CUDA Accelerated 3D Non-rigid Diffeomorphic Registration

AN QU
CUDA Accelerated 3D Non-rigid Diffeomorphic Registration

An Qu

Master Thesis in Medical Engineering
KTH Royal Institute of Technology
February 2017
Abstract

Advances of magnetic resonance imaging (MRI) techniques enable visual guidance to identify the anatomical target of interest during the image guided intervention (IGI). Non-rigid image registration is one of the crucial techniques, aligning the target tissue with the MRI preoperative image volumes. As the growing demand for the real-time interaction in IGI, time used for intraoperative registration is increasingly important. This work implements 3D diffeomorphic demons algorithm on Nvidia GeForce GTX 1070 GPU in C++ based on CUDA 8.0.61 programming environment, using which the average registration time has accelerated to 5s. We have also extensively evaluated GPU accelerated 3D diffeomorphic registration against both CPU implementation and Matlab codes, and the results show that GPU implementation performs a much better algorithm efficiency.

Keywords: diffeomorphic demons, non-rigid image registration, parallel programming, GPGPU, IGI
Acknowledgments

I am very grateful for Prof. Ka-Wai-Kwok providing me such an awesome working opportunity in his group of Interventional Robotic and Imaging Systems, the University of Hong Kong. Without his supervision and helps in experiment resources, there wouldn’t have been this work.

I would like to express my sincere gratitude to the smartest guy, Martin Leong, for his everyday discussion, helps in programming and great ideas in GPU performance optimization.

Furthermore I would like to thank Prof. Rodrigo Moreno, Prof. Dmitry Grishenkov and the supervision group for their critical but constructive comments, which helped me to improve the quality of my thesis time over time.

Finally I would like to thank my boyfriend who has been always encouraging me during the whole period.

Thanks for all your encouragement!
1 Introduction

The development of modern technology has significantly promoted the rapid progress in medical field, reflecting in medical imaging, medical measurement, recording and monitoring and medicine invention. Medical imaging, as the most transformational technique, leads to new insights in biology and therefore contribute to proper and correct diagnosis and improved treatment. Image-guided intervention (IGI) is one of the popular applications of medical imaging, where interventionist is able to reach the treatment target accurately then achieve the minimally invasion with the help of image registration technique. Nowadays, there are multiple imaging modalities available for it, such as X-ray, nuclear medicine, ultrasound and magnetic resonance imaging. As MRI performs much better for not only diagnosis but also treatment, we in this work are interested in a very important aspect of MRI applications: intra-operative visual guidance.

Advances of magnetic resonance imaging (MRI) techniques enable visual guidance to identify the anatomical target of interest during the medical diagnosis and treatment. With it, either the position of surgical apparatus or the critical/target tissue can be virtually augmented/aligned together with the MRI images, with image registration. It is an indispensable tool in many robot assisted minimally invasive surgeries nowadays in order to enhance surgical safety, accuracy, and efficiency. However, potential misalignment is one of the common problems of image registration in many soft tissue surgeries, which affects the certainty and accuracy of image registration. It results from tissue deformation caused by physiological motion or tool tissue interaction. For example, “brain shift” due to the loss of cerebrospinal fluid after craniotomy in robot assisted neurosurgery and repeated pumping motion of the heart in the robot assisted cardiac physio electrotherapy. Non-rigid image registration is a solution for the misalignment [1]. It is capable of locally warping the target image to align with the reference image to cope with the deformation of the subject due to breathing and anatomical changes. Except for the certainty and accuracy of image registration, computation speed is also significantly important in many robot assisted minimally invasive surgeries where real time feedbacks are required. Despite many well established open source packages are widely used in the community, the computation speed and registration accuracy they provide cannot satisfy the requirement of clinical robot assisted surgeries. Therefore, a significant research gap exists between the existing image registration techniques and their implementation on clinically ready robots.

Rather than developing a new algorithm for MRI image registration, this work aims to construct an accelerated GPU diffeomorphic registration by using multiple parallelization techniques in CUDA programming environment. In addition, an open source implementation can be produced in Insight Segmentation and Registration Toolkit(ITK), which is a widely-used open source library for medical image segmentation and registration, since the existing GPU based diffeomorphic demons is very unoptimized. In order to better evaluate our GPU accelerated diffeomorphic demons registration, we have also implemented the algorithm on CPU and compare the registration speed from these two version. In addition, a comparison against Matlab version algorithm is also performed. The experiment results show that registration implemented in Matlab takes the longest time while GPU accelerated registration is the most efficient, which has achieved approximately 5 seconds.
2 Related Work

As image-guided intervention provides clinicians with more certainty and accuracy during the intra-operative surgery, registering intra-op images in real-time for achieving visual interaction has became popular but challenging problems. Although the outstanding performances of GPU have been studied extensively during the past couple of decades, the non-rigid image registrations had paid more attentions on the more robust and advanced algorithm rather than optimizing it with GPU parallel programming. As the advantages of intra-op imaging in minimally invasion surgeries, we in this work provide an overview of related work on GPU accelerated image registration techniques. This section is organized by firstly presenting related works about GPU based image registration techniques, then focusing on GPU accelerated demons algorithm, which is one of the popular non-rigid image registration techniques.

2.1 GPU Based Image Registration Techniques

As the development of GPU on high computation performance, a wide range of fields has adopted parallel programming for accelerating the independently compute-intensive functions. For example, resolution based real-time robot motion planner utilizes standard graphic hardware for rasterization and gradient-following [2]. Deformable isosurfaces have solved the challenging problems such as expensive computation and parameter tuning reliance by GPU-based 3D level-set method [3]. Flow visualization takes advantages of GPU on texture advection techniques [4]. Image registration has high data-level parallelism where the operations are pixel independent, providing a great potential for accelerating computation speed. The outstanding computation performance of GPU is able to effectively optimize the expensive computation of image registration on CPU and facilitates the real-time interaction for intraoperative surgeries.

In terms of GPU accelerated image registration in medical field, researchers have paid more attention to digitally reconstructed radiograph(DRR) in rigid image registration. Authors of [5] have applied GPU-based 2D/3D intensity-based image registration algorithm on a biplane image and achieved 10 seconds for a completely registration process, which is faster than the previous GPU version algorithm based on cluster method [6]. The computational efficiency is gained from are DRR, cross correlation and image gradient. Patient positioning in radiation treatment has adopted intensity-based registration method to automatically calibrate the patient setup [7]. DRRs generation procedure is accelerated by GPU, which helps with a realtime calibration in order to guarantee the setup accuracy. For image-guided surgery, GPU accelerated DRR generation in preprocessing achieves 1s and is 150x faster than depending on software rendering solely, which is sufficient for intervention surgery undergoing [8]. In [9] and [10], researchers also harness GPU for accelerating fluoroscopy and CT intensity-based image registration for computer assisted surgery(CAS).

Rigid image registration has widely used in issues such as patient positioning and coarse image registration, while it cannot deal with local warps such as misalignment caused by tissue motion. Non-rigid image registration is a particularly needed solution but it also puts heavy burdens on computation, which limits the corresponding applications for intraoperative surgeries. Currently, not sufficient research has paid attention on GPU based non-rigid image registration. An early study was performed in [11]. Their work focuses on non-rigid image registration with 3D Bezier functions and extensively harnesses graphics hardware for improving computation efficiency. The algorithm is applied on MR clinical brain images and it successfully matched the intraoperative images and
preoperative images. However, one drawback of it is that the contrast is poor. This paper validates the feasibility of applying graphics hardware on non-rigid image registration, however, both the accuracy and computation speed of registration were not quantified. Another attempt is [12], where 3D texture mapping of graphics hardware is exploited in order to accelerate the registration process for registering preoperative and intraoperative MR brain images. Similar to [12], authors of [13] have taken advantages of 3D texture capabilities in graphics hardware and accelerated thin plate spline non-rigid image for computed tomography (CT) dataset. The authors compare the accelerated results with software algorithm and it has speeded up to 65 times for a complete registration. Authors of [14] have implemented the entire non-rigid multi-modal image registration process, which exploits mutual information (MI) and the Kullback-Leibler divergence, on GPU. The histogram computation and recursive Gaussian filtering are accelerated and computation time is up to 5x faster than that on CPU. Gradient flow is one of the transformation models in non-rigid image registration. Authors of [15] initially attempt to speed up it on $257 \times 257$ 2D images with graphics hardware. Thereafter the computation time has achieved to 3 seconds. A 3D version of non-rigid gradient flow algorithm is performed by [16] where the entire registration process is executed on GPU, however, the time used for registration is not as fast as expected due to the bottleneck on memory.

2.2 GPU Based Demons Algorithm

Demons algorithm is one of the most popular image registration algorithms at present. In this section, we present previous studies on GPU accelerated demons image registration algorithm.

