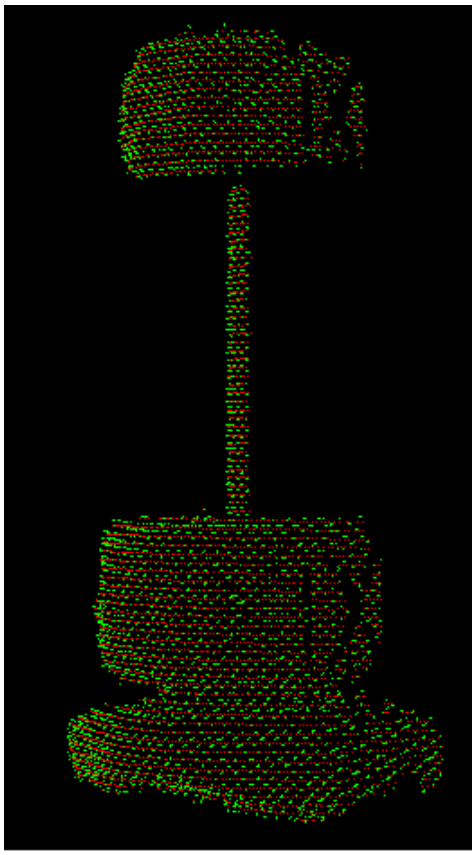

Figure 4-3: Model-points (shown in orange color) and 40% of the surrounding model-grid-points (shown in green)

4.2. Spatial partitioning

In this method to divide the registration process between the nodes, the overlapping-region[5] between the range images is divided in to 'K' tiles and each node is assigned K/N tiles, where N is the total number of slave-nodes. Firstly each slave-node computes the distance values of the grid-points inside the allocated tiles. Secondly, each slave-node computes the error and derivatives within the bounds of the allocated tiles and sends it to the master-node during each iteration. The master-node will compute the total error, and if the end condition is not met, the

[5] Overlapping-region can be computed from the bounding boxes of the two range images.

master will compute the new transformation parameters and send it to the slave-nodes. The algorithm is presented below.

Master node	Slave node
Step 1: Divide the overlapping region in to 'K' tiles. Broadcast the initial transformation parameters and send the bounds of the tiles assigned to each slave.	
	Step 2: For each model point, find the surrounding model-grid-points. Record the Model-grid-points if it falls inside the assigned tile. Select a random 40% of the recorded points from chunk 'I' where 'I' is the ID of the slave node and compute the distance values.
	Step 3: Project the selected model-grid-points to the different data co-ordinate system, and find the data-grid-points close to the projected model-grid-points.
	Step 4: Compute the distance for the model and data grid-points.
	Step 5: Compute the error for the selected grid-points within the tile and send the sum to the master-node.
Step 6: Compute the total average error by summing up all the errors received from the slave-nodes. Broadcasts the total average error to the slaves.	
	Step 7: Compute the partial derivatives for the selected grid-points and send the sum to the master.
Step 8: Compute the new transformation and broadcast it to the slaves.	
Step 9: Repeat steps 5 to 8 till the end condition a satisfied.	

Figure 4-4: Spatial partitioning algorithm to distribute the registration process.

The graphs in Figure 4-5 and Figure 4-6 show the time taken for registration on a cluster of five nodes (one master and four slave node) with 16 Intel Xenon E5540 CPU cores and 4 Tesla C010 GPU.

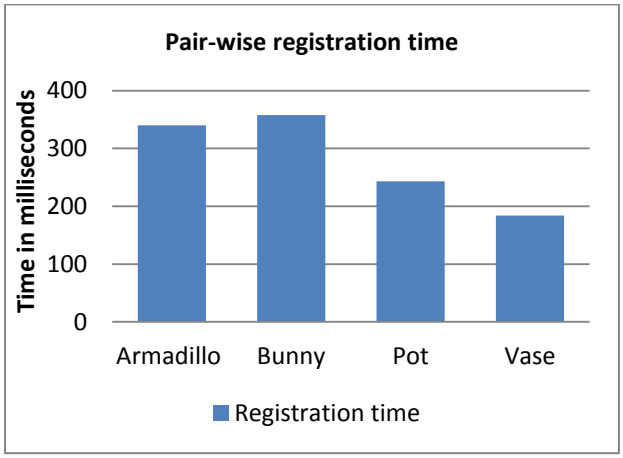

Figure 4-5: Time taken for pair-wise registration accelerated on our cluster using spatial partitioning

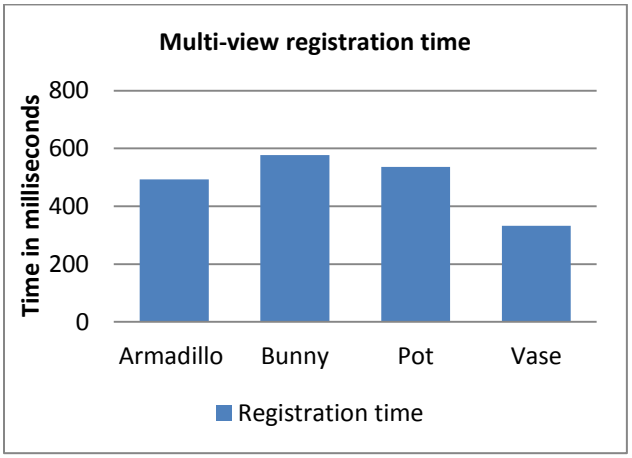

Figure 4 -6: Time taken for multi-view registration accelerated on our cluster using spatial partitioning

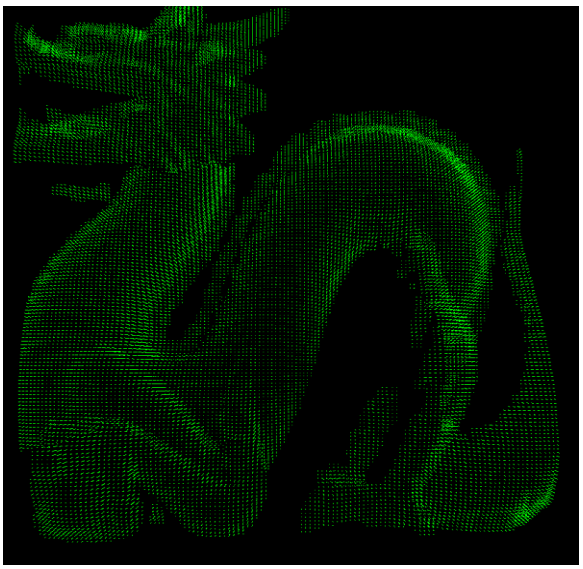

Figure 4-7: Model grid-points close to model-points

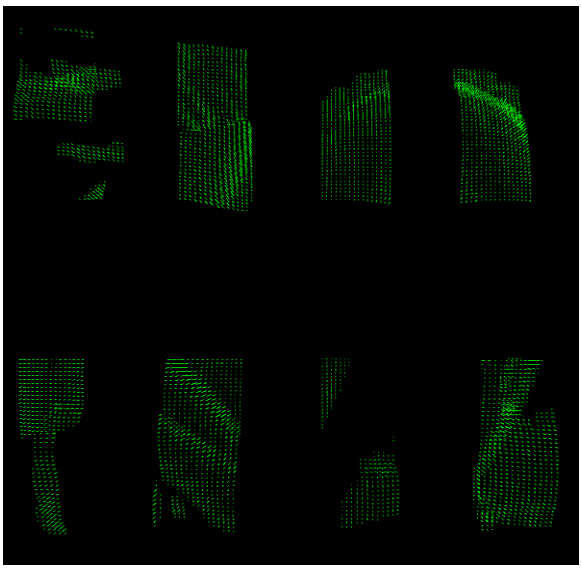

Figure 4-8: Model-grid-points chosen by Slave-1

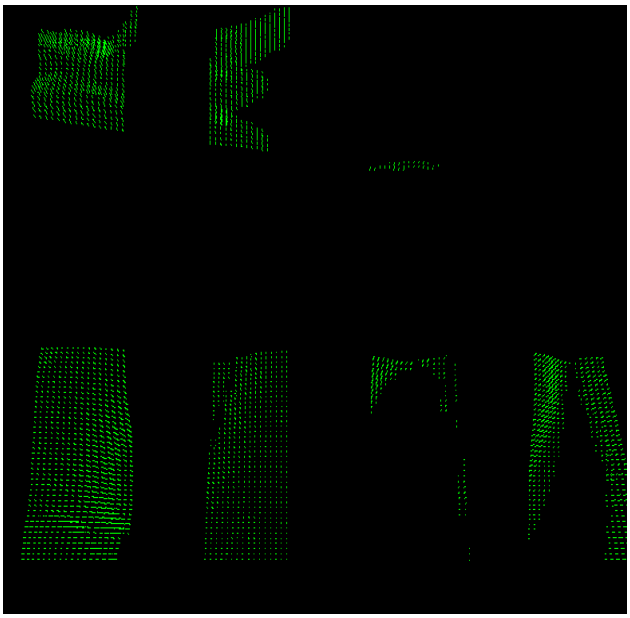

Figure 4-9: Model-grid-points chosen by Slave-2

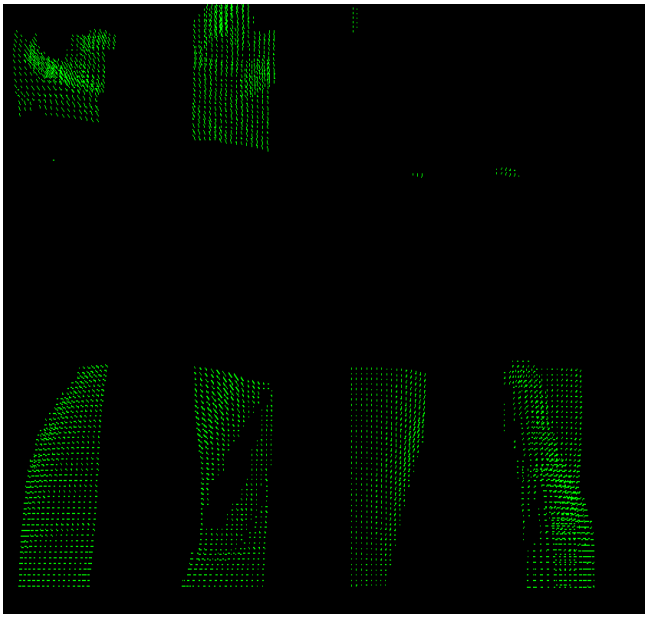

Figure 4-10: Model-grid-points chosen by Slave-3

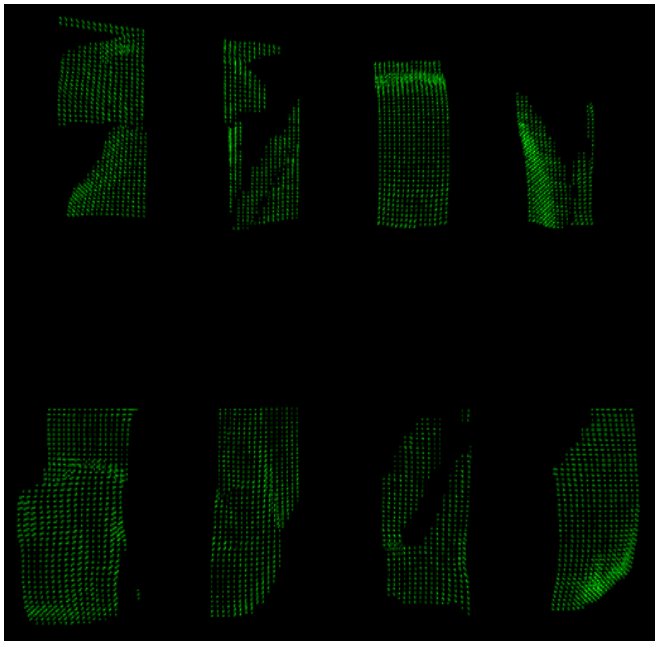

Figure 4-11: Model-grid-points chosen by Slave-4

4.3. Influence based partitioning

In this method, we redundantly compute the influenced data-grid points in all the slave-nodes. The influenced data-grid-points can be computed as discussed in 3.3.1. Then each node selects a non-overlapping sub-set of the influenced model grid points and computes the distance values, error, partial derivatives, and sends it to the master-node. The master-node computes the total error, and if iteration is needed computes the new transformation parameters and broadcasts it to the slaves. The algorithm is presented below.

Master node	Slave node
Step 1: Broadcast the initial transformation parameters to all the slaves.	
	Step 2: For each model point, find and record the surrounding model-grid-points. Divide the recorded model-grid-points in to 'N' chunks where N is the number of slave nodes. Select a random 40% of the recorded points from chunk 'I' where I is the ID of the slave node and compute the distance values.
	Step 3: Project the selected model-grid-points to the different data co-ordinate system, and find the data-grid-points close to the projected model-grid-points.
	Step 4: Compute the distance for the model and data grid-points.
	Step 5: Compute the error for the selected grid-points and send the sum to the master-node.
Step 6: Compute the total average error by summing up all the errors received from the slave-nodes. Broadcasts the total average error to the slaves.	

