Textured Mesh Reconstruction of Indoor Environments Using RGB-D Camera

Collin Boots

A THESIS

in

Robotics

Presented to the Faculties of University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Master of Science in Engineering

2014

Dr. Daniel D. Lee
Supervisor of Thesis

Dr. Camillo J. Taylor
Graduate Group Chairperson

Abstract

Your abstract goes here... ...

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Goals

As robots continue to be incorporated into human environments, the need for intelligent and high-speed reasoning about the objects around them increases dramatically. At the simplest level, mobile robots need to create a map of their environment for navigation. At a higher level, some robots need to recognize distinct objects in their environment, track object movement, and have some intuitive sense of object geometry that is easily stored and processed. Even more importantly, robots must be able to efficiently generate adaptable models of their environment, or worldview, from sensor data in real time. Many different methods for representing the world have been proposed and implemented, and they will be discussed below in more detail. Like the human brain, the robot should also be able to perform these low level functions with only minimal intervention from higher cognitive functions. Such technology also has potential uses beyond robotics. Applications may include easily modeling indoor environments for interior design concepts, generating 3D tours or maps for various buildings, or low resolution rough mapping of archaeological excavations.

This thesis is intended to work towards such a system based on RGB-D cameras, triangle meshes, and the powerful parallel computing capability modern Graphics Processing Units (GPUs) offer. RGB-D cameras like Microsoft's Kinect provide a great low cost solution for capturing 3D environments. Triangle meshes are efficient to store, simple to manipulate and refine, and very versatile. Meshes have the added benefit of being well suited to GPU hardware (which was originally designed for just that purpose). This thesis lays out a robust, high-speed GPU based pipeline that converts raw RGB and depth frames into a 3D textured mesh representation of large planar surfaces in the field of view.

1.2 Related Work

A great variety of methods have been applied to RGB-D camera data in an effort to construct a coherent worldview. Each has advantages and disadvantages.

1.2.1 Worldview Storage Models

Point Cloud Models The simplest approach to storing RGB-D data is as a raw point cloud. Each point is stored in a self contained data structure containing at least the point's position and color information. E.g.

$$P_i = \{pos_x, pos_y, pos_z, red, green, blue\}$$

This approach allows a complete record of the raw data to be stored very easily, but the size of the data stored will grow very quickly, and simply storing the data linearly results in very slow queries.

Nearest neighbor approximations have recently become a popular approach for speeding queries on large point clouds[20]. However, achieving reliably fast queries usually requires some form of hierarchical tree structure like K-d trees[23] or octrees[39]. K-d trees are much more adaptable and usually more efficient in terms of memory storage, but octrees have a significant advantage when it comes to incrementally building a point cloud because points can very easily be inserted into the appropriate octree leaf node with no duplication or restructuring. K-d trees are better suited for compressing point clouds offline.

Voxel Space Models Perhaps the most robust and impressive real-time surface reconstruction algorithm to date is Microsoft's Kinect Fusion[21, 13]. An open source implementation called KinFu is also available[27]. Kinect Fusion uses a bounded 3D voxel space where each voxel stores the distance to the nearest detected surface or empty space (the default). Figure 1.1 shows the workflow of the Kinect Fusion system. The point cloud generated by each RGB-D frame is projected into the voxel space and each point updates nearby voxels' distance to nearest surface metric. Implicit surfaces can then be rendered by raycasting through the voxel space and detecting the distance sign crossover point. This results in very high resolution and fidelity reconstructions of the implicit surfaces in the environment. The primary disadvantage of this approach is the workspace size is limited by memory and compute resources which scale with the resolution and dimensions of the voxel space.

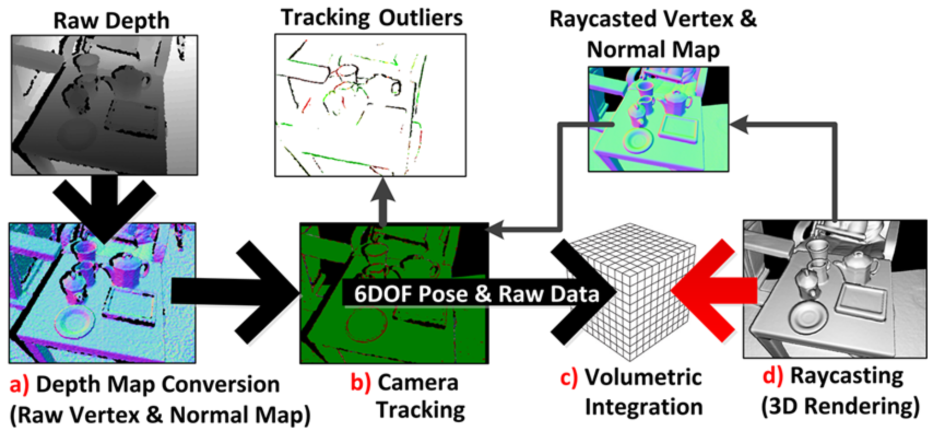

Figure 1.1: Kinect Fusion Tracking and Reconstruction Pipeline. Reproduced from[13]

A spacial extension of this system called Kintinuous was introduced in 2012[38, 1]. Kintinuous uses a mobile voxel space which periodically relocates based on camera motion. Voxels that move outside the space are converted to a triangular mesh using a greedy mesh triangulation procedure described by Marton *et al.*[19].

Others have attempted to use sparse voxel octrees (SVOs) to represent larger unbounded spaces using voxels of varying resolution[16, 30]. This approach is very amenable to human environments like buildings which are dominated by free space. However, all of these approaches are still limited to storing discrete data points and contain no inherent information about coherent objects or regions.