In GPU accelerated demons algorithm, Gaussian smoothing and intensity interpolation have always been the most computationally prohibitive functions in registration process. There is limited research investigating the influence of different smoothing and interpolation scheme on the computation time. Authors of [17] have applied ramp filter on the displacement field by averaging the six neighbours for each voxel in order to simplify Gaussian smoothing therefore speed up registration. However, approximating smoothing process with the simple ramp filter affects registration accuracy. Gaussian recursive filtering is implemented by [18] for their non-rigid image registration of 3D MR volumes. Gaussian recursive filtering is also an approximate Gaussian filter while it is equivalent to Gaussian filter when the size of the filter is small enough such as $\sigma < 10$, where $\sigma$ denotes the standard deviation of Gaussian filter. In addition, trilinear interpolation implemented by the authors is performed with texture memory on GPU. Compared with the same algorithm executed on CPU, their GPU accelerated demons algorithm has been speeded up 10 times. In [19], Gaussian smoothing is implemented by separable Gaussian filter in order to achieve better algorithm efficiency. One thing to notice is that [19] utilizes the Brook programming environment for GPU implementation. The results is evaluated on a preserved swine lung CT volume and has achieved 70 speed up compared with that on a 2.8GHz Intel processors. Authors of [20] have implemented their GPU based algorithm on 3D CT lung volumes by using Compute Unified Device Architecture (CUDA) programming environment. In the work, separable convolution and self-designed trilinear interpolation is performed in order to accelerate the moving volume deformation. Compared with CPU-based demons algorithm, the result from GPU implementation was speeded up maximally 55x in terms of multi-thread CPU. Because of the better performance by using CUDA, such as powerful features, shared memory access, and a broader set of libraries, authors of [21] have implemented their accelerated demons algorithm on pulmonary CT volumes by using CUDA as well. The convolution kernel is performed by separable Gaussian filters and interpolation kernel is implemented by self-designed trilinear interpolation in
combination with texture memory. One thing to notice is that CUDA-provided trilinear interpolation is available at this time however they did not utilize it in their work due to its insufficient precision.
3 CUDA Implementation

Time-critical intra operation surgeries, such as Adaptive Radiation Therapy (ATR) [22] and image guided surgery [23] [24], require maximum few seconds to complete response in order to achieve real-time visual interactions. However, CPU runtime used for an image registration process ranges between minutes to hours because it is based on a pixel by pixel operation, which is much more expensive while being executed by one thread. Accordingly, image registration based on CPUs cannot be real-time implemented for intra operations unless the computing time can be reduced to seconds. CUDA is a parallel computing architecture, which exploits the parallel computing engine in Nvidia GPUs to speed up tasks and functions that are time consuming on CPUs. GPU accelerated computing is also suitable to be implemented on image registration because its high data-level parallelism. In addition, GPUs parallel processor is more popular than other parallel processors due to the competitive advantages such as low cost and wide availability. Therefore, medical image registration that is embedded with GPU accelerated computing is imperative at present.

In this work, we have employed log-Euclidean approaches [25], which contributes to a stationary velocity field, for permitting large spatial transformation. Our GPU implementation is optimized by using Nsight CUDA profiler, which is installed as a plugin in Visual Studio 2015. With the help of it, the bottleneck and memory leakage can be recognized, the timing result for GPU kernels can be measured and the instruction divergence can be quantified. The results reported in the thesis are generated from the optimized settings according to the profiler.

In this chapter, we firstly introduce CPU and GPU implementation environment of this work. Then, a general scheme for the test data generation is explained by describing how to formulate a Matlab GUI for automatically generating as much test data as needed. Next, we detail the framework of accelerated CUDA kernels. Finally, we talk about other optimisation techniques implemented in this work.

3.1 Implementation Environment

We have implemented our diffeomorphic demons algorithm in C++ on CPU and GPU respectively. Our CPU implementation runs on Windows 8.1 with 4 GHz Intel(R) Core(TM) i7-4790k, which contains 4 cores, 8 threads, 8MB SmartCache and 5 GT/s DMI2 bus speed. Integrated graphic processor is Intel HD Graphics 4600 with 2 GB maximum memory. The CPU execution runs with single thread and is implemented using Visual Studio 2015 community edition.

Our GPU version algorithm is executed on Nvidia’s GPGPU programming environment, CUDA 8.0.61, equipped with Nvidia GeForce GTX 1070. It was released on May 7th, 2016 and has adopted Pascal GPU architecture, containing 1920 cores, 15 streaming multiprocessors(SMs), GDDR5 memory type, 8192MB memory size and 256.3 GB/s memory bandwidth. A warp contains 32 threads and the maximum threads per block is 1024. The maximum block dimension is 1024 * 1024 * 64. The computation capability of GeForce GTX 1070 is 6.1.

3.2 Ground Truth Generation and Matlab GUI

Only one real 3D MRI brain scan is given for this work due to the strict privacy regulations in medical data access. In order to better quantify the performance of our implementations, we firstly
need to generate some test data as the ground truth. A Matlab GUI is implemented for doing this automatically. With it, as much test data as needed can be generated.

The initial idea for ground truth generation is to generate a diffeomorphic deformed image by deforming the given 3D MRI brain scan by transformation parameter recovering, where both the given image and the deformed image are regarded as a set of ground truth. Since simulating a 3D large diffeomorphic transformation is very complex in mathematics, it is not feasible to generate a diffeomorphic deformed image with a self-created diffeomorphic transformation. In the work, we generate a diffeomorphic deformed image with an auxiliary deformed image by following steps.

Given one 3D MRI brain volume, which is the orgImg in figure 1, the ground truth is generated following three steps:

1) Deform orgImg using thin plate spline warping method with ten pairs of randomly generated landmarks. Then the deformed Image tempImg is acquired.
2) Register orgImg and tempImg by using diffeomorphic log demons registration algorithm \[25\] where tempImg is regarded as fixed image A.2 while orgImg is treated as the moving image. Diffeomorphic demons algorithm deforms moving image orgImg towards fixed image tempImg until their cost function A.2 is minimized. Then the deformed image warpImg with diffeomorphic properties is obtained.
3) During the registration, a transformation from orgImg to warpImg can also be acquired. OrgImg, warpImg and the transformation between them are regarded as a set of the ground truth.

Figure 1: Ground truth generation procedure. Original image orgImg is the given 3D brain volume. tempImg is the deformed image from orgImg by using thin plate spline method with randomly generated landmarks. Deformed image warpImg is obtained by deforming orgImg towards tempImg with diffeomorphic demons registration algorithm. Both orgImg and warpImg are regarded as our ground truth.

In order to automatically generate as much test data as needed, we have implemented a graphical user interface(GUI) in Matlab in order to integrate the ground truth generation in a batch processing manner. With this interface, user can load in a DICOM(Digital Imaging and Communications in Medicine) file. With the Matlab GUI, the diffeomorphic deformed image volume can be calculated. When calculation finishes, the user is able to drag the slider in the bottom of Matlab GUI and observe

---

\[1\] The diffeomorphic log demons algorithm in Matlab is used from http://www.mathworks.com/matlabcentral/fileexchange/39194-diffeomorphic-log-demons-image-registration
the visualized deformed 3D volume in slices, which is shown in figure 2. In addition, all the generated data is saved into a corresponding folder for later experiments.

In this work, we have generated five sets of ground truth to test the performance of our implementations, where the pixel resolution for ground truth is 434*362*362. The original image volume is used to generate five sets of ground truth following steps illustrated above, where the diffeomorphic deformed degrees of five datasets are differentiated. Five sets of image volumes are referred as dataset1, dataset2, dataset3, dataset4 and dataset5, whose dimensions are all equal to 209*174*174. Figure 3 utilizes the 70th slice in z direction of each image volume in the ground truth as the representations for 2D visualization.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>174 * 209 * 174</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#Slice_z</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Five sets of diffeomorphic deformed ground truth. The deformed degrees of moving images are differentiated. The 2D images reported in the figure are demonstrated by the 70th slice in z direction of each image volume.

### 3.3 CUDA Kernel Acceleration

In CUDA, functions in the application are expressed as kernels, where the Single Instruction Multiple Data model supports limited instruction divergence. CPU and GPU have their different advantages...
and are used to accomplish an application by combining them together. Therefore, accelerating an algorithm with CUDA is to identify the parallel portions in it, map those to the parallel programming environment and group them into kernels.

In order to accelerate the diffeomorphic demons registration with GPU, it is necessary to review the diffeomorphic demons algorithm framework, breakdown into functions and map the portions that can be paralleled to CUDA programming environment. Figure 4 illustrates the framework of diffeomorphic demons registration. A set of ground truth is utilized as the input for registration, where source image refers to the moving image and target image refers to the fixed image. After initialization, the first iteration of registration is performed. Then the transformation field that is calculated from registration update is exploited in the energy model, which is also referred as cost function, in order to measure the energy of the current deformed moving image. This energy is also used as an input for termination condition, which aims to evaluate whether the deformed moving image in the current iteration is similar enough to the fixed image. If the termination condition 3.5 is not satisfied, the moving image before this iteration will be updated based on the calculated transformation field and keep iterating until the termination condition is satisfied. Output of the framework is a deformed source image that is similar enough to the target image and the corresponding transformation field.

The most essential element of diffeomorphic demons registration is registration update in figure 4, which is also the most time consuming part. As shown in the figure, registration update can be divided into six functions, findupdate, imgaussian, composition, expfield and iminterpolation. Furthermore, these functions mainly call three types of CUDA kernels for the execution, which are gradient kernel, convolution kernel and interpolation kernel. Refer to the diffeomorphic demons algorithm pseudocode that is shown as below, findupdate function is the implementation of line 1, where the gradient kernel is utilized in order to calculate the gradient of fixed image and the current deformed image. imgaussian function is responsible for including Gaussian noise in vector fields, which corresponds to line 2 and line 4 in pseudocode. Convolution kernel is employed in this function. Line 3 is implemented by composition and expfield functions, where the interpolation kernel plays an important role. iminterpolation and energy functions are executed after line 4, where iminterpolation aims to interpolate the deformed image in the current iteration based on that in the previous iteration and the updated transformation field and energy measures the difference between the current deformed image and fixed image.