	Step 7: Compute the partial derivatives for the selected grid-points and send the sum to the master.
Step 8: Compute the new transformation and broadcast it to the slaves.	
Step 9: Repeat steps 5 to 8 till the end condition a satisfied.	

Figure 4-12: Influence based partitioning algorithm to distribute the registration process.

The graphs in Figure 4-13 and 4-14 show the time taken for registration on a cluster of five nodes (one master and four slave node) with 16 Intel Xenon E5540 CPU cores and 4 Tesla C010 GPU.

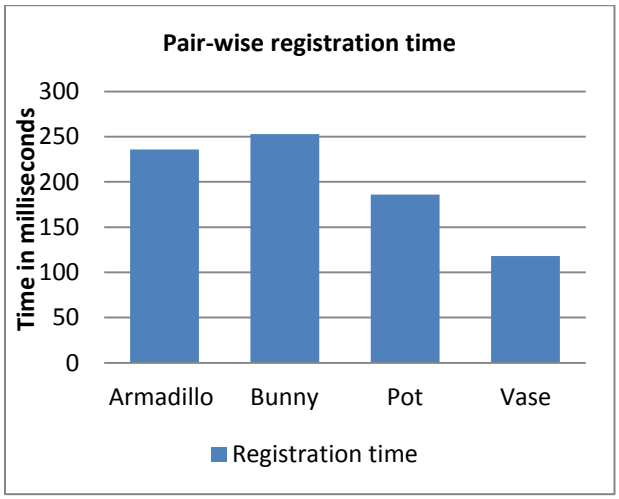

Figure 4-13: Time taken for pair-wise registration accelerated on our cluster using influence based partitioning.

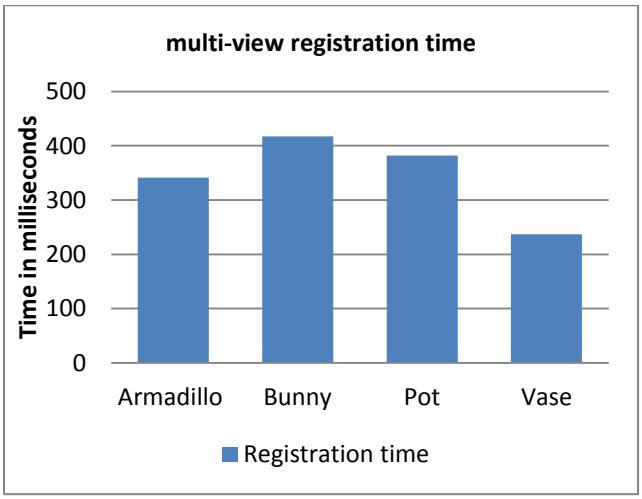

Figure 4-14: Time taken for multi-view registration accelerated on our cluster using influence based partitioning.

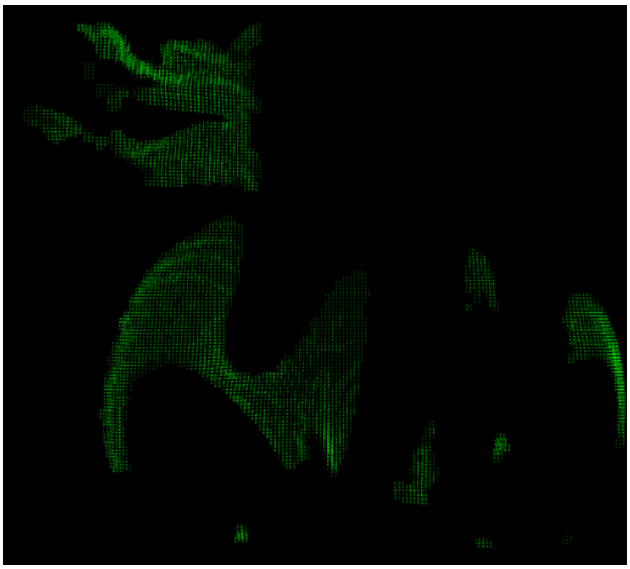

Figure 4-15: Model-grid-points chosen by Slave-1

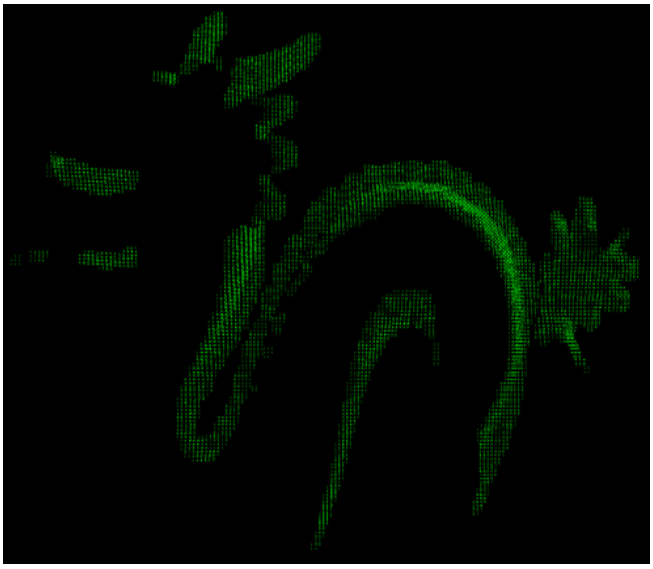

Figure 4-176: Model-grid-points chosen by Slave-2

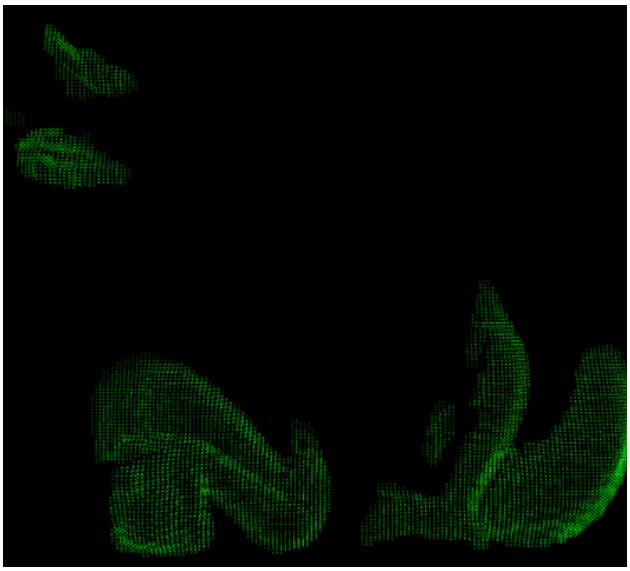

Figure 4-167: Model-grid-points chosen by Slave-3

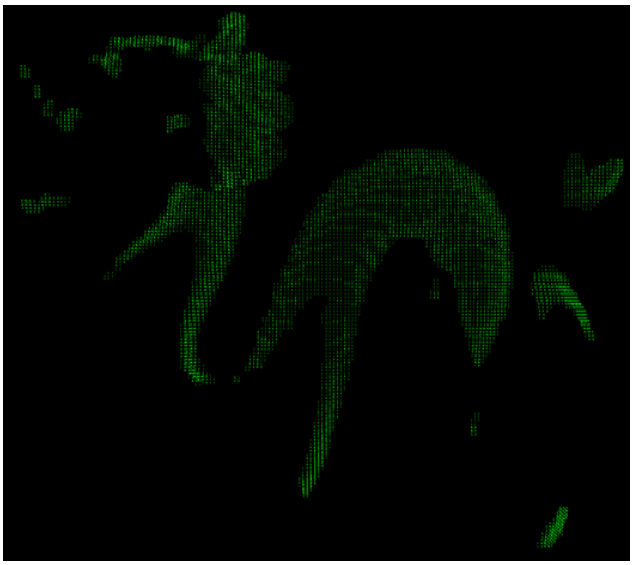

Figure 4-18: Model-grid-points chosen by Slave-4

The graphs in the figure 4-19 and figure 4-20 compare the speed-up achieved for both the methods. The speed-up is measured by comparing the time taken for registration on the cluster with the base-line time taken using a single CPU core (presented in section 3.3.1).

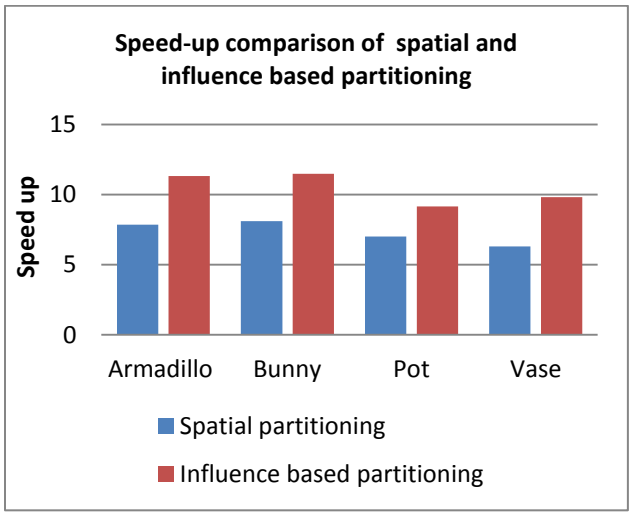

Figure 4-19: Speed-up comparison of spatial and influence based partitioning for pair-wise registration

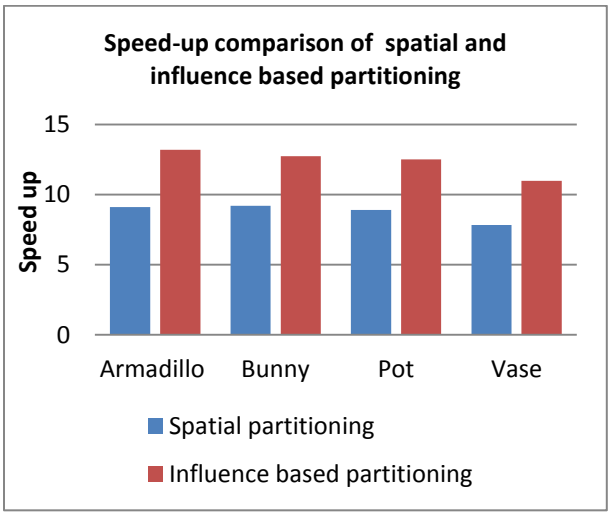

Figure 4 -20: Speed-up comparison of spatial and influence based partitioning for multi-view registration

From the graph we can observe that influence based partitioning of the registration process gives better speed than spatial partitioning of the registration process. So, we conclude that influence based partitioning is more efficient than spatial partitioning.

4.4. Conclusion

In this chapter, we extended the single multi-core CPU and single GPU based functionality distribution scheme for accelerating the distance based registration to a cluster of multiple multi-core CPUs and multiple GPUs. We presented two methods to distribute the registration process on the cluster: 1) based on spatial partitioning and 2) based on partitioning the influenced grid points. Our experiments showed 9.8 times to 13 times speed-up are possible for influence based partitioning of registration, while only 6 times to 10 times speed-up are possible spatial based partitioning of registration.

Chapter 5: Experimental comparison of pair-wise, multi-view, and KinFu registrations.

In this chapter, we present the following experiments:

- Study the scalability of our influence based partitioning algorithm (discussed in section 4.3) for accelerating the registration process using a cluster of multi-core CPUs and GPUs.
- Compare the efficiency and speed-up of pair-wise registration with multi-view registration.
- Compare the accuracy and time taken for registration, for multi-view registration with KinFu (an open-source pair-wise registration system) [35].

5.1. Scalability test

In parallel computing there are two types of scalability: 1) Scalability with increase in problem size 2) Scalability with increase in number of processors. To evaluate the scalability of influence based partitioning algorithm with increase in problem size, we registered the range images from the model "pot" using different combinations of multi-core CPUs and GPUs for different problem sizes. In order to vary the problem size we varied the number of selected grid points. Tables-5.1 and 5.2 show the time taken to register 2 and 4 range images for different problem sizes using different CPU, GPU combinations.

.

	40 % grid points	75% grid points	100 % grid points
1 CPU core	1714 ms	2564 ms	3286 ms
4 CPU – core's & 1-GPU	446 ms	694 ms	877 ms
8 CPU – core's & 2-GPUs	291 ms	439 ms	537 ms
12 CPU – core' & 3- GPUs	228 ms	327 ms	424 ms
16 CPU – core's & 4- GPUs	188 ms	273 ms	348 ms

Table 5-1: Time taken to register 2 range images for different problem sizes.

	40 % grid points	75% grid points	100 % grid points
1 CPU core	4796 ms	7233 ms	8508 ms
4 CPU – core's & 1-GPU	1120 ms	1658 ms	1927 ms
8 CPU – core's & 2-GPUs	656 ms	941 ms	1081 ms
12 CPU – core' & 3-GPUs	502 ms	716 ms	816 ms
16 CPU – core's & 4- GPUs	421 ms	627 ms	714 ms

Table 5-2: Time taken to register 4 range images for different problem sizes.