In diffeomorphic demons algorithm, the most time-consuming part is the exponential of the field. It is computationally expensive because there are masses of complex differential equations need to be
solved. In this work, we have employed an efficient approach for the exponential field computation, which is referred as fast vector field exponentials [26]. This approach only deals with a pseudo-group instead of a strictly Lie group, therefore gains the computational efficiency. In addition, imgaussian is the second time-consuming part in the 3D diffeomorphic algorithm, which has also consumed more than 40% of the total execution time. CUDA kernels that is used for exponential field and imgaussian are convolution and interpolation kernels, which should be mainly accelerated. In the following sections, we primarily illustrate the optimization of convolution and interpolation kernel implementation.

**Diffeomorphic Demons Pseudocode**

1. With the known current transformation \( t \), deduce an update velocity field \( v \) by minimizing the global energy \( E_{corr}^t(t_n) = \arg\min_v \frac{1}{\sigma^2} Sim(I_F, I_M \circ t \circ \exp(v)) + \frac{1}{\sigma^2} dist(t, t \circ \exp(v))^2 \), where \( t_n = t \circ \exp(v) \).
2. Considering Gaussian noise on the update velocity field \( v \) by applying \( v \leftarrow K_{fluid} \ast v \), where \( K_{fluid} \) represents a Gaussian kernel.
3. Update transformation by \( t_n \leftarrow t \circ \exp(v) \).
4. Considering Gaussian noise on the updated transformation field \( t_n \) by applying \( t \leftarrow K_{diff} \ast t_n \), where \( K_{diff} \) represents a Gaussian kernel.

### 3.3.1 Convolution Kernel Acceleration

Convolution kernel is used to perform Gaussian smoothing on velocity field and transformation field. In this work, we have implemented different versions of convolution kernels based on different filter types, block shape and memory types. The best accelerated convolution kernel of those designed models follows three bullet points. First of all, performing separable convolution in order to reduce computation complexity and data reuse times. Secondly, block design should satisfy the half warp requirement in order to coalesce data access in global memory. Finally, utilizing shared memory and texture memory for minimizing the time used for data access.

The intuitive idea is to perform the convolution kernel in two general ways, 3D Gaussian filter and separable Gaussian filter. First of all, we have convolved the image volume with a 3D Gaussian filter, which refers to the red mask in the top left image of figure 5, assuming the Gaussian filter is a square with side length being three pixels. Because of the high parallelism of convolution operation, we regard each pixel as a thread that is executed independently. In the figure, the red mask aims to calculate the convolution of the purple thread, which refers to the pixel at the second row and the second column. A block \( \in \mathbb{R}^{M \times N \times K} \) is consist of a group of threads, which is represented by the blue rectangular in figure 5. Its corresponding shared memory is denoted by \( \mathbb{R}^{(M+2) \times (N+2) \times (K+2)} \), which contains the padded values such as 0 or image intensity of the outer boundary in order to calculate the threads’ convolutions along the block boundary. However, there may be inactive threads in the blocks that are located along the image boundary due to the unpredictable input image size, which is highlighted green. Inactive threads waste the parallelism, which reduces the efficiency and should be avoided as much as possible. Then, we have also implemented separable Gaussian filters, which is the top right image of figure 5. In terms of computation complexity, it is cheaper time consuming by implementing separable Gaussian convolution, which is depicted in figure 6. However, it is expensive in data loading, which means loading data from global memory to shared memory. The schematic of separable filtering is described in the top right of figure 5. 3D Gaussian filter is
Figure 5: 3D convolution with different types of Gaussian filter. Red rectangular represents the filter mask, referred as Gaussian filter, which is used to convolve with the image volume. Blue rectangular denotes block, where threads are grouped and assigned a memory space to share data values in it. Shared memory for the sample block is represented by the yellow rectangular. Top left: 3D Gaussian filter where the block size is self-defined. Top Right: separable Gaussian filter. The filter at each direction is a row. Bottom: separable Gaussian filter. The length of the x-wise Gaussian filter is equal to the image width. The lengths along x direction of both y-wise and z-wise filter are satisfied half-warp requirement, which is equal to 16 pixels.

decomposed into three simple 1D arrays and convolve with the image volume one by one. The blocks designed for x, y and z-wise convolution are all a row, where their lengths are equal to image width, height and depth. The corresponding shared memories are longer than the block where padded values are loaded in order to perform the convolution along the image boundary. Compared with the designed model in top left of figure 5, one of the most important contributions for computation efficiency is that it significantly reduces the data access and reuse. In 3D Gaussian convolution, all pixels except for those along the boundary should involve the convolution computation for its 27 neighbours. While for separable convolution, the reuse time of a single no boundary pixel is only 9, which is significantly less than 3D convolution.

However, it is not the best optimized design for separable filter because the global memory access of this model in y and z direction is rather poor. Take the convolution at y direction as an example, the first 32 threads in a block are regarded as a warp, which will address the consecutive memory addresses in one clock cycle. When loading from global memory to shared memory, the problem is that the consecutive intensity values in a column block are located sparsely in global memory,
which results in strided global memory access. Therefore, it takes more than one memory transactions to finish loading all intensity values in a warp, which deteriorates the computation efficiency. Therefore, we have implemented the third type of convolution function, where the x-wise length of the blocks used for y-wise and z-wise convolution are satisfied the half warp requirement in order to guarantee the coalesced global memory access. This is regarded as the best optimized convolution kernel, which is shown in the bottom of figure 5.

In addition, we have employed shared memory to store the data of image volume and texture memory for storing Gaussian mask. The access speed for GPU global memory is much slower than shared memory and texture memory. Also texture memory access is faster than shared memory access. In our work, we only utilize shared memory for storing image volume because the size of texture memory is very limited with only 8KB for GPU in this work. In addition, texture memory is read-only. However, in separable convolution, y-wise and z-wise convolution calculation base on the x-wise and y-wise convolution result respectively, which means the values of image volume will be updated in the memory after each separable convolution. Therefore, texture memory is not suitable for storing the large image volume intensities.

### 3.3.2 Interpolation Kernel Acceleration

Interpolation kernel aims to map the updated vector field or image back to their original grid. For improving the algorithm efficiency, this work has applied 3D texture object for storing image volume and utilized a CUDA-provided hardware function for trilinear interpolation.

The conduction of trilinear interpolation is shown in figure 7. The purple cube represents an interested point, moving from a circle point towards a triangular point. Circle points represent pixels in the current iteration, whose positions and intensity values are known. Triangular point represents the corresponding pixel in the next iteration, which is also referred to the updated pixel. However, the updated pixel may move to a place that is between the current pixels. Therefore the intensity of the updated pixel may not directly be obtained from the current image and need interpolating. The motion of the purple cube is quantified by the displacement projections relative to the bright purple reference point along x, y and z direction, denoting by $s_x$, $s_y$, $s_z$. However, the triangular point may not situated at the voxel coordinate thus its value needs interpolating from the neighbouring reference points. Intensities from neighbouring points influence interpolated value differently, which depends on their distance and described as weight. The closer to the interpolation point a reference point is, the higher weight it is assigned. For better visualization, we use the coloured cube, which is diagonally opposite its reference point, to describe the weight concept. Therefore, the interpolated
value is the weighted average from eight neighbouring reference points. It is also regarded as the updated value for the known voxel in the next iteration.

Figure 7: Schematic visualization of trilinear interpolation. Circle points denote the image voxels while triangular point represents the interpolation point that may be not situated at the current voxel coordinate. The value of interpolation point is the weighted average from neighbouring reference points, where the weight is denoted by the corresponding coloured cube that is diagonally opposite its reference point.

Interpolation kernel is biggest computation bottleneck in the whole registration because of its massive random memory access. To accelerate this, we have employed 3D texture object for storing the image volume and utilized CUDA-provided hardware function. Although the size of texture memory is very limited, its data fetching still have higher bandwidth than other memory when it is impossible to realize coalescing memory access. In addition, nowadays, CUDA-provided hardware function has conducted many low level optimizations, which improves the interpolation kernel efficiency.

3.4 Other Acceleration Techniques

In addition to the GPU kernel acceleration, we have also implemented other optimization techniques, which is used to reduce the parallel code latency. Furthermore, the framework of GPU accelerated registration is optimized, which is also discussed in this section.

First of all, we have employed float4 data structure on vector fields. Figure 8 is a fragment of the GPU accelerated algorithm, which aims to perform separable convolution for a given a vector field. Before separable convolution, three derivative matrices are calculated along x, y and z directions on the given vector field. Then three derivative matrices are obtained, where the dimensions of them are the same as the original vector field. Next, separable filtering is performed on three derivative matrices respectively. The left image in figure 8 describes the unoptimized float data structure, where separable convolution is performed three times totally. The right image in figure 8 illustrates the optimized float4 data structure. It is a built-in struct with particular alignment, where the length of one instance in float4 structure is equal to that of four instances in float structure. In this example, three derivative matrices are combined into one matrix, where an instance in float4 structure is consist of the corresponding voxels in three matrices and an inactive voxel, which is designed for satisfying the alignment. With float4 structure, separable convolution is only performed once in total. Furthermore, a memory bus can transmit 128 bits or the multiple of that in one clock cycle nowadays. The length of one instance in float4 structure is exactly 128 bits, which can be transmit in only one memory
transaction. Since the corresponding voxels in three derivative matrices execute exactly the same instructions, float4 structure then coalesces the data access and reduces the instruction divergence.

![Figure 8: Data structure. Left: float data structure. Right: float4 data structure.](image)

Secondly, we have avoided unnecessary texture memory creation and destroy for a better efficient performance. Texture memory is read-only, therefore old texture object needs to be destroyed and then new texture object is created when the values in texture memory is updated. The initial approach is to optimize the code framework in order to eliminate unnecessary creation and destroy. In addition, float4 data structure also contributes to avoiding unnecessary texture memory creation and destroy. As shown in figure 8 for float, texture memory creation and destroy is performed three times in total. While for float4 structure, it only performed once.

In addition, the framework of GPU accelerated registration is optimized in order to avoid the redundant function calls. The GPU accelerated diffeomorphic registration is based on the available Matlab code, which is downloaded from File Exchange in MathWorks. However, the Matlab code is unoptimized, where several redundant functions deteriorate the algorithm efficiency. After optimization, the call times of `composition` and `expfield` functions has reduced, which speeds up the registration process.

### 3.5 Termination Condition

Diffeomorphic demons algorithm is behaved in an iterative manner, where energy function contributes to the registration termination. In our code, user defined parameter `max_iter` denotes the maximum number of iterations when the algorithm does not convergence. If the best registration is not achieved during the limited iteration, the registration will still be terminated. The best registration is evaluated based on the energy function, which measure the difference between current deformed image and fixed image. The less the energy is, the more similar the current deformed image and original fixed image. For each iteration, energy of the current deformed image will be calculated and recorded. If the difference between the energy in the current iteration and the maximum energy in the previous four iteration is smaller than a constant, then the registration is converged to the best

---

2 The diffeomorphic log demons algorithm in Matlab is used from http://www.mathworks.com/matlabcentral/fileexchange/39194-diffeomorphic-log-demons-image-registration
registration. The constant is the multiplication of the energy after the first iteration and a parameter stop_criterion. The terminate condition is denoted as follows:

\[
N = \begin{cases} 
  n & \text{if } n < \text{max}_\text{iter} \text{ and } E_{n_{\text{max}}^n-5}n-1 - E_n^n < E_{n}^1 \times \text{stop}_\text{criteria} \\
  \text{max}_\text{iter} & \text{otherwise}
\end{cases}
\]  

where \( N \) denotes the terminated iteration and \( E_n \) represents the energy of deformed image after specific iteration. Figure 9 is an example showing when to terminate the iteration once the registration result is good enough.

Figure 9: Energy function progresses over iteration.
4 Results

This chapter shows the 3D diffeomorphic registration timing results, implemented in Matlab\(^1\) C++ single-thread CPU version and C++ single-thread GPU version. First of all, we focus on the total runtime of diffeomorphic demons registration. Then the algorithm is decomposed into individual functions, such as \texttt{findupdate}, \texttt{imgaussian}, \texttt{composition}, \texttt{iminterpolation}, \texttt{expfield} and \texttt{energy}, and the timing results of them are measured. The function timing results reported are evaluated from 10 iterations of the algorithms in order to obtain the performance results without the effect of random noise. Finally, the results generated from CPU and GPU implementation are discussed.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>174 * 209 * 174</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop Criteria</td>
<td>0.001</td>
</tr>
<tr>
<td>Dataset</td>
<td></td>
</tr>
<tr>
<td>Matlab runtime _single threading (s)</td>
<td>7543.98 7142.61 7601.38 13693.96 2296.00</td>
</tr>
<tr>
<td>C++ CPU runtime _single threading (s)</td>
<td>1414.24 1387.38 1420.32 2571.15 432.42</td>
</tr>
<tr>
<td>C++ GPU runtime (s)</td>
<td>4.88  4.70  5.01  8.64  1.73</td>
</tr>
</tbody>
</table>

Figure 10: Registration runtimes of Matlab, GPU and CPU implementation.

Figure\(^{[10]}\) shows the registration runtime of Matlab, CPU and GPU implementations on five datasets, where the \texttt{stop criteria} is set as 0.001. Both Matlab code and C++ CPU implementation are executed on single thread. From the figure, we can conclude that Matlab code takes the longest time to finish the 3D diffeomorphic registration, which is around thousands of seconds. C++ single threading CPU implementation is approximately 5x faster than Matlab code. This is because Matlab application is based on the scripting language and it is slower than programming language. GPU accelerated diffeomorphic registration has performed the best, which only takes seconds to finish the whole registration. In addition, datasets with different deformation degrees show significantly different performances on the total runtime. As showed in figure\(^{[3]}\) moving image of dataset5 is the most slightly deformed, therefore the total runtime of it is much faster than the other datasets. While, dataset4 has many small but complex local deformations, such as fine wrinkles along the skull. It increases the difficulty of recovering moving image back to the fixed image therefore it takes longer time to finish the registration.

Figure\(^{[11]}\) presents function timing results of Matlab, CPU and GPU implementation by using dataset1. Generally, reported six functions in 3D diffeomorphic registration can be summarized into three GPU kernels, gradient, convolution and interpolation kernel, where the convolution and interpolation kernel has consumed more than 40% of the total execution time respectively. One thing to notice is that the function timing result is not equal to the kernel runtime, which also contains time used for other calculations such as texture object creation and destroy, Jacobian matrix calculation and intermediate parameter calculation. In addition, \texttt{composition} function is also employed in \texttt{expfield} function, which is explained in Appendix\(^{[A.2.2]}\). From the figure, we can conclude that our GPU accelerated 3D diffeomorphic registration has improved the computation speed significantly, especially in functions with convolution and interpolation kernels. For example, \texttt{imgaussian} function has speeded up 2000x faster than that of CPU implementation and \texttt{iminterpolation} function has achieved 900x speedup.

\(^{1}\)The diffeomorphic log demons algorithm in Matlab is used from http://www.mathworks.com/matlabcentral/fileexchange/39194-diffeomorphic-log-demons-image-registration
For the other functions, GPU accelerated version also performs well, improving the computation speed several times to hundreds times faster compared with CPU version.

Figure 12 compares the GPU convolution kernel and CPU convolution function in order to explain why GPU convolution kernel runtime is much faster than CPU convolution function. There are four factors contributing to this significant acceleration. First of all, CPU implementation is single threading while GPU accelerated diffeomorphic registration has 1920 cores to parallel execute theoretically. In fact, the simultaneous active threads on GPU is less than 1920 while it is still far more than single thread, which speeds up GPU code. Secondly, CPU implementation utilizes 3D Gaussian filter while GPU accelerated diffeomorphic registration employs separable filter, which reduces the computation complexity and data reuse therefore significantly improves the algorithm efficiency [3.3]. Thirdly, the data type in CPU implementation exploits float while it utilizes float4 struct in GPU accelerated diffeomorphic registration. Float4 data struct is able to coalesces data access, reduce the instruction divergence and avoid unnecessary texture memory creation and destroy, which improves the computation speed [3.4]. Furthermore, the memory used for storing image volume in CPU implementation is CPU global memory, while in GPU accelerated diffeomorphic registration, we have exploited the combination of shared memory and texture memory. The CPU global memory access is far slower than GPU global memory access and GPU global memory access is far slower than shared memory and texture memory access. Therefore, the timing result of convolution kernel in GPU accelerated diffeomorphic registration is much faster than that of CPU convolution function, which has been accelerated by 2000x.
GPU interpolation kernel has also been accelerated significantly than CPU interpolation function, which mostly thanks to the CUDA-provided hardware interpolation function. Furthermore, the reason why `iminterpolation` function is much faster than other functions using GPU interpolation kernel is because the GPU interpolation kernel is called less times than that in other functions. In addition, `iminterpolation` performs interpolation on a 3D image while other functions performing it on a `float4` vector field, where the size of the data is 3 times than that of a 3D image.

![Figure 13: Comparison between CPU and GPU implementation results.](image)

In this work, we also evaluate the results from CPU implementation and GPU implementation, which is shown in figure 13. The difference between CPU implementation result and GPU implementation result is calculated, mathematically, which is the standard deviation of intensity per voxel. From the table, our CPU and GPU implementation results are equivalent. This is because our CPU implementation and GPU implementation is the same, but we have used GPU and some accelerated tricks in GPU implementation. The errors are mainly caused by the data type precision difference between CPU and GPU implementation.
5 Discussion

In this chapter, we firstly investigate the tradeoff between result accuracy and registration efficiency. Then the limitation of ground truth data will also be discussed.

Accuracy and Efficiency

Result accuracy and registration efficiency is a pair of mutual restraint elements. If a more accurate result is required, the registration should iterate more times until the cost function energy is small enough. However, it absolutely extends the time for registration computation. Therefore, we always need to find a balance between result accuracy and computation efficiency. Figure 14 presents the timing results of five datasets under the different stop criteria. As illustrated in section 3.5, stop criteria is an user defined parameter that controls when to terminate the registration process. The experiment is performed on all the five sets of ground truth and the results, registered images, are presented on the right. The reported result is calculated when stop criteria is equal to 0.001, which is also selected as the empirical best parameter for this system. For better visualization, all 3D image volumes are demonstrated by the 70th slice in z direction. In the left table, #iter denotes the maximum number of iteration when the algorithm is converged. In addition, accuracy is measured by the difference between two images, mathematically, which is calculated from the standard deviation of intensity per voxel. Error between IF and IM quantifies the deformation degree of moving image in voxel with standard deviation while Error between IF and MP measuring the accuracy of resultant image.

From the figure, if the stop criteria is set smaller, it will guarantee a more accurate result. This is because the termination condition becomes stricter when stop criteria is small. Then the deformed moving image will be as similar to the fixed image as possible in order to guarantee the energy of it is smaller enough over iterations. Meanwhile, the time for finishing registration becomes longer as the increasing iterations. If the stop criteria is set relatively larger, the algorithm converges much faster but this will deteriorate the result accuracy. As shown in the figure, image accuracy has achieved a great performance when stop criteria is smaller than 0.001 while it does not improve a lot as the stop criteria decreases. However, the registration time is increasing significantly as the stop criteria decreases. In order to achieve the best performance, we need to balance the algorithm accuracy and efficiency. Therefore, stop criteria being 0.001 are regards as empirical best parameter for this system.