The efficiency of each test can be calculated using the standard formula.

Efficiency = Ts / PTp

Where Ts is time taken for registration using 1 CPU-core, P is the number of processors used to speed up registration and Tp is the time taken for registration using P processors. The graph in Figure 5-1 shows the efficiency of pair-wise and multi-view registrations.

From the graphs in Figure 5-1 we can observe that, as the problem size increases efficiency achieved is stable. This shows that our algorithm to accelerate the registration process using a cluster of GPUs and multi-core CPUs is scalable with increase in problem size. From the graphs we can also observe that as the number of GPUs and CPU-core's used increases the efficiency decreases. This is in agreement with Amdahl's law[6].

[6] Amdahl's law states that if P is the proportion of a program that can be made parallel, and $(1 - P)$ is the proportion that cannot be parallelized, then the maximum speedup that can be achieved by using N-processors is $S(N) = \frac{1}{(1-P) + \frac{P}{N}}$.

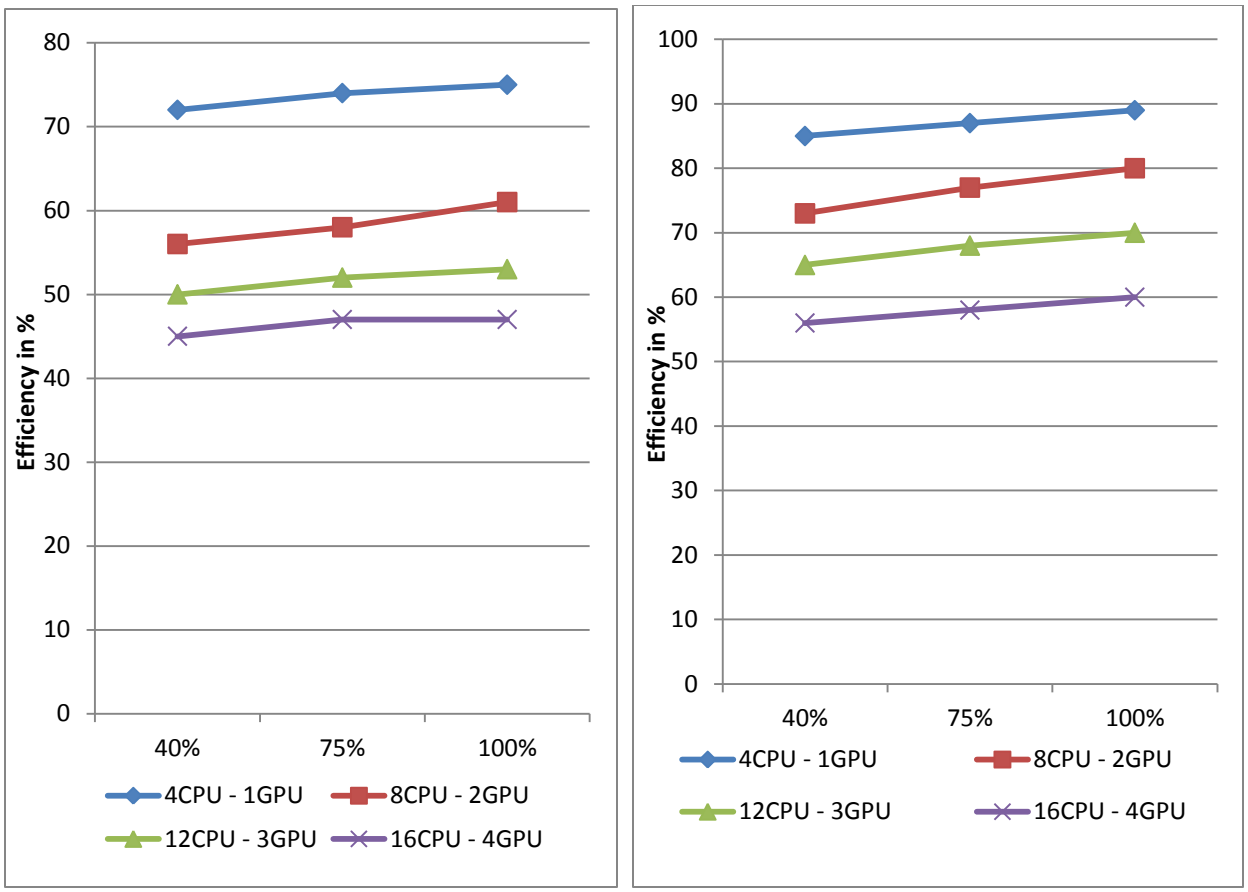

Figure 5-1: Efficiency of pair-wise registration (left) and multi-view registration (right).

To evaluate the scalability of influence based partitioning algorithm with increase in number of processors, we registered 2, 3, and 4 range images from models “pot” and “vase” using different CPU and GPU combinations. The graph in figures 5-2 shows the speed-up achieved for pair-wise and multi-view registrations. The speed-up is calculated by comparing the registration time to the base-line registration time using 1 CPU core (presented in Section 3.3.1).

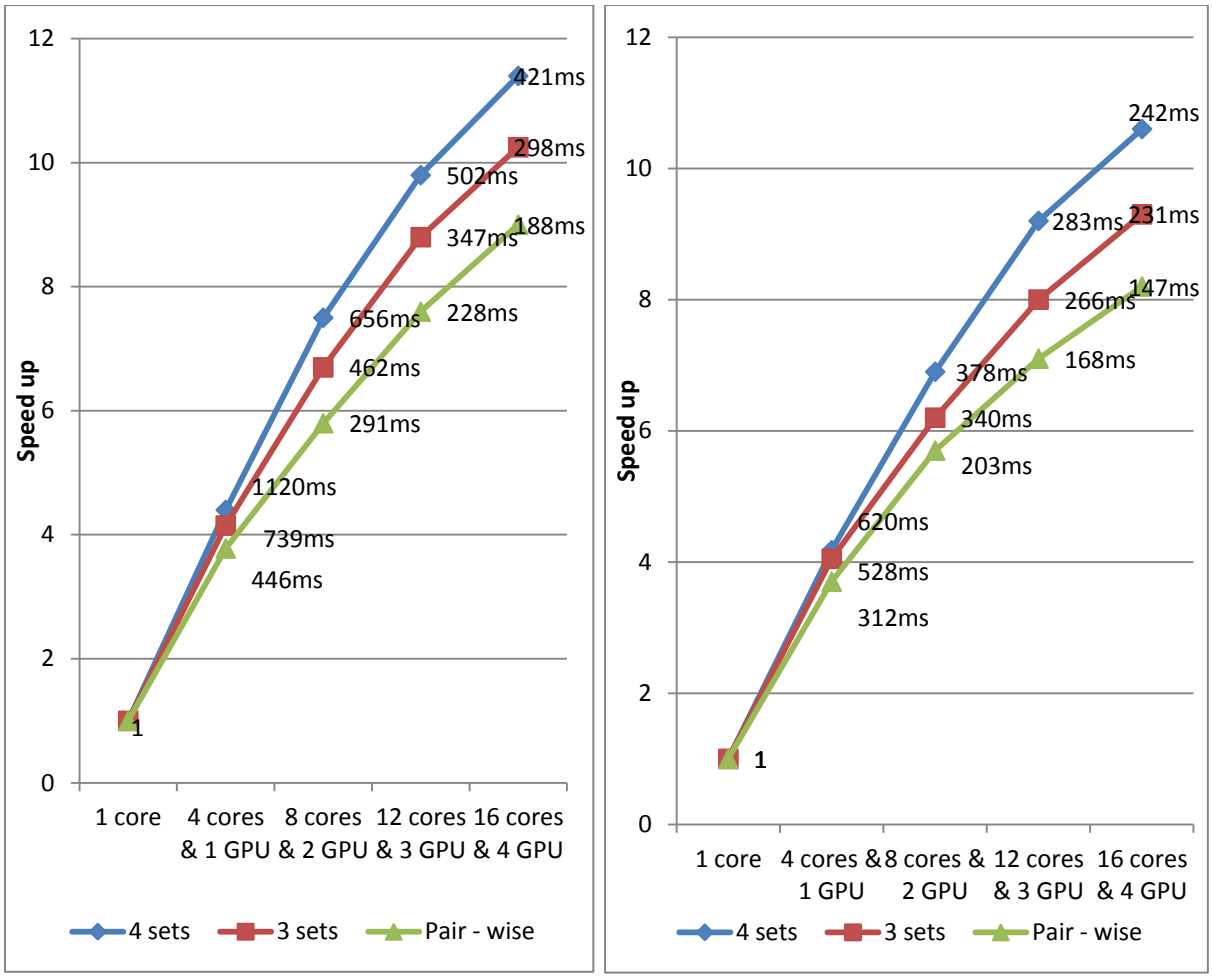

Figure 5-2: Speed-up achieved for pair-wise and multi-view registrations. Model pot (left) and vase (right).

From the graphs in Figure 5-2 we can observe that, as the number of processors increases the speed-up achieved also increases. This shows that our algorithm to accelerate the registration process using a cluster of GPUs and multi-core CPUs is scalable with increase in number of processors used.

5.2. Efficiency of pair-wise and multi-view registration

From Figure 5-2 we can also observe that multi-view registration consistently achieves better speed-up than pair-wise registration. This shows that multi-view registration is more efficient than pair-wise registration in a CPU, GPU cluster. To make a proper comparison of the

efficiency of pair-wise registration with multi-view registration we registered 4 range images using both pair-wise and multi-view using different CPU, GPU combinations. For pair-wise registration, the 4 range images were registered in 3 registration steps as shown in the Figure 5-4. For multi-view registration, the 4 range images were registered simultaneously in one step. The graph in Figure 5-4 compares the efficiency of pair-wise registration with multi-view registration. From the graph we can observe that multi-view registration is more efficient than pair-wise. There are two reasons for this.

Firstly, Multi-view registration registers all the 4 range-images simultaneously, while on the other hand, pair-wise registration requires 3 registration steps to register 4 range images. As distributing each registration step on the cluster has an overhead, pair-wise registration has more overhead than multi-view registration.

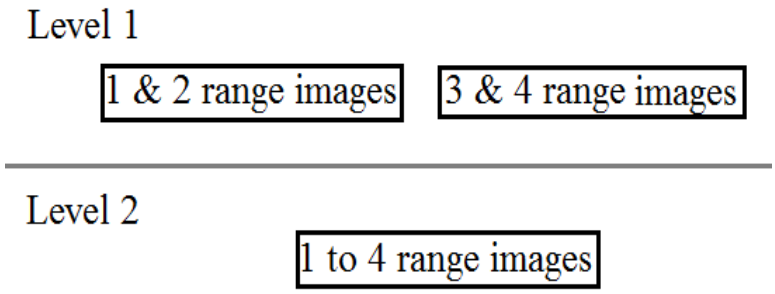

Figure 5-3: Three pair-wise registrations to register 4 range images.

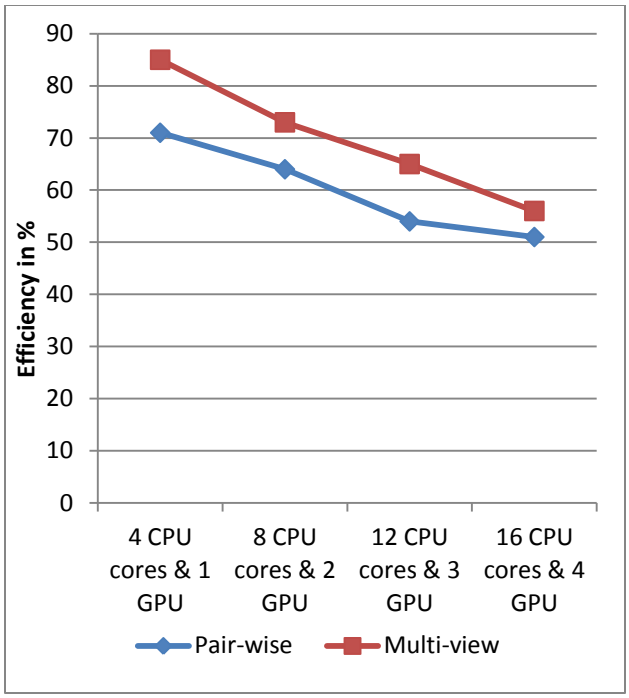

Figure 5-4: Comparison of the efficiency of pair-wise registration with multi-view registration.

Secondly, multi-view registration has a higher GPU utilization than pair-wise registration. Table 5-3 compares the GPU utilization parameters measured using the Nvidia CUDA toolkit during pair-wise and multi-view registrations. Higher GPU utilization of multi-view registration is because the GPU kernel of multi-view registration does more work than the GPU kernels of pair-wise registration. This enables CUDA to hide the memory latencies during multi-view registration more efficiently than pair-wise registration. Figure 5-5 shows the pseudo-code of pair-wise and multi-view GPU kernels.