In addition, dataset1 possesses the largest deformation from Error between IF and IM, while its registration time is much less than dataset4, which is only medium deformed. This is because there are many small but complex local deformations in dataset4, such as fine wrinkles along the skull. It increases the difficulty of recovering moving image back to the fixed image therefore it takes longer time to finish the registration.

Test Data Limitation

Test data in this work can not simulate clinical real world data perfectly. As described in section 3.2, all datasets are generated from the same 3D MRI brain image volume, thus they have similar intensity range, same image contrast and same image saturation. However, in clinical applications, patient is possible to be scanned by two different MRI scanning machine for preoperative image and intraoperative image. The problem is that two different MRI scanning machines cannot produce exactly the same images because the strength of their magnetic field cannot be completely the same.
Even for the same machine, it is difficult to produce MRI images with exactly the same image contrast and image saturation in every time scanning. Besides, the contrast of preoperative image and intraoperative image is different because patient is used to be scanned longer time for preoperative image collection. Furthermore, noise is also another interference factor. The problems discussed above are also tricky situations in medical imaging nowadays and there is still no good solution for it.
6 Future work

The future improvements are planned to be conducted in four aspects.

First of all, the throughput of GPU implementation needs improving. There are still instruction divergence and memory bottleneck in GPU accelerated registration. We would like to keep improving on it.

Secondly, a multiple-scale scheme for the whole algorithm will be implemented. In this work, the pixel resolution of the test data is 174*209*174, which is a computationally expensive resolution to find the solution. Multiple-scale scheme aims to find an approximate transformation solution with down-sampled input image volumes. Thereafter, low resolution input image volumes are up-sampled to a higher resolution and the coarse solution from the previous low level resolution will be refined. The process stops when the refined transformation from the original resolution image volumes is obtained. Multiple-scale scheme is able to improve the algorithm efficiency because the computations at coarse resolution are less expensive than fine resolution.

Thirdly, a 2D diffeomorphic demons registration will be conducted. This is because, nowadays, only 2D image registration is utilized for performing image registration in the clinical usage. In this work, GPU accelerated 3D diffeomorphic registration has achieved less than 10 seconds to register a relatively large deformed image volume, whose dimension is 174*209*174. Therefore, it has a great potential to achieve real-time performance on 2D image registration.

Finally, we plan to contribute to the community with an open source implementation that can facilitate the research in the field. Note that the code repository will be first hosted privately on github.com, and based upon the legal matters of the host university of this project (The University of Hong Kong), we will consider open source the repository on an agreed time under certain license.
References


Appendix A  Literature Review

A.1  Image-guided Intervention

Image-guided intervention (IGI) or minimally-invasive surgery and therapy, is a technique used in the clinical surgeries, which integrates different but related patient data inputs into a common interface, represents either the operating patient’s anatomy structure or the therapeutic and surgical environment in order to guide the interventionist to the treatment target accurately and achieve the minimally invasion. A number of imaging techniques are included in IGI such as image processing, visualization, augmented reality and image registration. Nowadays, it has mainly contributed neurology, abdomen and orthopaedics.

A.1.1  Overview and modern IGI systems

Although the modern embodiment of the IGI field is over 30 years, its concept of assisting surgeries with the help of external devices or anatomical medical images and videos had been developed for more than one hundred years [27]. The prototype of IGI concepts and methods is the stereotaxy in neurosurgery, where an external device is used to define a coordinate of an anatomical structure and guide the surgical instrument to the target inside it. One famous example is Horsley and Clark stereotactic frame [28], which was affixed to a monkey’s head and allowed electrodes introduced to the target location inside the head skull. This methodology had been further developed, tested and refined for decades and then human stereotactic frames [29] had been emerging, such as Leksell frame [30] [31] and Todd–Wells frame [32]. The rise of IGI was started from removing frames in stereotaxy. For example, [33] [34] was the first published frameless stereotactic” systems, which attaches a tomographic image to the microscope. Then a number of pioneering systems combining with an articulated arm had been integrated to the image-guided surgeries [35] [36] [37] [38]. Later, image-guided technique has been evolved into projecting a space to another.

The modern image-guided intervention has been developed over two stages. The conventional system is to project a pointer or an surgical instruments into medical data set with respect to the virtual image of the therapeutic environment. Surgeons observe a separated workstation, such as a monitor, where the projection of instruments on the anatomy is displayed and track the surgical tools with the guidance of it. However, operating in a virtual surgical space in a monitor remote from patients instead of in the actual patient anatomy divides surgeons’ attention and causes a psychophysical decoupling. Recent image-guided intervention has adopted augmented reality (AR), which maps the preoperative or intraoperative patient data input directly on the 3D context of patient anatomy during the surgery. Therefore inside operative site can be perceived on the patient, which facilitates surgeons’ hand-eye coordination.

A.1.2  IGI Process

IGI is implemented by the following five processes:
1. Preoperative data collection and surgical planning.
   The preoperative data is generally collected in tomography images, which provides surgeons with patient-specifically anatomical information in order to help with planning the treatment procedure.

2. Surgical instruments or devices positioning and tracking.
   In conventional IGI system, the position of surgical instrument will be tracked based on the relationship compared with the surgical scenarios.

3. Preoperative data and surgical scenario registration.
   In conventional IGI system, the key structures in the preoperative data is rigidly registered with the virtual image of surgical scenario while it is projected directly on the 3D context of surgical scenario in IGI system with AR technique.
Registration in IGI consists of rigid registration and non-rigid registration. Rigid registration is widely used either as itself or as an preliminary step for non-rigid image registration. However, it cannot response timely for physiological motion or tool tissue interaction, such as brain shift due to the loss of cerebrospinal fluid after craniotomy in robot assisted neurosurgery and repeated pumping motion of the heart in the robot assisted cardiac physio electrotherapy, which is also one of the disadvantages for conventional IGI system. Non-rigid registration solves this problem by finding the correspondences between preoperative data and intraoperative data. During the intervention, non-rigid registration contributes to not only match the surgical plan and actual patient coordinates in terms of soft tissues operation but also deform the plan and track motion when there is a motion for the surgical target.

In non-rigid image registration, there are two methods that are widely used, feature-based algorithm and intensity-based algorithm. Feature-based registration algorithms require prior object identification, which aims to identify and match correspondingly distinctive features between two images in order to minimize the feature distance measure. Feature detection as a preprocessing procedure is significantly important in feature-based registration algorithm, which contains ridges and crest lines detection [39] and contour orientation detection [40] [41]. Intensity-based registration algorithms rely directly on voxel intensities, which aims to optimize not only the similarity of intensity distribution but also the transformation fitness [42]. The transformation can be modelled by viscous fluid motion [42], B-spline [43] and thin plate spline functions [44] Demons algorithm is one kind of fluid based intensity-based registration algorithm [45] [46] where the transformation is modelled by the optical flow. Feature-based registration algorithm is faster than intensity-based registration algorithm while the accuracy of prior object recognition may affect the the performance. The evaluation for image registration involves accuracy, reliability and computation time.

4. Visualization.

Conventional IGI system visualizes the preoperative data and surgical scenario in a workstation that is separable to patients. However, for IGI system with AR technique, the necessary anatomical structures in preoperative data are visualized on the surface of surgical scenario. Visualization aims to navigate the interventionists to localize the interested target in order to enhance the accuracy of the surgery.

5. Motion of surgical instrument planning(if necessary).

As described in process 3, in some cases, the motion of surgical instrument can be scheduled with non-rigid registration technique during the surgery.

A.2 Image Registration

As described in the previous section, image registration is essential for robot assisted image-guided surgeries, where the accuracy and computation speed of registration is in a high demand. Therefore, in this section, we present the general background knowledge of image registration in order to lay a foundation for further analysis.

Given a fixed image \( I_F \in \mathbb{R}^{m \times n} \) and a moving image \( I_M \in \mathbb{R}^{m \times n} \), where \( m \) and \( n \) denotes the image height and width respectively, with geometrical objects representing similar context. In order to align these two images, it makes sense to identify the feature points such as corners, distinctive texture and colors in two images, thereafter, match them as much as possible by pushing features in \( I_M \) to the corresponding location of that in \( I_F \). Consider a registration of hand image in Figure 15. The feature points are commonly selected from fingertips and interdigital folds, which are marked with bright colors. A primary step for registration is to match the corresponding features as much as possible. However, except for the easily recognized feature points, the whole contours and indistinguishable flat points are expected to be in correspondence with that in the similar position of the corresponding finger as well. Therefore, image registration is to find a spatial transformation \( T \), which aligns corresponding points in \( I_F \) and \( I_M \) as much as possible. More specifically, deform the moving image \( I_M \) by a transform \( T \) in order to let \( T \circ I_M \) approach to \( I_F \) as much as possible, where \( T \circ I_M \) can be regarded as the deformed image. The spatial transformation can be also denoted by \( T = T(I_M, I_F) \).
Mathematically, image registration aims at minimizing a dissimilarity between the fixed images $I_F$ and deformed moving image $T \circ I_M$, which is denoted by:

$$T_H(I_M, I_F) = \arg \min_{T \in H} \text{reg}(T) + \lambda \ast \text{dis}(I_F, T \circ I_M)$$  \hspace{1cm} (2)$$

where $H \in H_A$ is a set of spatial transformations and the image registration process is to go through all transformations in $H$. $\text{dis}(I_F, T \circ I_M)$ is the dissimilarity measure and $\text{reg}(T)$ is the regularization term. $\lambda$ is a tradeoff factor that balances both terms.

Dissimilarity measure quantifies how similar the points are of a geometrical object with respect to their corresponding points in another geometrical object [48], which is usually defined by the sum-squared intensity difference between $I_F$ and $I_M$:

$$\text{dis}(I_F, T \circ I_M) = ||I_F - T \circ I_M||^2$$ \hspace{1cm} (3)$$

where $|| \cdot ||$ is the Euclidean norm. However, this measure is not validate for all applications. For example, it performs poorly when the two images are from different image modality. An alternative dissimilarity measure term for different modality image is to maximize the mutual information [49].

Regularization term aims to measure the degree of deformation and wrinkle in ambient space. Weird and abnormal transformation $T$ possess higher regularity value, which is more possible to be discarded during the registration process.

Image registration algorithms can be categorized into two classes based on the type of the transformation model. The first class is rigid transformation, which refers to register images with rigid transformation such as translation, rotation and affine. The parameters in the transformation are explicit and it is generally used as the pre-processing before non-rigid transformation in order to reduce the non-rigidly aligning time for images. In addition, it also plays an important role in positioning patient before the radiation therapy for maximal exposing lesions to radiation while protecting the healthy tissue to be minimally affected. For example, rigid registration for brain tumour patient before the radiation therapy [20]. However, the rigid transformations are insufficient for more complex deformation models such as elastic deformation since it lacks degrees of freedom. Non-rigid transformation models are able to warp the moving image $I_M$ locally in order to best align with the fixed image $I_F$. There are two main methods of non-rigidly registration, feature-based and intensity based registration algorithm. Feature-based image registration identifies the image features while intensity-based image registration algorithm depends on the pixel/voxel values in order to register the images. Our work focuses on diffeomorphic demons image registration algorithm, which is one of the non-rigid intensity-based registration algorithm, for MRI intra operative image registration.
A.2.1 Demons Algorithm

Demons algorithm, proposed by Thirion [45], is one of the most popular non-parametric non-rigid image registration methods for the same imaging modality. It exploited Maxwell’s demon concepts in the system, which is able to push pixels from moving image towards the corresponding location in the fixed image following an iterative approach. The forces that cause the pixels’ movement, which refers to the displacement field, is based on optical flow method. The iterative method alternates between the computation and regularization of the displacement field until convergence, where registered image is deformed resemble to fixed image as much as possible.

In the following sections, we will illustrate the basic concepts in demons algorithm and its general scheme.

Maxwell’s Demons

In the 19th century, the physicist James Clerk Maxwell proposed the concept of demons, which is in contradiction with the Second Law of Thermodynamics. Figure 16 describes the experiment. Given two chambers $C^a$ and $C^b$ of gas that composed of two types of gas molecules $m^a$ and $m^b$, and a semi-permeable membrane that is located between chambers. Assume there are set of effectors located on the membrane, which called ‘demons’. When a gas molecule approach to the membrane, demons can distinguish the type of it and only allow the same type of particles to spread into the same chamber. For example, $m^a$ are only allowed to spread into $C^a$ and $m^b$ are only allowed to spread into $C^b$. In the end, the same type of gas molecules will remain in the same chamber.

Demons illustrates the paradox of the Second Law of Thermodynamics since the entropy of the system increases in order to distinguish the type of gas molecules.

Figure 16: Maxwell’s demons model.

Diffusion Model

Given a fixed image $I_F$ and moving image $I_M$ which are defined in section A.2, the basic idea of the diffusion model is to assume the object boundaries in $I_F$ is semi-permeable with effectors called demons situated on and consider $I_M$ as a deformable grid. The deformable grid of $I_M$ is able to diffuse through the semi-permeable boundaries in $I_F$, influenced by demons. We regard each contour point in fixed image $I_F$ as a demons with a vector orthogonal to this contour. The orientation of demons vector can be either from outside to inside or from inside to outside, which is commonly denoted by the image gradient. The pixels situated along the same gradient line will be assigned an identical orientation. In the deformable grid of the moving image $I_M$, the vertices can be labelled as either ‘inside’ or ‘outside’ according to the orientation of the correspond point in $I_F$. Therefore, an informal definition for demons is that: if the point in moving image $I_M$ is marked as ‘outside’, the corresponding demons in fixed image $I_F$ will pull $I_M$ outside of the contour of $I_F$. Vice versa. In addition, the diffusion process is also equivalent to apply a Gaussian filter with $\sigma$ variance [50].

Except for the diffusion, deformable model with attraction is also a frequently used non-rigid image matching method. For example, an arbitrary point $p^M$ in the moving image $I_M$ is attracted by all points $p^F$ in fixed image $I_F$. The attraction force for $p^M$ is denoted by:

$$ f^M(p^M) = \sum_{p^F \in I_F} \frac{S(p^M, p^F)}{D(p^M, p^F)} p^M - p^F $$

(4)
where $S(p^M, p^F)$ refers to the similarity term while $D(p^M, p^F)$ is a metric of distance. For simplification, the point $p^M$ can be regarded as only being attracted by the point $p^F$ that is 'closest and most similar' to $p^M$. As the most essential concept in attraction, similarity and distance determine the forces points to the closest point.

In demons algorithm, the forces alternates between attraction and diffusion, which results in the efficiency of demons algorithm compared with other registration method, to name a few, which based on linear plasticity. In addition, both methods above are iterative. At each step, all valid forces are applied on the object in $I_M$, which results in the pixel motion and intensity flow.

Algorithm A fixed image $I_F$ and a moving image $I_M$ have been defined in section A.2. Image registration requires an iterative process, during which each pixel in $I_M$ is moved by a small displacement. The goal of registration is to form a transformation $T$ after all iterations, making deformed moving image $T \circ I_M$ approach to $I_F$ as much as possible. In non-parametric spatial transformation, the transformation is represented by the displacement filed $t$ in one iteration. More specifically, at each step, the pixels $p^M$ in $I_M$ are updated by adding a small displacement field $s$ to itself, which is denoted by:

$$t : p^M \leftarrow p^M + t'(p^M)$$

where $t'(p^M)$ refers to transform of all the points $p^M$ in $I_M$ at each iteration, which also called update field $t'$. $t$ refers to the updated transformation field. The transformation $t'(p^M)$ represents to move $p^M$ with a displacement $t'$. Therefore, The expected transformation $T$ is a family of the update filed $t'$ during the iteration, which is denoted by

$$T = t_n = \{t'_1, t'_2, \ldots, t'_n\}$$

where $n$ is the number of iteration.

As mentioned in section A.2 registration process is to minimize the dissimilarity between the fixed images $I_F$ and deformed moving image $T \circ I_M$. In this section, we call this optimized term as global energy, which is represented as follows:

$$E(t) = \arg\min_t \frac{1}{\sigma^2_t} \text{Sim}(I_F, I_M \circ t) + \frac{1}{\sigma^2_t} \text{Reg}(t)$$
where $\sigma_i$ describes the noise on image intensity and $\sigma_T$ determines the degree of regularization we expected. $\text{Sim}(I_F, I_M \circ t)$ is the metric of dissimilarity and $\text{Reg}(s)$ depicts the regularization term. In demons algorithm, the similarity criterion is defined as:

$$\text{Sim}(I_F, T \circ I_M) = \frac{1}{2}||I_F - T \circ I_M||^2$$

(8)

[51] optimized the global energy [7] by considering the continuous and smoothness of the transformation $t$ and supplementing a hidden variable, correspondence. With the idea, a non-parametric spatial transformation $t_n$ is introduced, $t$ can be regarded as the idealization form of $t_n$, which does not allow errors at image point such as noise. Therefore, after applying with Gaussian noise on displacement $t$, the global energy at each step is:

$$E(t_n, t) = \arg\min_{t_n,t} \frac{1}{\sigma_i} \text{Sim}(I_F, I_M \circ t_n) + \frac{1}{\sigma_d} \text{dist}(t, t_n)^2 + \frac{1}{\sigma_T} \text{Reg}(t)$$

(9)

where $\text{dist}(t, t_n)^2 = ||t_n - t||^2$, $\text{Reg}(t) = ||\nabla t||^2$ and $\sigma_e$ represents the uncertainty on the correspondences. In addition, the correspondence in reflected by the term $\text{dist}(t, t_n)^2$.

The key concept behind demons algorithm is alternate optimization over $t_n$ and $t$. It decomposes the complex minimization process $\arg\min$ into two simple steps $\arg\min$ and $\arg\min$, which is described as follows:

1. $E_{t_n}^{corr}(t_n) = \arg\min_{t_n} \frac{1}{\sigma_i} \text{Sim}(I_F, I_M \circ t_n) + \frac{1}{\sigma_d} \text{dist}(t, t_n)^2$
2. $E_{t_n}^{corr}(t) = \arg\min_t \frac{1}{\sigma_d} \text{dist}(t, t_n)^2 + \frac{1}{\sigma_T} \text{Reg}(t)$

Let’s take the additive demons algorithm as an example. The following iterations are processed until convergence:

**Additive Demons Algorithm**

1. With the known current transformation $t$, deduce an update field $t’$ by minimizing the global energy $E_{t’n}^{corr}(t_n) = \arg\min_{t’n} \frac{1}{\sigma_i} \text{Sim}(I_F, I_M \circ (t + t’)) + \frac{1}{\sigma_d} \text{dist}(t, t + t’)^2$, where $t_n = t’ + t$.
2. Considering Gaussian noise on the update transformation field $t’$ by applying $t’ \leftarrow K_{fluid} * t’$, where $K_{fluid}$ represents a Gaussian kernel.
3. Update transformation by $t_n \leftarrow t + t’$.
4. Considering Gaussian noise on the updated transformation field $t_n$ by applying $t \leftarrow K_{diff} * t_n$, where $K_{diff}$ represents a Gaussian kernel.