GPU occupancy per multiprocessor	Multi-view registration	Pair-wise registration
Occupancy	83 %	67 %
Active threads	1280	1024
Active warps	40	32
Active thread blocks	5	4

Table 5-3: GPU utilization parameters for multi-view and pair-wise registration.

The less overhead and higher utilization of the GPU makes multi-view registration more efficient than pair-wise registration for accelerating the registration process in a CPU – GPU cluster environment.

Pair-wise GPU kernel	Multi-view GPU kernel
Error calculation kernel	Error calculation kernel
Step 1: Read a data grid-point.	Step 1: Read a model grid-point.
Step 2: Transform the data grid-point to model co-ordinate system.	Step 2: Transform the model grid-point to "N-1" data co-ordinate systems where N is the number of range images to be registered.
Step 3: Interpolate the distance value of transformed data grid-point in model co-ordinate system.	Step 3: Interpolate the distance value of transformed model grid-point in "N-1" data co-ordinate systems.
Step 4: Calculate the distance value of data grid-point and interpolated distance value of transformed data grid-point.	Step 4: Calculate the distance value of model grid-point and "N-1" interpolated distance values of transformed model grid-point.
Error minimization kernel	Error minimization kernel
Step 1: Read a data grid-point.	Step 1: Read a model grid-point.
Step 2: Calculate the partial derivatives for the data grid-point in model co-ordinate system.	Step 2: Calculate the partial derivatives for the model grid-point in "N-1" data co-ordinate systems.

Figure 5-5: Pair-wise (left) and multi-view registrations (right) GPU kernel.

To create a complete 3D model, the range images with a common overlap are first registered in pair-wise or multi-view fashion. Then the registered range-images are combined together using pair-wise registration. To compare the model creation process using pair-wise registration with multi-view registration, we registered continuous sequences of range images from four different models using pair-wise and multi-view registration. Figure 5-6 shows grouping of a sequence of 12 range images during model creation using pair-wise and multi-view registration. Each block in the figure represents one registration where Rn is the range images registered.

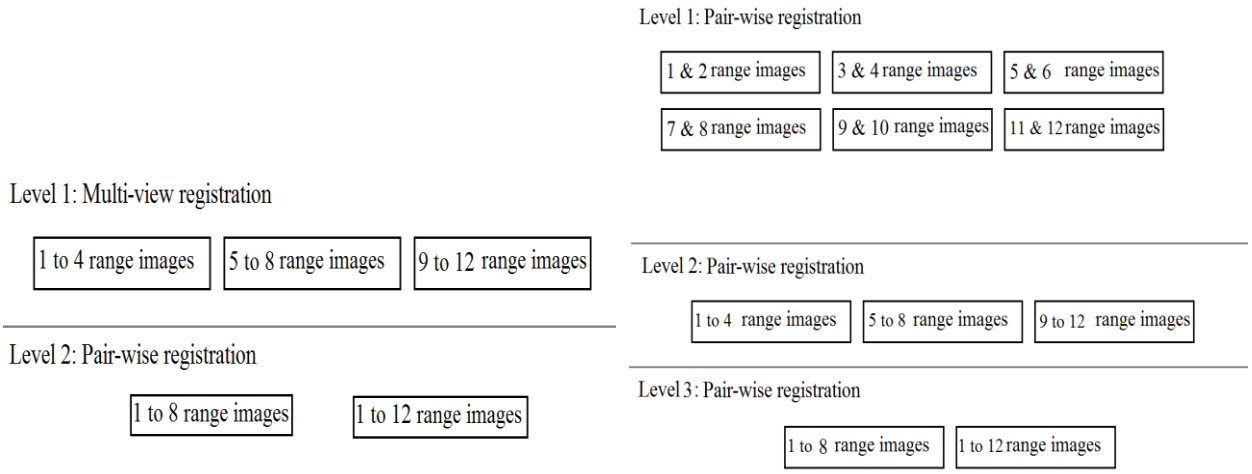

Figure 5-6: Grouping of range image during model creation using multi-view (left) and pair-wise registration (right).

Table 5-4 shows the time taken for model creation using pair-wise and multi-view registration with 16 CPU cores – 4 GPUs. We observed that model creation using multi-view registration was 25% - 30% faster than model creation using pair-wise registration.

Model	**Number of range images**	**Multi-view registration**	**Pair-wise registration**
Armadillo (515,000 points)	14	2287ms	2972ms
Bunny (350,000 points)	10	1614ms	2026ms
Pot (265,000 points)	12	1433ms	1781ms

Vase (150,000 points)	10	956 ms	1219 ms

Table 5-4: Time taken for model creation using pair-wise and multi-view registration.

The following pictures show the sequence of range images from the model "vase" registered using pair-wise registration to create a complete 3D-model.

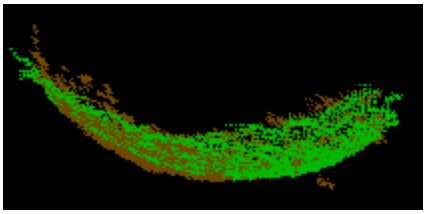

Figure 5-7: 1 and 2 range images.

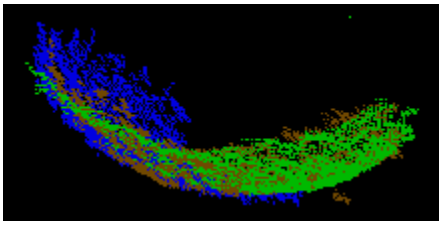

Figure 5-8: 1 to 3 range images.

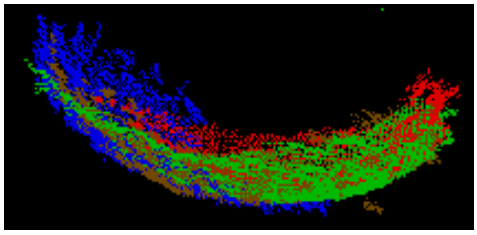

Figure 5-9: 1 to 4 range images.

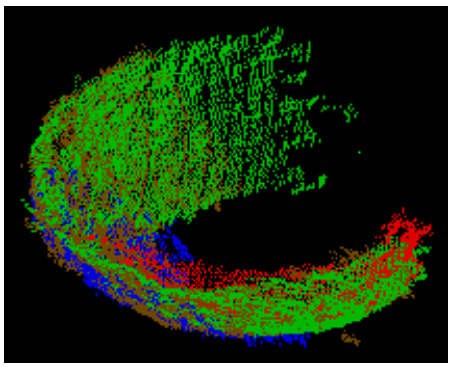

Figure 5-10: 1 to 6 range images.

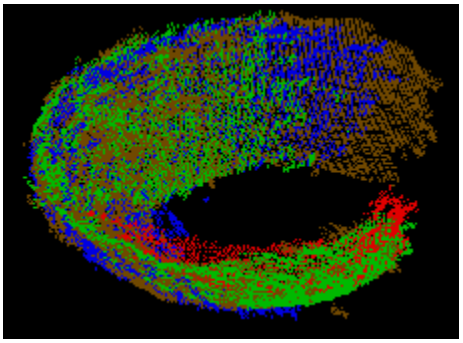

Figure 5-11: 1 to 7 range images.

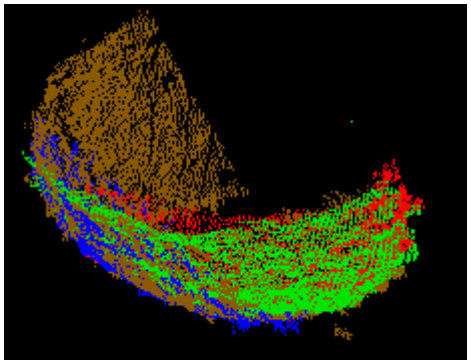

Figure 5-12: 1 to 5 range images.

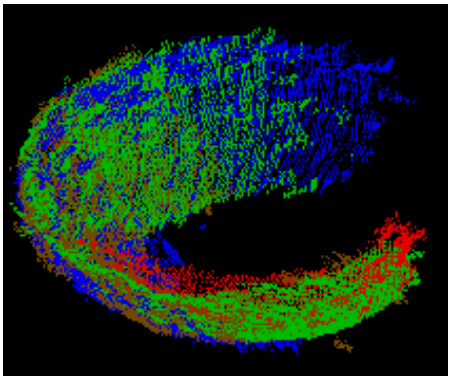

Figure 5-13: 1 to 8 range images.

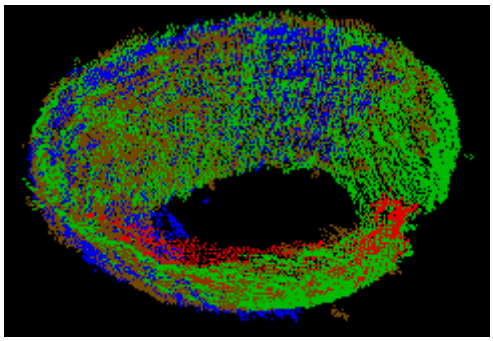

Figure 5-14: 1 to 9 range images.

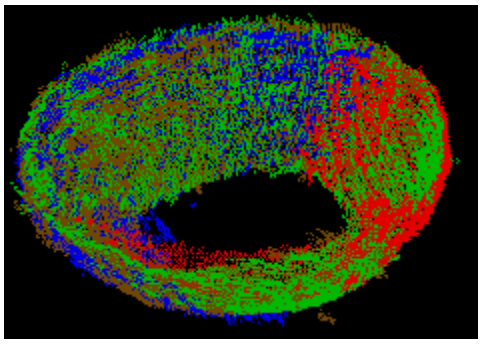

Figure 5-15: 1 to 10 range images.

The following pictures show the sequence of range images from the model "vase" registered using multi-view registration to create a complete 3D-model.

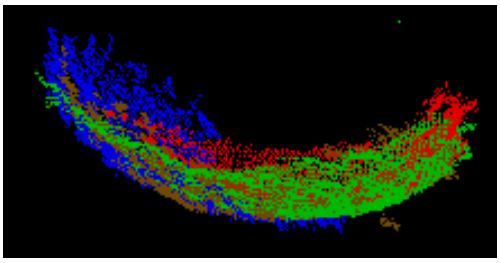

Figure 5-16: 1 to 4 range images.

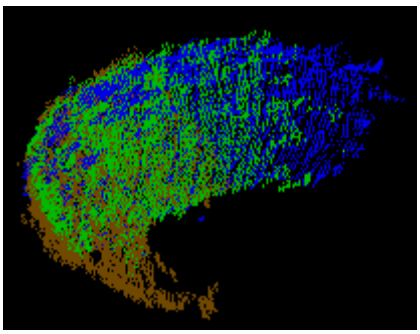

Figure 5-17: 9 to 12 range images.

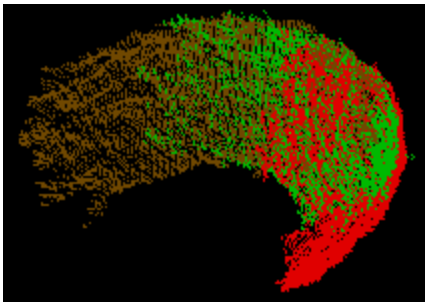

Figure 5-18: 5 to 8 range images.

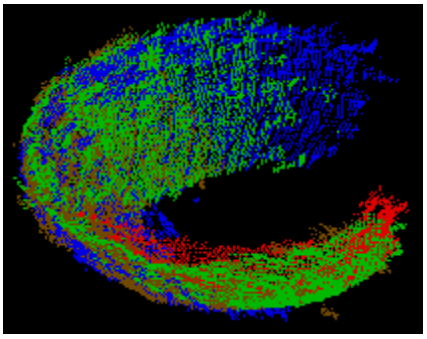

Figure 5-19: 1 to 8 range images.

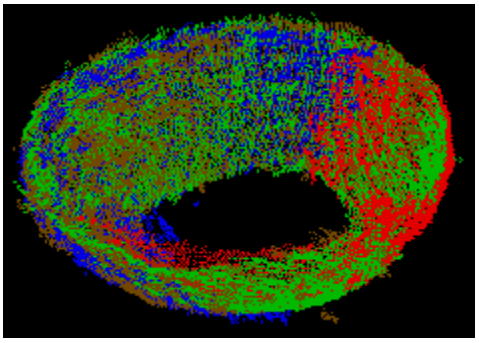

Figure 5-20: 1 to 12 range images.

5.3. Comparing multi-view registration with KinFu.

In August 2011, a group of researchers from Microsoft presented a system called KinectFusion for fast registration of range images obtained from Microsoft Kinect[7]. A user can create a complete 3-D model of an object by moving the Kinect slowly around the object. KinectFusion uses a single GPU to accelerate the Point-to-Plane variation of the ICP, and registers the obtained range images. The important part of ICP is choosing a good initial transformation for the range images that is close to the final transformation. KinectFusion solves this problem by assuming that the camera moves only a small distance between the two frames. So it uses the final transformation of the previous frame as the initial transformation of the next frame.