**A.2.2 Diffeomorphic Demons Algorithm**

Although demons algorithm is efficient, its disadvantages are obvious as well. One of it is the inversion of the output transformation $T$ cannot be guaranteed thereafter folding will be introduced into the transformation. The main reason is that the topology of demons deformation framework is not necessarily preserved although it preserves when the displacement is small. Diffeomorphic demons necessarily preserves the transformation topology. It adapts the global energy [9] optimization to diffeomorphism space in order to ensure the transformation is diffeomorphic. The input data can be generated with diffeomorphic demons algorithm since its transformation is invertible.

**Inversion and Composition**

The inversion of a registration algorithm means that the input image can be reproduced by performing an inverse transform on the deformed image. An invertible transform is used to describe the smooth and continuous mapping without folding. In other words, the topology of the transform is strictly preserved during the transformation.
In order to preserve the topology, many image registration method tries to limit the step of displacement at each iteration. However, [52] points out those method is not such reliable even when the displacement is small. Diffeomorphic demons algorithm used the Lie group structure on diffeomorphic transformation in [53] in order to ensure the transformation is diffeomorphism. Diffeomorphism refers to a gradient invertible, one to one mapping without discontinuity and non-smooth.

Composition is a fundamental concept in image registration, which pointwise passes the result of one function as the argument of another in order to produce a new function. In image registration, it describes the iterative procedure of forming expected transformation $T$. As illustrated in A.2.1 the operation of generating an expected transformation $T$ is equivalent to the composition of discrete finite update fields. Assume the number of iteration is $n$, then the deformed image is represented by $t'_n \cdots t'_2 \circ t'_1 \circ I_M$ and the expected transform $T$ is the composition of update fields in an iterative manner $t'_n \cdots t'_2 \circ t'_1$. Therefore, $\circ$ is defined as the composition operation, which is described as follows:

$$t'_n \cdots t'_2 \circ t'_1 \circ I_M = t'_n(\cdots t'_2(t'_1(I_M)))$$ (10)

The consistency of composition is essential for diffeomorphism transformation. If the update field $t'$ is diffeomorphism, the composition of them such as expected transform $T = t'_n \cdots t'_2 \circ t'_1$ is diffeomorphism as well.

One of the advantages of composition consistency is that even large deformation can be necessarily valid under the diffeomorphic transformation.

In the paper [52], the inversion and composition is investigated by establishing a transform $I + u(x)$ and inverse transform $I - u(x)$ firstly, then composing them in order to attempt to regenerated the identity transform $I$, which is described in Figure 18 and 19. It can be concluded that small deformation setting is not reliable for yielding a diffeomorphic transformation while the diffeomorphism does.

Figure 18: Inversion and composition operation of small deformation. Top left: deform an identity transform $I$ with a displacement $u(x)$. Top right: inverse transform by subtracting the displacement $u(x)$ from identity matrix. Bottom: an attempt to reproduce the identity transform $I$ by composing the transform and inverse transform. Reproduced from [52] with permission from Elsevier.

Newton Method

In diffeomorphic image registration algorithm, differential equation is established to model the process that moving image $I_M$ flows to align the shape of the fixed image $I_F$. [54] and [55] illustrates the greedy ‘viscous fluid’ registration approach, which is fundamental for the development of diffeomorphic image registration.
algorithm in the early stage. In the method, finite difference is used to solve the differential equation however it is computationally expensive. Since then, increasing methods for solving the equation in a faster manner has appeared, such as the use of Fourier transform [56]. Diffeomorphic Anatomical Registration using Exponentiated Lie algebra (DARTEL) [52], which is one of efficient diffeomorphic registration algorithms evolving Newton method, is able to register images with invertible and fast computed diffeomorphic transformation. In this section, we take it as an example to illustrate the application of newton method on diffeomorphic registration.

Newton Method for Lie Groups

The main idea of the Newton methods for Lie groups is to find an update velocity field \( \mathbf{v} \) based on minimizing the global energy with the current transformation \( t \) known at each iteration. It computes an unconstrained update \( \mathbf{v} \) on the Lie Algebra then projects it back to the Lie group through the exponential map, which yields the equation:

\[
t \leftarrow t \circ \exp(\mathbf{v})
\]  \hspace{1cm} (11)

Figure 19: Inversion and composition operation of diffeomorphic deformation. Top left: deform an identity transform \( I \) with a diffeomorphic displacement \( u(x) \). Top right: inverse transform by subtracting the diffeomorphic displacement \( u(x) \) from identity matrix. Bottom: an attempt to reproduce the identity transform \( I \) by composing the transform and inverse transform. Reproduced from [52] with permission from Elsevier.

Figure 20: Lie group geometry. Reproduced from [26] with permission from Elsevier.
As mentioned above, the registration process can be viewed as a flow motion from $I_M$ to $I_F$. When a point flows by different locations in the image, it will be assigned different velocities, which relates to the deformation at that location. At any time $t$, the velocities of all points in the image is called a velocity field at $t$ $v(t)$. The flow motion can be discretized into unit time intervals, during which the velocity field $v$ is assumed to be constant. In algorithm, the time interval is processed in an iterative manner. Thereafter the diffeomorphism is evolved by the velocity field $v$ during time interval $t$:

$$\frac{d\phi}{dt} = v(t)\phi(t)$$  \hspace{1cm} (12)

where $\phi$ is the update displacement field, which is equivalent to $t'$ in the section [A.2.1]. In order to distinguish it with time $t$, we use $\phi$ in this section to avoid confusion. Then the diffeomorphism in a unit time interval, $\phi$, is obtained by integrating the velocity field $v(t)\phi(t)$ over unit time. Initially, the displacement field $\phi(0)$ is an identity transform $I$.

Newton method provides a simpler approach to solve the differential equation (12):

$$\phi(t+h) = \phi(t) + hv(\phi(t)) = (I + hv) \circ \phi(t)$$  \hspace{1cm} (13)

where $h$ refers to successive time intervals in unit time. With Newton method, the diffeomorphic spatial displacement field during unit time can be calculated as follows:

$$\phi(1/8) = I + \frac{v(1)}{8}$$  \hspace{1cm} (14)

$$\phi(1/4) = \phi(1/8) \circ \phi(1/8)$$  \hspace{1cm} (15)

$$\phi(1/2) = \phi(1/4) \circ \phi(1/4)$$  \hspace{1cm} (16)

$$\phi(1) = \phi(1/2) \circ \phi(1/2)$$  \hspace{1cm} (17)

And the inverse of the diffeomorphic spatial displacement field during unit time is:

$$\phi(-1/8) = I - \frac{v(1)}{8}$$  \hspace{1cm} (18)

$$\phi(-1/4) = \phi(-1/8) \circ \phi(-1/8)$$  \hspace{1cm} (19)

$$\phi(-1/2) = \phi(-1/4) \circ \phi(-1/4)$$  \hspace{1cm} (20)

$$\phi(-1) = \phi(-1/2) \circ \phi(-1/2)$$  \hspace{1cm} (21)

As mentioned above, the velocity field $v$ is exponentiated to generate a spatial transformation, which is denoted by:

$$\phi(1) = \exp(v)$$  \hspace{1cm} (22)

Therefore, $\phi(1)$ can be regarded as a fast computation for $\exp(v)$ during the diffeomorphic image registration process.

**Algorithm**

Diffeomorphic demons algorithm is an extension of demons algorithm. Unlike demons algorithm using a displacement filed, diffeomorphic demons algorithm exploits exponentiating velocity field to update the transformation during iterations. Such update is caused by a “force” which results from image gradient and the image difference. There are several forces that is used for updating, which are passive force, active force, double force and so on. In this work, we have employed log-Euclidean approaches [25], which contributes to a stationary velocity field, for permitting large spatial transformation. The pseudocode is shown as below:
Diffeomorphic Demons Algorithm

1. With the known current transformation \( t \), deduce an update velocity field \( v \) by minimizing the global energy
\[
E_{corr}^{t}(t_{n}) = \arg\min_{v} \frac{1}{\sigma^2} \text{Sim}(I_{F}, I_{M} \circ t \circ \exp(v)) + \frac{1}{\sigma^2} \text{dist}(t, t \circ \exp(v))^2,
\]
where \( t_{n} = t \circ \exp(v) \).
2. Considering Gaussian noise on the update velocity field \( v \) by applying \( v \leftarrow K_{\text{fluid}} \ast v \), where \( K_{\text{fluid}} \) represents a Gaussian kernel.
3. Update transformation by \( t_{n} \leftarrow t \circ \exp(v) \).
4. Considering Gaussian noise on the updated transformation field \( t_{n} \) by applying \( t \leftarrow K_{\text{diff}} \ast t_{n} \), where \( K_{\text{diff}} \) represents a Gaussian kernel.

A.3 Graphic Processing Unit

In addition to the certainty and accuracy of image registration, computation speed is also significantly important in many robot assisted minimally invasive surgeries where real time feedbacks are required. Generally, it requires maximum few seconds to complete response in order to achieve real-time visual interactions during intra-operative surgery. However, Central Processing Unit (CPUs) runtime used for an image registration process ranges between minutes to hours because it is based on a pixel by pixel operation, which is very expensive. Accordingly, image registration based on CPUs cannot be real-time implemented for intra operations unless the computing time can be reduced to seconds. As the development of GPUs, time-consuming functions can be accelerated with it if the threads in the functions can be executed independently. In this section, we introduce the current situation of CPUs and Graphic Processing Unit (GPUs) by illustrating their architecture firstly. Then we discuss their capabilities in terms of specific applications.