[7] Kinect is a sensor that supplies 30 range images per second.

After KinectFusion was demoed at SIGRAPH 2011, an open-source project called KinFu (a clone of KinectFusion) was started by Point Cloud Library (PCL). The beta version of KinFu was released in January 2012. In this section, we present our tests comparing multi-view registration with KinFu.

5.3.1. Comparing the accuracy

In this test, we compare the accuracy of our distance based multi-view registration[8] with KinFu by comparing the registration error. The registration error is measured by comparing the range images registered using multi-view and KinFu, with the baseline registration result from ICP.

KinFu requires a large overlap between the range images, as the range images used previously do not have sufficient overlap we scanned 15 range images each from three models (chair, computer table, and tripod). The graphs in Figure 5-21 and Figure 5-22 present the error for model-creation pair-wise and multi-view registrations while registering 15, 13, 10 and 8 range images for the three different models. The sub-sets of 13, 10 and 8 range images were randomly chosen for the three models to eliminate any bias.

[8] Note: Throughout this chapter we have used the term "multi-view registration" in place of "distance based multi-view registration" for brevity.

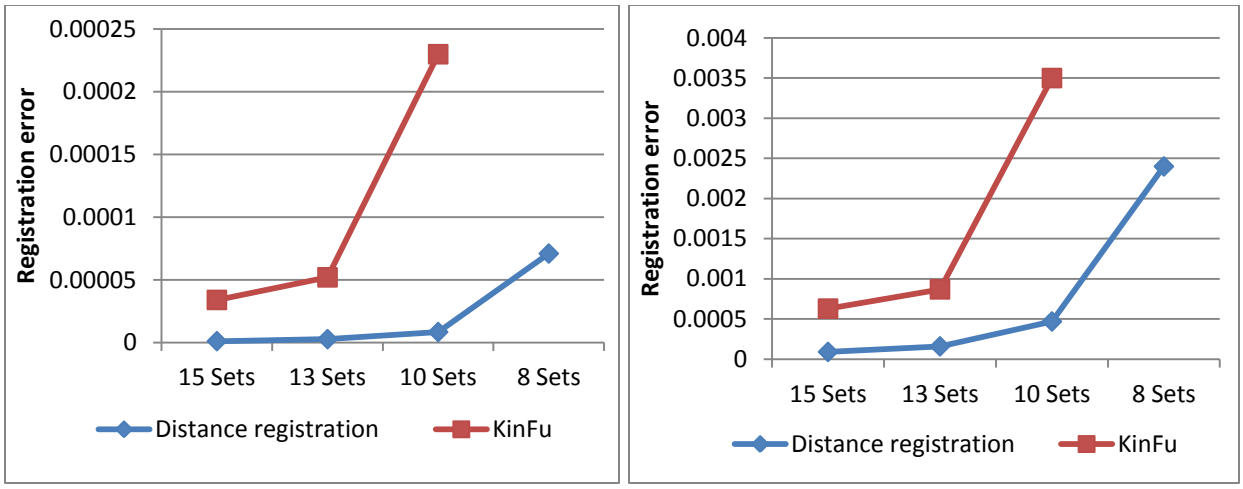

Figure 5-21: Final error for model chair (left) model computer table (right).

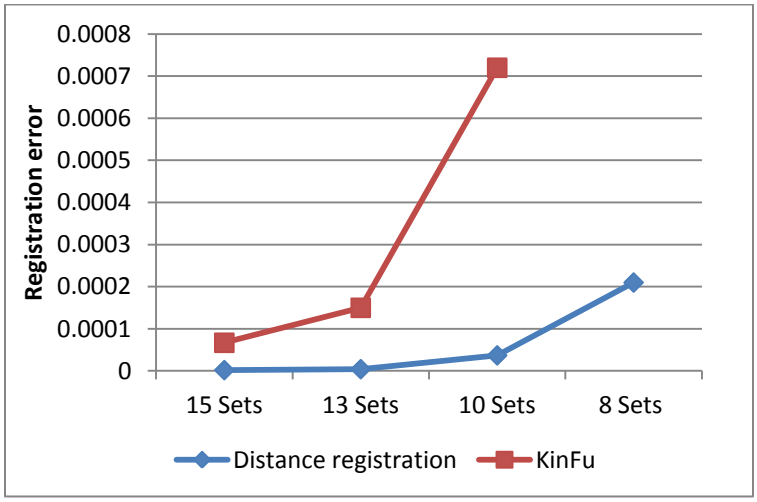

Figure 5-22: Final error for model tripod.

From the graphs in Figure 5-21 and Figure 5-22 we can observe that multi-view registration is consistently more accurate than KinFu. We can also observe that registering 10 range images using multi-view registration results in a more accurate model than registering 15 range images using KinFu.

The graphs in the Figure 5-23 and Figure 5-24 compare the error convergence of registering 10 range images (a random subset chosen from the total 15 range-images) using multi-view registration with the error convergence of registering 15 range images using KinFu. From the

graphs we can observe that multi-view registration converges to a smaller error than KinFu, this shows that during model creation multi-view registration requires only a sub-set of the range images required by KinFu to produce the same quality 3D model.

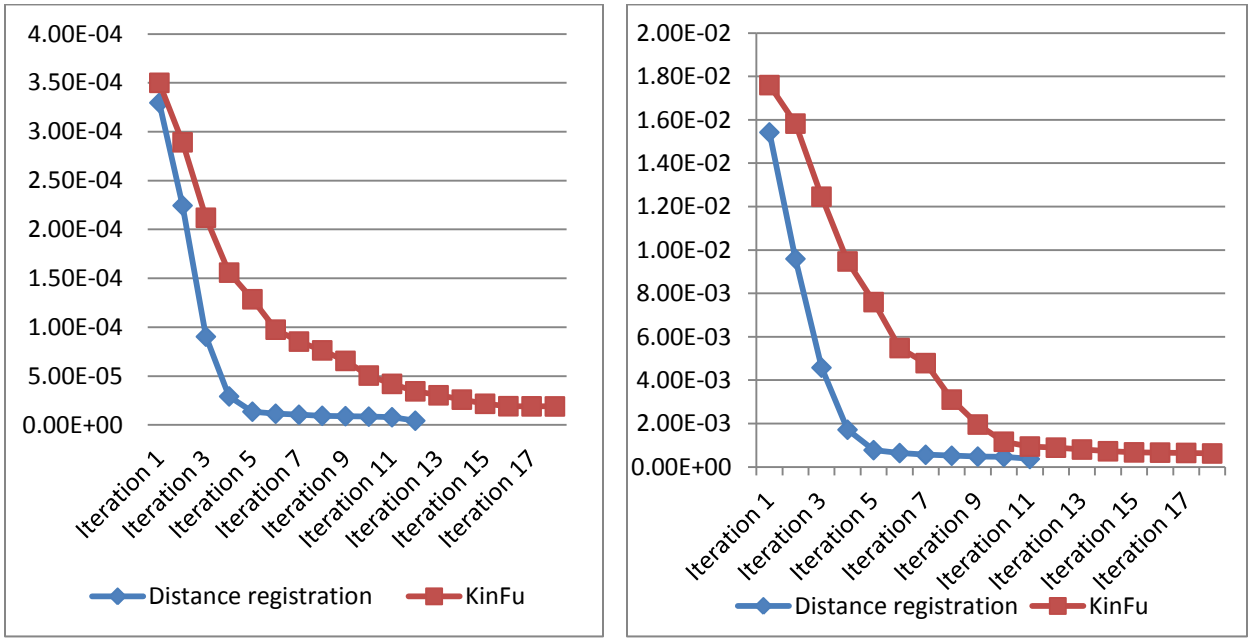

Figure 5-23: Error convergence model chair (left) model computer table (right).

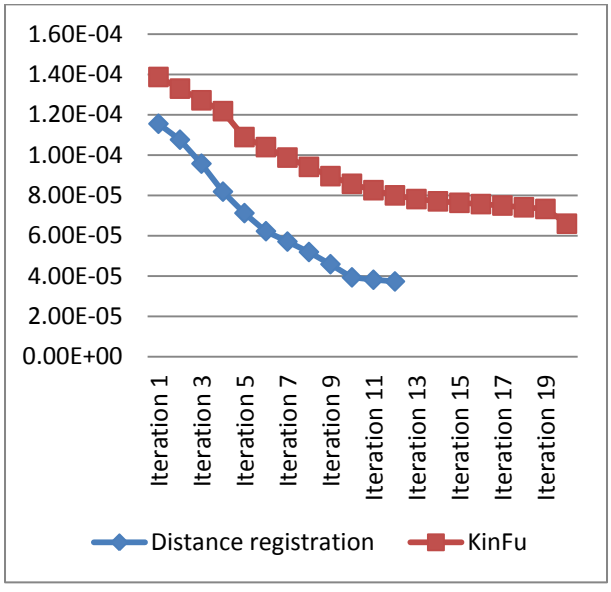

Figure 5-24: Error convergence model tripod.

Figures 5-25 to 5-31 show 15, 13, 10 and 8 range images registered (from the model chair) using multi-view registration and KinFu. The transformation parameters for the registered range images are presented in Appendix-B.

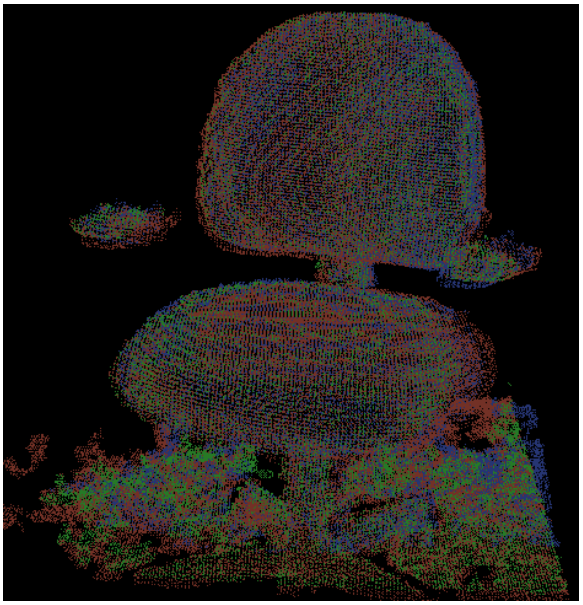

Figure 5-25: 15 range images distance registration.

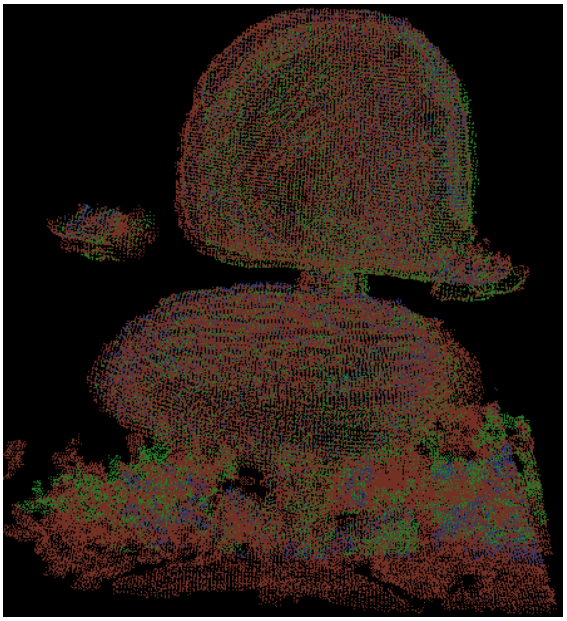

Figure 5-276: 15 range images KinFu.

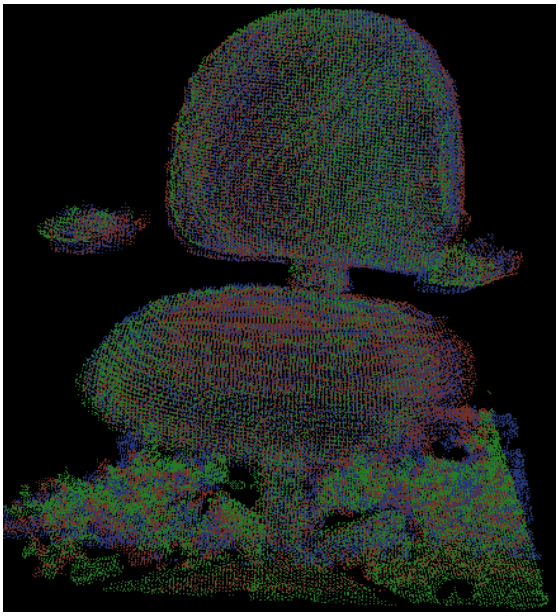

Figure 5-26: 13 range images distance registration.

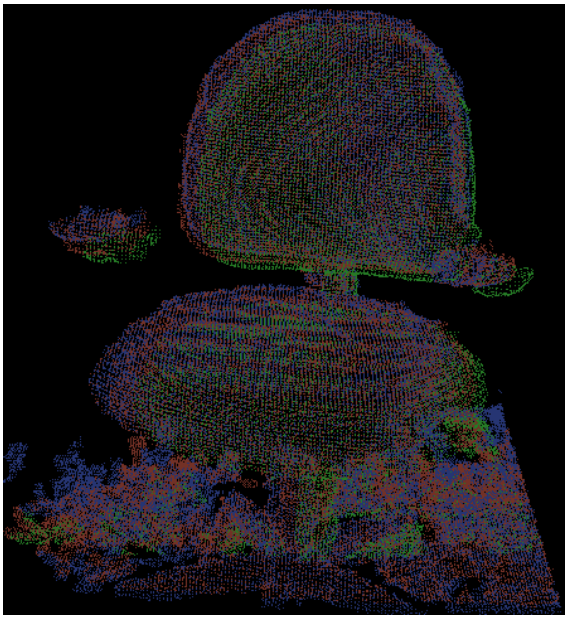

Figure 5-28: 13 range images KinFu.

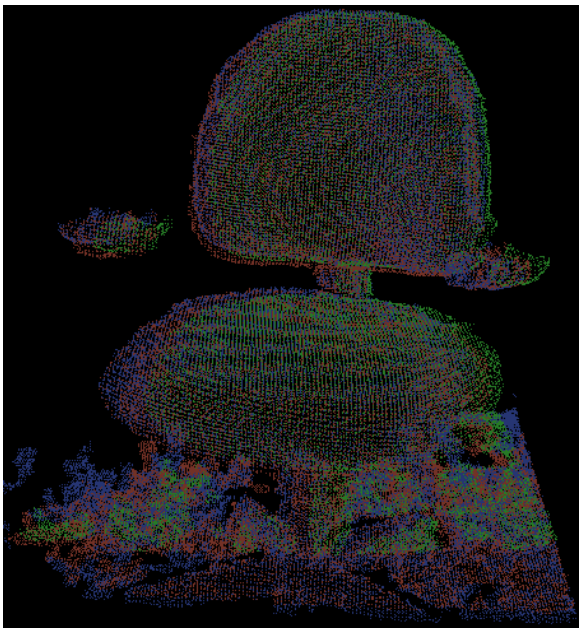

Figure 5-289: 10 range images distance registration.

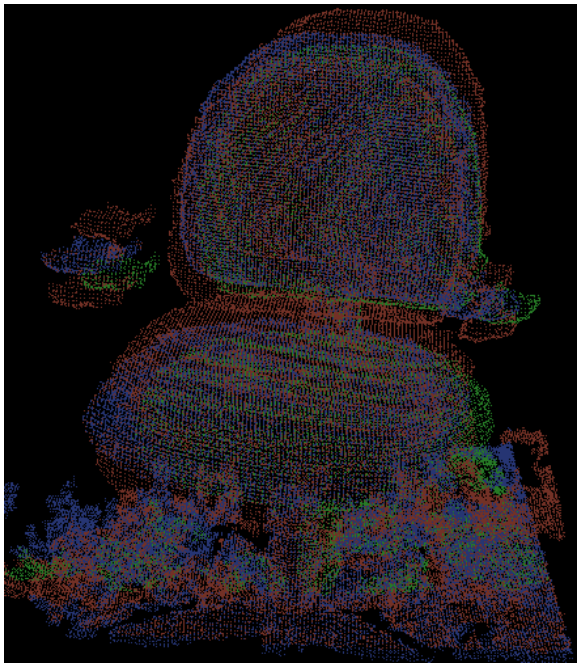

Figure 5-29: 10 range images KinFu.

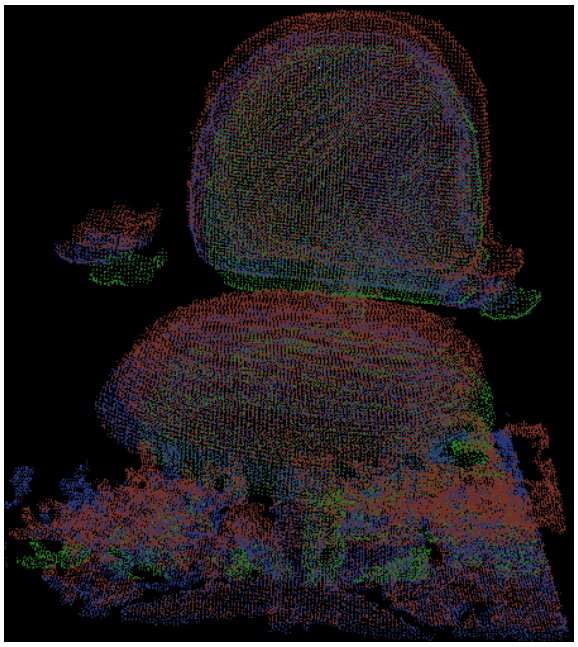

Figure 5-31: 8 range images distance registration.

5.3.2. Comparing the speed

The graphs below compares the time taken for registering 10 range images using multi-view registration and 15 range images using KinFu. We can observe that KinFu is faster than multi-view registration using 1 GPU-1 CPU combination. But while using 1 GPU – 4 CPU core combination multi-view registration is faster than KinFu.

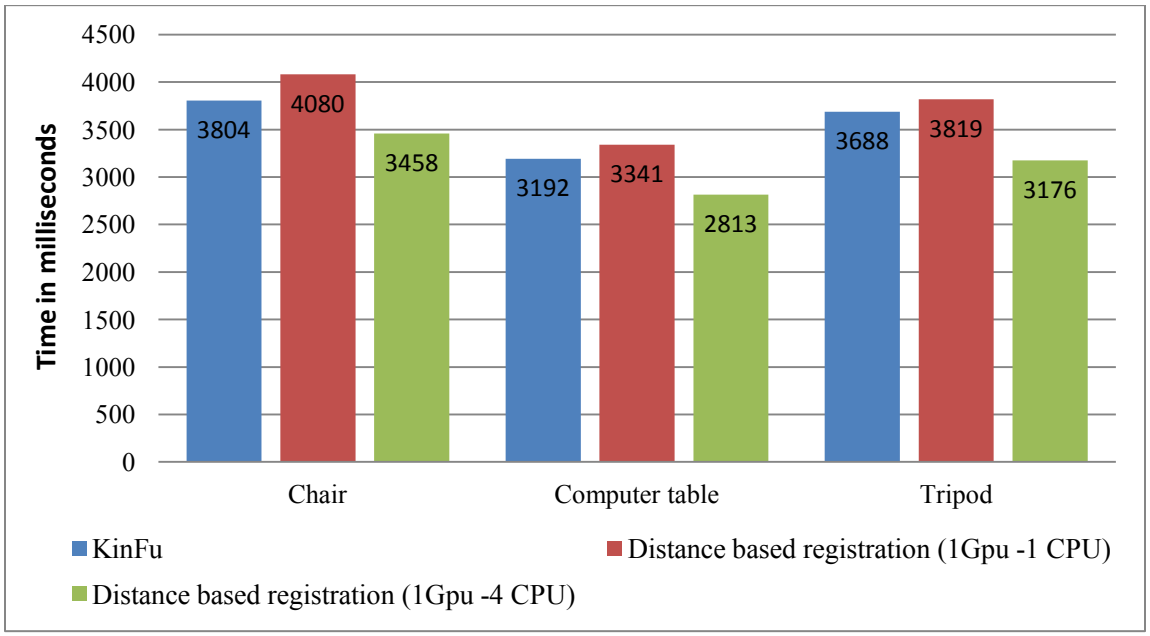

Figure 5-30: Registration time for multi-view registration and KinFu.

5.3.3. Comparing the speed on the cluster

To compare the time taken for model creation on the CPU - GPU cluster, we registered 10 range images using multi-view registration and 15 range images using KinFu on a 3 node cluster with 12 CPU cores and 3 GPUs (each node has 4 CPU cores – 1 GPU). Figure 5-33 shows the grouping of range images during model creation using KinFu and multi-view registration.

Each block in the figure represents one registration process, Rn is the ID of the registration and the numbers inside the block gives the range images registered in that particular registration.

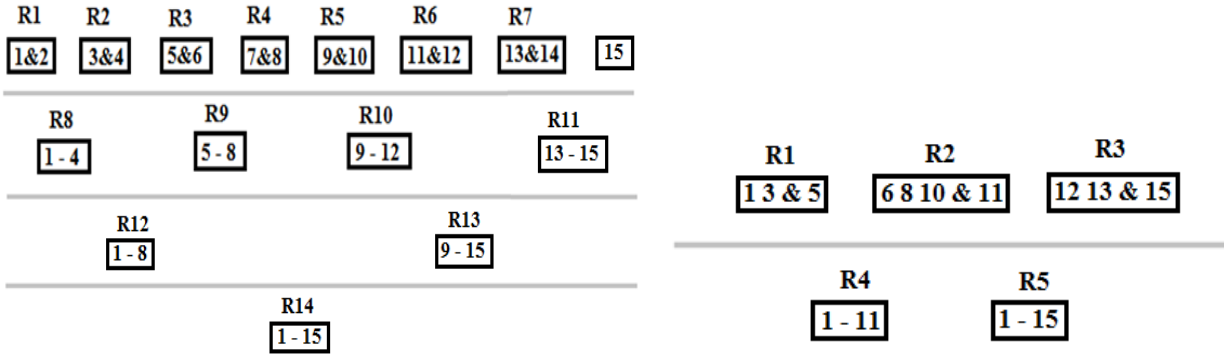

Figure 5-31: Grouping of range images during model creation using KinFu (left) and multi-view registration (right).

Figure 5-34 shows the scheduling of the registration process during model creation on the 3 node cluster. The available KinFu implementation only uses 1 CPU core – 1 GPU, so we created a wrapper to run multiple instances of KinFu on the cluster to distribute the registration process. But, as there is no algorithm to distribute a single KinFU registration process on multiple nodes, Nodes 2 and 3 are idle towards the end of the model creation when the number of available registration process falls below the number of nodes.

	Node 1	Node 2	Node 3
Level 1	R1	R2	R3
Level 2	R4	R5	R6
Level 3	R7	R8	R9
Level 4	R10	R11	R12
Level 5	R13	-----	----
Level 6	R14	-----	----

	Node 1	Node 2	Node 3
Level 1	R1	R2	R3
Level 2	R4	R4	R4
Level 3	R5	R5	R5

Figure 5-32: Scheduling KinFu (left) and multi-view registration (right) process on the cluster.

The graph in Figure 5-35 shows the time taken for model creation on the CPU - GPU cluster using multi-view registration and KinFu. From the graph we can observe that multi-view registration is 35% to 40% faster than KinFu on the CPU, GPU cluster.

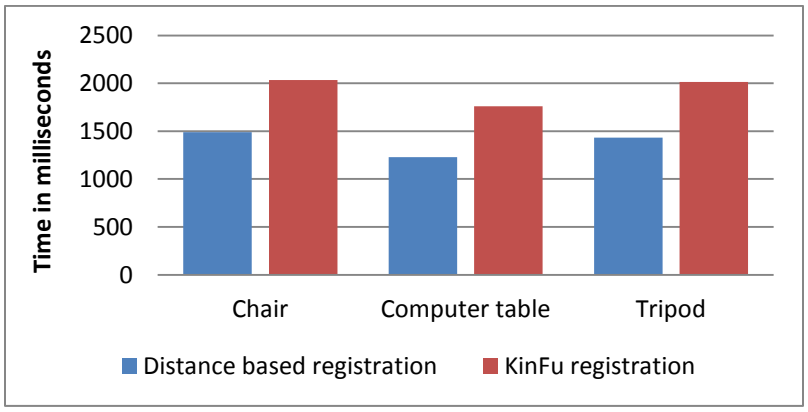

Figure 5-33: Time taken for model creation on the CPU - GPU cluster.

The graph in Figure 5-36 shows the measured % of CPU- GPU idle time of the cluster for different models. On an average 45% and 63% of CPU and GPU respectively is idle during out method, but 95% and 42% of the CPU and GPU respectively is idle during KniFu. This shows that multi-view registration uses the resources more efficiently than KinFu on the CPU and GPU.