A.3.1 Central Processing Unit and Graphic Processing Unit

Except for CPU, all personal computers have configured much slower and cheaper Integrated Graphics Processor (IGP) to display basic visual applications such as low-resolution games. IGP is a kind of Graphic Processing Unit (GPU), which are integrated into the motherboard directly. A separate GPU, which was popularized by Nvidia, is far more powerful than IGP and originally invented for 3D game rendering. It refers to the efficient graphics and image processors and nowadays, are more widely used to accelerate computationally intensive tasks such as deep learning, financial modelling, climate modelling and intra-operation image registration. Recent years, the excellent performance-price ratio and outstanding computation efficiency of GPU have showed its significant advantages for exploding the amount of data in a timely manner compared with CPU. GPU goes mainstream for now.

Architecture Comparison

Next, we will explain the differences between CPU and GPU by introducing how they handle tasks. As shown in Figure 21 CPU consists of up to eight bigger cores while GPU has thousands of smaller cores. Although the number of cores in CPU is far less than that in GPU, its control hardware is much more complex, which allows flexibility in performance for a single complex task. However, blindly increasing the number of functional calculating units in order to obtain the maximum performance from a stream of instructions is useless because most of the time they are unused. In addition, the complicated control hardware is costly in terms of power and design complexity. In contrary, GPU has simpler but power efficient control structure for each smaller core. It contributes to support more independent computations in the data path based on many simple cores at the same time. In other words, A large number of GPU’s calculation units are actually used in a parallel manner. To the end, CPU aims to shorten the time on each step of a computation with a few very powerful processors, while GPU processes parallel stream of data by harnessing a large amount of weaker processors to reduce the time for the whole execution.

Latency and Throughput
As mentioned above, both CPU and GPU aim to achieve power-efficient processors on their own specialized processing capabilities. Moreover, the differences of their processing capabilities results from the different optimized targets. CPU contributes to optimize the latency while GPU aims for throughput. More specifically, latency represents the amount of time to compute a task, which is measured in units of time while throughput means the tasks completed per unit of time, which is measured in unit as stuff per time. The powerful control hardware for CPU processors enables it to reduce the time spend on each single task, which minimize the latency. Although GPU processors are less functional than CPU, the parallel computing capability of it allows massive tasks to complete per unit of time, which maximize the throughput.

**GPU-accelerated Computing**

Although GPU is able to execute hundreds times more instructions than CPU per clock, it is ill-suited to complex processing especially for non-parallel tasks. In other words, both CPU and GPU are indispensable in data processing and they are suitable to different applications according to their own architectures. CPU is regarded as an executive, which performs well on a few complex streams of data, while GPU is a labourer that is good at handling a large amount of streams of data efficiently and simultaneously. GPU-accelerated computing is not implemented on GPU only. For a specific application, the serial code still runs on CPU, while for the computationally intensive functions in the code, they will load to GPU for parallel computation. Figure 22 illustrate this procedure.

**A.4 Compute Unified Device Architecture**

Compute unified device architecture(CUDA) is an application programming interface(API) that allows to efficiently run high-performance parallel computing, which is introduced by an American technology company Nvidia. It is a set of software layers in responsible to communicate with GPU, more specifically, accessing to
the GPU’s virtual instruction set and computational elements in order to execute the compute kernels. With CUDA-enabled GPU such as Nvidia GeForce 8 and its updated version and the latest Quadro GPU, software engineers are able to handle computationally intensive tasks through for general purposes. The approach that process computation with the use of GPU is termed General-Purpose Computing on Graphics Processing Unit (GPGPU) and CUDA is one of these platforms. CUDA also supports other programming framework such as Microsoft’s DirectCompute, Open ACC and Khronos Group’s OpenCL. The industry-standard programming languages on CUDA platform are C, C++ and Fortran, which does not require graphics programming skills and is easier to access GPU resources. CUDA C/C++ is provided for C/C++ programmers, which is compiled with Nvidia’s LLVM-based C/C++ compiler while CUDA Fortran is for Fortran programmers, which is compiled with the PGI CUDA Fortran compiler. In addition, Matlab, Python, Java, R and Perl is available on CUDA platform through the third party wrappers. In our work, we use CUDA platform with C++ to achieve diffeomorphic demons algorithm and the GPU we have is Nvidia GeForce GTX 750 Ti.

A.4.1 Heterogeneous Programming

Heterogeneous Programming stands for systems applying multiple kinds of processors or cores in order to achieve performance efficiency. CUDA programming model is a heterogeneous model since it allows to program two different processors, CPU and GPU, in a one program. Processors in CPU and coprocessors in GPU process particular portion of the task by incorporating specialized processing capabilities [57]. More specifically, the CUDA compiler will compile the program into pieces, the serial computation is executed on CPU that refers to the host, while the computationally intensive code is running on GPU that is called device. A.3.1 explains the work-flow flow for a GPU-accelerated computation in details. Host operating environment allows to read and write files, allocate memory, use external libraries, launch kernel and copy data to/from the GPU’s sub-program. While device operating environment has its separated memory and computing cores, which is able to handles the process transferred from host. In the model, host and device maintain their own physical dedicated memory in the form of DRAM to store data, which is located in CPU and GPU separately. Typically, GPU’s memory is a very high performance block of memory. The sequential operations of a CUDA programs is: firstly declare and allocate host and device memory and initialize the host data by serial code; secondly, load data from host memory to device memory ; thirdly, launch kernels on GPU and compute in parallel; finally, transfer results from device memory back to the host memory. Figure 23 illustrates the data transfer between the host and device.

![Figure 23: Data transfer between host and device.](image)

A.4.2 GPU Thread Hierarchy

Tread refers to the smallest sequence of programmed instructions managed by a scheduler independently in a operating system. In a single processor system, different thread executes chronologically according to CPU’s instruction, which is in a similar manner as the serial computation executed on CPU. While multiprocessors system allows multiple threads to execute independently and in parallel, which resembles the sub-programs with computational intensive functions loaded to GPU [A.4.1]. Specifically, thread is defined as one independent path of execution through the code. Each of the kernels that is the sub-programs transferred to GPU for parallel computation, is usually executed by hundreds to thousands of threads. GPU is good at not only efficiently launching a large number of threads but also running the threads in parallel.
A group of threads executed in parallel refers to a thread block, which is better for processing and data mapping. Since a thread is represented by a 3-component vector in GPU, a thread block formed by threads can be organized into one-dimension, two-dimension or three-dimension. However, the number of threads in one block has a limitation, which is no more than 1024(one-dimension), 1024*1024(two dimension) or 1024*1024*64(three-dimension) for the available GPU in this work, Nvidia GeForce GTX 750 Ti. This is because all threads in one block are expected to run on the same streaming multiprocessors(SMs) on the hardware and the size of SM has a limitation. Each thread and block have a build-in index, with which the parameters corresponding to the computation in this block can be selected correctly. A kernel launched on GPU can be executed by many thread blocks with the same size. Moreover, a group of blocks organized in one-dimension, two-dimension or three-dimension can form a grid. The size of grid and thread block is indispensable when launch a kernel on GPU, which is mentioned in the third step in A.4.1. Figure 24 depicts the thread hierarchy.

![Figure 24: Schematic of thread hierarchy.](image)

### A.4.3 GPU Hardware Architecture and Memory Hierarchy

CUDA threads access data from different kinds of memories. In this section, we start with GPU hardware architecture to introduce the memory hierarchy.

CUDA GPU has a bunch of streaming multiprocessors(SMs), each of which consists of many simple processors(SPs) to run a bunch of parallel threads as illustrated in figure 25. A SP also refers to a core and is able to execute one thread maximally at once. In a SM, each thread on a core will be assigned their private local memory in the register. Therefore a thread can read and write from its corresponding local memory. A thread blocks is assigned a memory space visible to all threads in it termed shared memory, which is shown in figure 25. Therefore, shared memory is shared among threads in the same thread block and every thread can read and write to its corresponding share memory. In addition, there is a memory space that all threads in the entire GPU system can read and write is called global memory, which is the same as the device memory mentioned in A.4.1.

In terms of the memory accesses speed, local memory is the fastest and global memory is the slowest. One reason why the local memory is so fast is that it stores data in either registers or L1 cache, which are both quite fast. Moreover, local memory and shared memory are much faster than the global memory and all of them are much faster than the host memory. In order to minimize the time of the program, we need to spent time less on memory accesses, accordingly, store frequently used data in a faster memory.
A.4.4 Global Memory Access

*Coalesced* Instead of using faster memory, another approach to efficiently minimize the time spend on the whole program is called coalescing, which means coalesce threads’ accesses to the global memory. More specifically, whenever a thread on GPU read or writes global memory, it accesses a large chunk of memory at once, which is explained in figure 25. If other threads are accessing the memory unit in the same chunk of memory, GPU can reuse that for all threads that are trying to access that memory. Under this condition, GPU is most efficiently used since threads read or write contiguous global memory locations.

*Strided* Strided is another global memory access pattern, where adjacent thread accesses discontinuous global memory locations that may be in different chunk of memory. In the case, in order to read or write all these full threads, GPU needs to pull in two memory transactions to the chunks of memory in order to service that, the speed of which is half of that in coalescing. Accordingly, the more memory transactions a program has to do, the lower performance on the speed it will get.

*Random* Random is the slowest global memory access pattern, which affect the program speeds seriously. Random represents the memory unit that each thread is going to access is far or unrelated to each other in memory. Therefore, every thread gets its own memory transaction, which leads to bad memory performance for the memory system.
Figure 26: Schematic of global memory access.