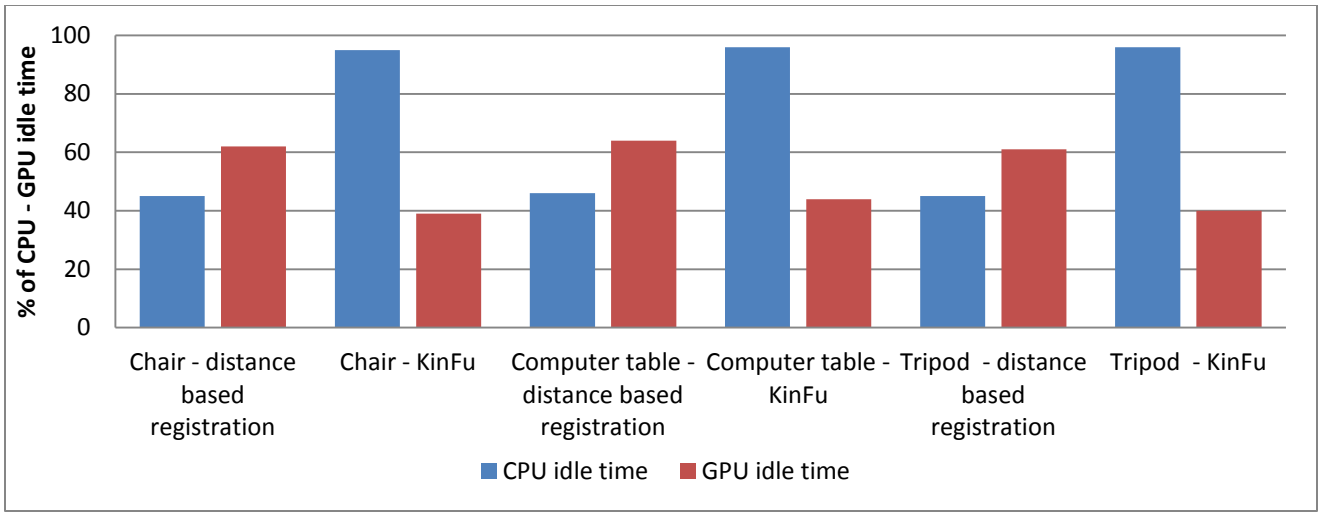

Figure 5-34: % of CPU- GPU idle time in the cluster.

5.4. Conclusion

In the first part of this chapter we presented our experiments with the following CPU and GPU combinations: < 4-CPU, 1-GPU >, < 8-CPU, 2-GPU >, < 12-CPU, 3-GPU > and < 16-CPU, 4-GPU >. Experimental studies have shown that, 1) influence based partitioning of registration is scalable as both problem size and the processors increase. 2) as the number of processors increase, the speed up achieved by multi-view registration consistently increases at a faster rate when compared to the pair-wise registration, showing that multi-view registration is more efficient than pair-wise registration in a multi-core CPU and GPU cluster environment.

In the second part of this chapter we presented our experiments comparing multi-view registration with KinFu. Our experiments comparing the accuracy have shown that registering 10 range images using multi-view method produces more accurate 3D models than registering 15 range images using KinFu. Our experiments to compare the registration time have shown, KinFu is faster than multi-view registration for the combination < 1-CPU,1-GPU >, while registration is faster than KniFu for the combinations < 4-CPU,1-GPU >, and < 12-CPU,3-GPU >. To

summarize our experiments presented in the second part of the chapter, multi-view registration only needs a sub-sample of the range images required by KinFu and multi-view registration is faster than KniFu in a multi-core CPU and GPU environment.

Chapter 6: Conclusion and future work

Multi-core CPUs and GPUs have become part of virtually every computer. The programmability of the multicore CPUs and GPUs now make it possible to accelerate applications. The focus of our research has been to study the use of multi-core CPUs and GPUs to accelerate 3D registration. While there have been few attempts to accelerate 3D registration using GPUs, ours is the first research investigation to use functionality distribution in accelerating 3D registration using both multi-core CPUs and GPUs. Based on this investigation we formulated a first of its kind method to accelerate 3D registration using a cluster of multi-core CPUs and GPUs. It is these investigations that led us to formulate our thesis that multi-view registration is more efficient than pair-wise registration to accelerate 3D registration using a multi-core CPU and GPU cluster.

6.1. Conclusions and Contributions

In chapter 3, we proposed our functionality distribution scheme for a single computer with one multi-core CPU and one GPU to accelerate the distance based registration. To devise the proper functionality distribution, we experimentally compared three different methods to accelerate distance based registration. The speed-up was measured by comparing with the baseline performance obtained on a single core CPU with no GPU. Experiments conducted using four different data sets (two obtained from Stanford and two scanned in our lab), showed 3.6 times speed-up for pair-wise registration and 4.2 times speed-up for multi-view registration is possible while using a Tesla C010 GPU and Intel E5510, with functionality distribution. But without functionality distribution only 1.9 for times speed-up for pair-wise and 2.1 times for multi-view registration is possible.

In chapter 4, we extended the functionality distribution scheme for accelerating the distance based registration, to a cluster of multiple multi-core CPUs and multiple GPUs. We devised two methods 1) spatial based partitioning and 2) influenced grid points based partitioning. Our experiments showed 10.4 times speed-up for pair-wise registration and 12.36 times speed-up for multi-view registration is possible with method-2, while only 7.3 times speed-up for pair-wise registration and 8.7 times speed-up for multi-view registration is possible with method-1.

In first part of chapter 5, we demonstrate the scalability of the influenced grid points based partitioning method with the following CPU and GPU combinations: <4-CPU, 1-GPU >, <8-CPU, 2-GPU >, <12-CPU, 3-GPU > and <16-CPU, 4-GPU>. The experimental studies showed a minimum of 3.5 times for pair-wise registration and 4 times for multi-view for the combination <4,1>; and a maximum of 9 times speed up for pair-wise registration and 11.5 times for multi-view registration for the combination <16,4> are possible with method-2. This demonstrates that in a multi-core CPU and GPU cluster, as the number of processors increase, the speed up achieved by multi-view registration consistently increases at a faster rate when compared to the pair-wise registration, showing the multi-view registration is more efficient than pair-wise registration.

In the second part of chapter 5, we compared the accuracy and registration time of multi-view registration with KinFu. Experiments comparing the accuracy have shown that, registering 10 range images using multi-view method produces more accurate 3D models than registering 15 range images using KinFu. Experiments comparing the registration time have shown that KinFu is 1.2 times faster than multi-view registration for the combination <1-CPU,1-GPU>. But as KinFu does not use all the CPU cores, multi-view registration is 1.1 times and 1.4 times faster

than KniFu for the combinations <4-CPU,1-GPU>, and <12-CPU,3-GPU> respectively. This shows that multi-view registration only needs a sub-sample of the range images required by KinFu and multi-view registration is faster than KniFu in a multi-core CPU and GPU environment.

6.2. Future work

There are still several areas to explore further in this work. A few of the possible areas are listed below.

- During our research we assumed a homogeneous cluster i.e. all the multi-core CPUs and GPUs are of the same type. In future we would like to extend our algorithm to heterogeneous clusters with different types of multi-core CPUs, GPUs and also other kinds of processors.

- Both multi-core CPUs and GPUs are not active at the same time. To further improve the speed-up we would like to introduce pipelining so that when GPUs are in the error calculation and minimization phase for one group of range-images the multi-core CPUs can compute the distance values for the next group of range-images.

- The grouping of range-images during registration is manual. We could develop an approach to automate the grouping of range images.

Appendix A

This section presents the models scanned in our lab.

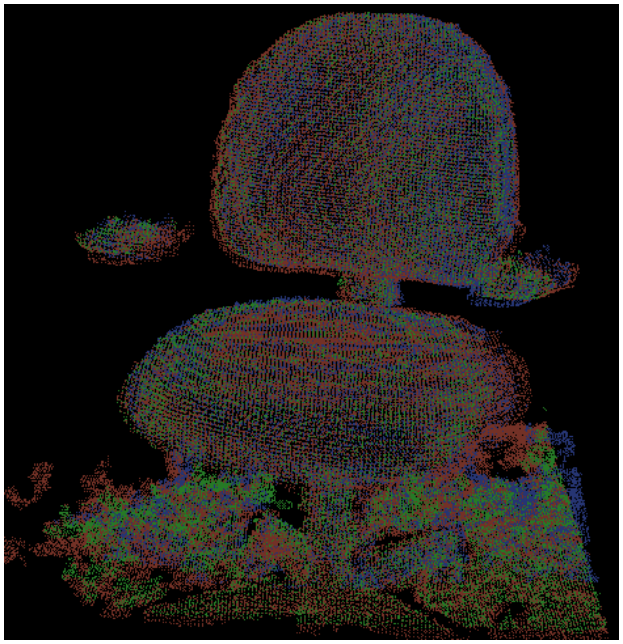

Figure A-1: Chair.

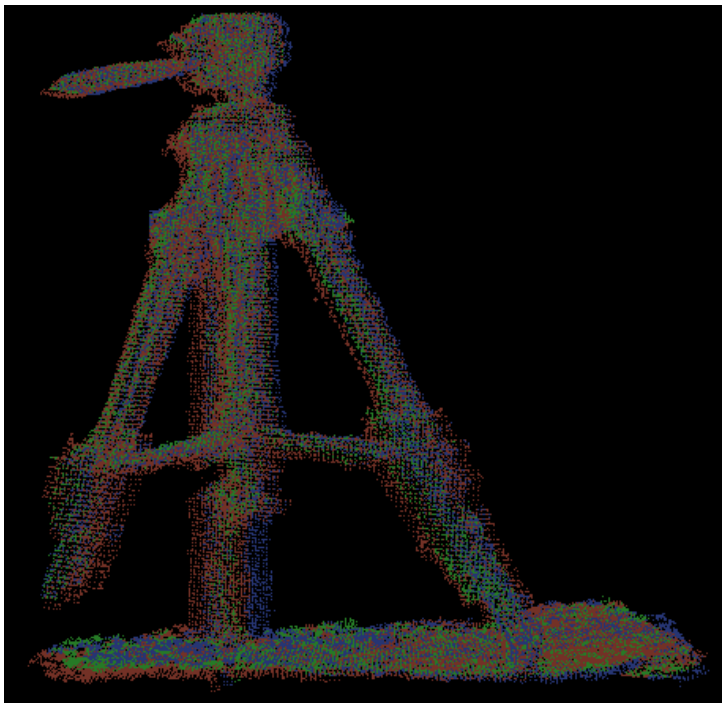

Figure A-2: Tripod.

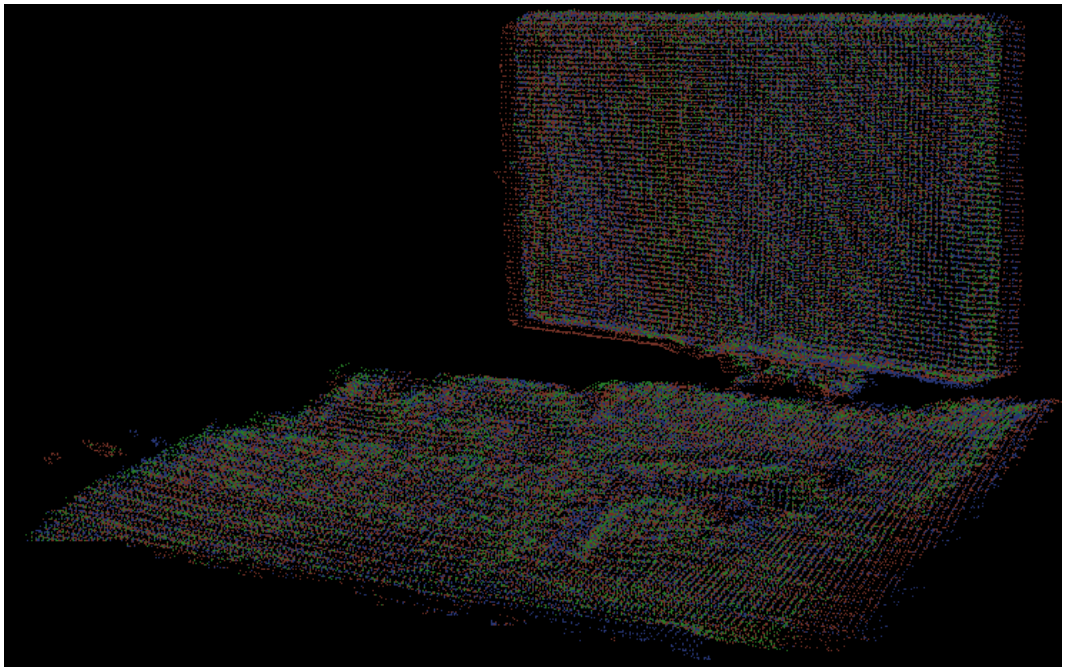

Figure A-3: Computer table.

Appendix B

Tables A-1 to A-6 give the difference in transformations[9] (transformation error) of nine range images when compared to the base line ICP transformation for the models chair, computer table and tripod. The transformation error values of 'Distance based registration' that are greater than the transformation error values of 'KinFu' is highlighted. From the tables we can observe that the transformation error values of 'Distance based registration' are less than 'Kinfu' this shows that 'Distance based registration' is more accurate than 'Kinfu'.

Model chair

Distance based registration.

Q1	Q2	Q3	Q4	tx	ty	tz
0.054677	0.00039	0.00277	0.00012	0.22323	0.21139	0.03684
0.061884	0.00277	0.00835	0.00079	0.26485	0.237494	0.03273
0.096615	0.003462	0.01882	0.00193	0.24624	0.21572	0.02538
0.048158	0.004301	0.0169	0.00259	0.27019	0.230802	0.0202
0.118041	0.01708	0.05841	0.00136	0.23428	0.19244	0.01006
0.023007	0.002278	0.01253	0.00092	0.20039	0.16294	0.00764
0.169401	0.024064	0.10858	0.01338	0.28278	0.22797	0.0097
0.085751	0.013396	0.05902	0.00706	0.23143	0.18423	0.00553
0.078129	0.02075	0.05542	0.00646	0.40969	0.323567	0.00675

Table A-1: Transformation error values of distance based registration

KinFu.

Q1	Q2	Q3	Q4	tx	ty	tz
0.105888	0.0002069	0.0036	0.0004	0.40231	0.40937	0.0135
0.135611	0.0012664	0.013	0.0073	0.58038	0.520424	0.0172
0.13557	0.004858	0.0201	0.0027	0.4878	0.36686	0.04316
0.124295	0.011101	0.00362	0.00669	0.3187	0.411632	0.0303

[9] The transformation parameters give the alignment between the reference range image and another range image. The transformation parameters have a rotational component represented in quaternion (Q1, Q2, Q3, and Q4) and a translational component (tx, ty and tz).

0.228034	0.0088445	0.07284	0.00064	0.4259	0.37176	0.00244
0.087546	0.008668	0.04768	0.0035	0.2363	0.32006	0.01501
0.155446	0.022081	0.09963	0.01228	0.2525	0.32671	0.01389
0.142406	0.022247	0.09802	0.0173	0.5558	0.44244	0.00139
0.125812	0.012897	0.08925	0.0104	0.69205	0.556567	0.01141

Table A-2: Transformation error values of kinfu

Model computer table

Distance based registration.

Q1	Q2	Q3	Q4	tx	ty	tz
0.031844	0.000119906	0.00161	7.2E-05	0.130007	0.123109	0.02146
0.032929	0.00167359	0.01044	0.0042	0.140928	0.361367	0.01741
0.029375	0.0015252	0.01572	0.0059	0.336021	0.119158	0.01402
0.026747	0.00837882	0.00939	0.00144	0.150069	0.128193	0.01122
0.077532	0.00580899	0.03837	0.0009	0.353881	0.126396	0.00661
0.058963	0.00583787	0.03211	0.00236	0.360492	0.130498	0.00612
0.055709	0.01791356	0.09571	0.0044	0.159759	0.297194	0.0548
0.058202	0.00909253	0.04006	0.0479	0.157083	0.125045	0.00376
0.024653	0.00406739	0.01749	0.0204	0.152621	0.120537	0.0252

Table A-3: Transformation error values of distance based registration

KinFu.

Q1	Q2	Q3	Q4	tx	ty	tz
0.069225	0.000260665	0.0035	0.00016	0.282624	0.267628	0.04665
0.071584	0.001464326	0.00966	0.00092	0.306365	0.274711	0.03786
0.063858	0.002288087	0.01244	0.00127	0.295698	0.259039	0.03048
0.058146	0.005193087	0.02041	0.00313	0.326237	0.27868	0.02439
0.168547	0.012628239	0.08341	0.00195	0.334524	0.274774	0.01437
0.128181	0.012691022	0.06981	0.00513	0.348896	0.283691	0.0133
0.121107	0.017203391	0.07762	0.00956	0.347302	0.279987	0.01191
0.126527	0.01976637	0.08709	0.01042	0.341485	0.271837	0.00816
0.053593	0.008842152	0.03802	0.00443	0.331785	0.262037	0.00547

Table A-4: Transformation error values of kinfu

Model tripod

Distance based registration.

Q1	Q2	Q3	Q4	tx	ty	tz
0.022745	8.56471E-05	0.00115	5.1E-05	0.192862	0.087935	0.01533
0.02352	0.00098136	0.00917	0.0013	0.100663	0.090262	0.01244
0.020982	0.0027518	0.00409	0.00042	0.097158	0.085113	0.02001
0.019105	0.0017063	0.00671	0.00103	0.107192	0.091566	0.00802
0.05538	0.008149279	0.07141	0.0064	0.109915	0.090283	0.00472
0.042117	0.00969907	0.04294	0.00169	0.114637	0.093213	0.00437
0.039792	0.005652543	0.0255	0.00314	0.114114	0.091996	0.03191
0.041573	0.04294664	0.02862	0.01342	0.242202	0.089318	0.00528
0.017609	0.022905279	0.01249	0.00426	0.229015	0.086098	0.0118

Table A-5: Transformation error values of distance based registration

KinFu.

Q1	Q2	Q3	Q4	tx	ty	tz
0.039804	0.000149883	0.00201	9E-05	0.162509	0.153886	0.02682
0.041161	0.000841988	0.00555	0.00053	0.17616	0.157959	0.02177
0.036718	0.00131565	0.00715	0.00073	0.170026	0.148948	0.01752
0.033434	0.002986025	0.01173	0.0018	0.187586	0.160241	0.01403
0.096914	0.007261238	0.04796	0.00112	0.192351	0.157995	0.00826
0.073704	0.007297338	0.04014	0.00295	0.200615	0.163123	0.00765
0.069637	0.00989195	0.04463	0.0055	0.199699	0.160993	0.00685
0.072753	0.011365663	0.05008	0.00599	0.196354	0.156306	0.00469
0.030816	0.005084238	0.02186	0.00255	0.190776	0.150671	0.00315

Table A-6: Transformation error values of kinfu

Glossary of terms

1. Range Image

The raw data obtained from a scanner consists primarily of 3d point cloud called range-images. Typically a modern scanner can produce 30 range images per second.

2. 3D Registration

These range images obtained from the 3d scanner needs to be merged to construct a complete 3d model. The process of merging the range images is called 3d registration.

3. Texture

Along with the 3d points in the range image the color at each 3d point can also be recorded. The color is usually represented in RGB format. The colors at all the 3d points are together known as texture map.

4. Transformation parameters / Transformation matrix

Transformation parameters are a function between two vector spaces. Transformation parameters have a rotation and a translation component. The registration problem can be defined as a process of determining the transformation parameters to be associated with each range image, to merge all the range images.

5. Initial transformation parameters

The registration algorithm requires an approximate transformation parameter as an input, we call this initial transformation parameters.

6. Final transformation parameters

The transformation parameters returned after registration is the final transformation parameters.

7. Heterogeneous cluster

A computer cluster consists of several computers that are connected to each other through fast local area networks. The computers in a heterogeneous cluster have different types of processors like single/multi core CPU, GPU, Cell processors. The heterogeneous cluster used for this thesis had a multi-core CPUs and GPUs.

8. Node

A node refers to a single computer in a cluster. Each node must have a processor, memory, network card, and an operating system. A node may or may not have secondary storage. The nodes used for this thesis had a secondary storage.

9. Master node

A master node is a node that co-ordinates all the other nodes in the cluster. A cluster may have one or more master nodes. In this thesis we used a single master node. The master node send's instructions, receives intermediate results and combines the results received from all the other nodes.

10. Slave node

A slave node is a node that receives instructions from the master node, processes the instructions and sends the results back to the master node.

11. Microsoft Kinect

Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs. The kinect has a depth sensor, RGB camera and a microphone. The depth senor in kinect can be used as a 3d scanner. The low-cost of kinect compared to the traditional 3d scanners has made it a popular chose among the researchers.

References

[1]Kari Pulli, Marc Levoy, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. Proceedings of ACM SIGGRAPH 2000, pp. 131-144, July 2000.

[2]Park S.Y, Choi. S.I, Kim .J.: "Real-time 3d registration using GPU", Machine Vision and Applications, vol.225,pp. 837-850, 2011.

[3]Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D. "The Digital Michelangelo Project: 3d Scanning of Large Statues," Proc. SIGGRAPH, 2000.

[4]R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "A survey of medical image registration on multicore and the GPU," IEEE Signal Processing Magazine, vol. 27, no. 2, Article ID 5438962, pp. 50–60, 2010.

[5]Pulli, K., ''Multiview registration for large data sets,'' Proc. 2nd Int'l Conf. on 3-D Digital Imaging and Modeling, IEEE, 1999, pp. 160-168.

[6]Blais, G. and Levine, M. "Registering Multiview Range Data to Create 3d Computer Objects," Trans. PAMI, Vol. 17, No. 8, 1995.

[7]Jason Sanders and Edward Kandrot. 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming (1st ed.). Addison-Wesley Professional.

[8]Yamany, S., Farag, A.: Free-form surface registration using surface signatures. In: Proceedings of 7th International Conference on Computer Vision, pp. 1098–1104 (1999).

[9]Johnson, A., Hebert, M.: Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Trans. Pattern Anal. Mach. Intell. 21(5), 433–449 (1999).

[10]C. J. R. Chua, Point signatures: A new representation for 3d object recognition, International Journal of Computer Vision 25(5), pp. 63–85, October 1997.

[11]Besl, P., McKay, N.:Amethod for registration of 3-D shapes. IEEE Trans. Pattern Anal. Mach. Intell. 14(2), 239–256 (1992).

[12]K. Nishino and K. Ikeuchi, "Robust Simultaneous Registration of Multiple Range Images", Proc. of Fifth Asian Conference on Computer Vision ACCV '02, pp454-461, Jan., 2002.

[13]R. Benjemaa and F. Schmitt. Fast global registration of 3d sampled surfaces using a multi-z-buffer technique. In Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling, pages 113–120, May 1997.

[14]Blais, G., Levine, M., 1995. Registering multiview range data to create 3d computer object. IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (8).

[15]P.J. Neugebauer, "Geometrical cloning of 3D objects via simultaneous registration of multiple range images," sma, pp.130, 1997 International Conference on Shape Modeling and Applications (SMA '97), 1997

[16]Soon-Yong Park, Murali Subbarao: An accurate and fast point-to-plane registration technique. Pattern Recognition Letters 24(16): 2967-2976 (2003)

[17]Olaf Hall-Holt and Szymon Rusinkiewicz. Stripe Boundary Codes for Real-Time Structured-Light Range Scanning of Moving Objects. Eighth International Conference on Computer Vision (ICCV), July 2001.

[18]S. B. Kang and A. E. Johnson. Registration and integration of textured 3-D data. In 3DIM, 1997.

[19]Chitra Dorai, Juyang Weng, Anil K. Jain: Optimal Registration of Object Views Using Range Data. IEEE Trans. Pattern Anal. Mach. Intell. 19(10): 1131-1138 (1997).

[20]Sharp, G.C.; Lee, S.W.; Wehe, D.K.; , "Multiview registration of 3D scenes by minimizing error between coordinate frames," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.26, no.8, pp.1037-1050, Aug. 2004

[21]Qiu, D., May, S., Nüchter, A.: GPU-accelerated nearest neighbor search for 3d registration. In: Proceedings of the 7th International Conference on Computer Vision Systems (ICVS '09), LNCS, vol. 5815, pp. 194–203 (2009).

[22]Choi, S.I., Park, S.Y., Kim, J., Park, Y.W.: Multi-view range image registration using CUDA. In: International Technical Conference on Circuits/Systems, Computers and Communications (2008).

[23]Kitaaki, Y., Okuda, H., Kage, H., Sumi, K.: High speed 3-D registration using GPU. In: SICE Annual Conference, pp. 3055–3059 (2008).

[24]Fast multi-core based multimodal registration of 2D cross-sections and 3d datasets Michael Scharfe, Rainer Pielot, and Falk Schreiber.

[25]Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon, KinectFusion: Real-Time Dense Surface Mapping and Tracking in ISMAR 2011

[26]Bhakar. S, Wang. R, Mudur. S.P, "Multi-view 3d Scanned Data Registration", Proceedings of the 2008 C3S2E conference on C3S2E 08 (2008)

[27]S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. In ACM Transactions on Graphics (SIGGRAPH), 2002.

[28]W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, Cambridge, England, second edition, 1992.

[29]Chen,Y., Medioni,G.: Object modelling by registration ofmultiple range images. Image Vis. Comput. 10(3), 145–155 (1992).

[30]The Stanford 3D Scanning Repository. <http://graphics.stanford.edu/data/3Dscanrep/>

[31]Pavan Balaji, Darius Buntinas, Ralph Butler, and Anthony Chan. "MPICH2 User's Guide." http://www.mcs.anl.gov. Ed. Mathematics and Computer Science Division Argonn, n.d. Web. <http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-1.4.1-userguide.pdf>.

[32]"OpenMP Application Program Interface." http://www.mcs.anl.gov. http://www.openmp.org, n.d. Web. <http://www.openmp.org/mp-documents/OpenMP3.1.pdf>.

[33]"Optimizing CUDA part 3." http://www.nvidia.com. Nvidia, n.d. Web. <http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_Further_CUDA_Optimization.pdf>.

[34]S.H. Gunther et al., “Managing the Impact of Increasing Microprocessor Power Consumption,” Intel Technology Journal 1st quarter, 2001.

[35]"An open source implementation of KinectFusion." http://www.pointclouds.org. Pointclouds, n.d. Web. <http://www.pointclouds.org/news/kinectfusion-open-source.html>.