Point Cloud Offline Processing Point clouds can also be processed offline to extract surfaces and objects. These systems are not burdened by the strict requirements of real-time computation so they generally can produce more globally optimal solutions than their online counterparts. These systems generally target unordered point clouds. Marton *et al.* proposed an incremental approach to triangulation of noisy point clouds[19]. The approach was amenable to introduction of new registered RGB-D frames by only triangulating points that did not correlate with existing mesh models, but the implementation was far too slow for real-time applications. Ma and Others have introduced a system for planar simplification of dense point clouds using a QuadTree based algorithm[18, 17] and texture maps. This thesis will rely heavily on this innovation. Other approaches worked towards implicit surfaces like parameterized smooth surface fits [12] or Poisson surfaces[14, 5].

1.2.2 Real-Time Processing

A crucial part of this thesis is achieving high speed plane segmentation. Many different researchers have created efficient parallel GPU implementations of common segmentation approaches like Markov Random Fields[3], the Potts model[2], parallelized graph-based approaches[37], seeded region growth (SRG)[26], and region growth based on local smoothness constraints[29]. However, this thesis parallelizes an algorithm by Holz *et al.* [10] based on clustering in normal space. This approach is superior for application in this thesis because it can readily detect large planes and solve them globally in a very data parallel manner.

1.2.3 Additional Resources

Although this thesis focuses on detecting and representing only planar elements, the envisioned worldview generating pipeline would need several additional components and capabilities to be successful. Additionally, many data processing technologies exist that can improve the quality of the input data through more sophisticated filtering.

RGB-D Filtering The Kinect sensor provides its own set of filtering challenges. Khoshelham and Elberink provided a very useful in depth analysis of Kinect accuracy and resolution specifically with indoor mapping applications in mind[15]. Without some amount of preprocessing and filtering, quantization of the depth data and the fact that resolution and accuracy decrease with increasing distance from the sensor would completely prevent this thesis's pipeline from producing any usable results.

This thesis uses bilateral filtering of the depth image[25, 35]. A simple Gaussian filter is used to smooth local point normals, but other methods exist that could improve results if they could be implemented efficiently in parallel. One such method is adaptively computing filter windows using integral images[11]. Kinect data also is notoriously full of holes from washed out areas, shadows and other noise related effects. Some research makes an effort to fill these holes[6, 32, 40], but this thesis is geared towards progressive improvement of the world model and hopes that any holes will be patched by other viewing angles.

3D SLAM This thesis only deals with processing the current frame, but it was designed with an eye towards creating a closed loop system to integrate data from multiple frames. To accomplish this, a Simultaneous Localization and Mapping (SLAM) system will be needed. SLAM has been practically implemented using a Kinect using a GPU [34, 33]. Whelan *et al* did fantastic work in comparing multiple SLAM methods combining RGB feature tracking and full point cloud registration algorithms[1].

Mesh Processing and Modification One future direction for this work is to incorporate new information into existing meshes through efficient topology changes and resolution modification. The graphics community offers a wide range of insight into these methods, ranging from tracking surfaces through complex topology evolution[4], modeling deformable solids[31], Mesh subdivision and simplification approaches[28], and texture re-mapping or the effects of image warping[9, 8, 36, 24, 7].

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 will review some basic precepts of parallel algorithm design, along with specific optimization considerations for programming GPUs with CUDA. Chapter 3 will provide an overview of the system design as well as break down the pipeline into sub-modules to be explored in much more detail in Chapter 4. Finally, Chapters 5 and 6 will provide performance analysis, conclusions, and areas of potential improvement for future work.

Chapter 2

Parallel Programming Paradigms

This chapter will provide and introduction to the fundamentals parallel programming. The remaining chapters will assume a working knowledge of parallel algorithms, GPU architecture, and CUDA optimization techniques.

2.1 Principles of Parallel Programming

As the physical limits of transistor size are being reached, the trend in computing has been away from making processors faster and towards parallel processing. The introduction of General Purpose Graphics Processing Units have made data parallel algorithms a very attractive approach to accelerating computationally intensive tasks. However, migrating from sequential computing to parallel platforms is not always simple or even wise. The following are some principles to keep in mind when considering using GPU acceleration.

Keep data parallel Parallel computing is best suited to performing simple operations on large numbers of independent data elements, such as n-body simulations and per-pixel image processing operations. Aside from these trivial cases, adapting sequential algorithms to a GPU architecture efficiently can be challenging. Not all loops are easily parallelizable.

Memory is a bottleneck Computing with thousands of cores in a GPU makes memory I/O into a major bottleneck. Usually the data transfer to and from the parallel device will take much longer than the actual compute time. Be extremely conscious or memory usage.

Minimize code divergence On most architectures, parallel code runs in batches for increased efficiency. If the code is full of if/then/else statements, the divergent execution paths can dramatically impact performance.

Avoid sequential operations and synchronization Atomic operations that must be performed in order or synchronizations between concurrent threads of execution can be very slow. The more the threads can be independent of each other, the more efficiently the hardware will be able to execute them in parallel.

Be mindful of the hardware When writing performance critical code, understanding the underlying architecture can be a big help. Understanding how the code will actually be distributed and executed across multiple processors will make optimization much easier.

2.2 Parallel Algorithm Building Blocks

Just as there are fundamental building blocks of sequential computing like search, recursion, and iteration, many parallel algorithms make use of primitive operations like reduce, scan, and stream compact.

2.2.1 Reduction

Reduction is any operation on an array of elements that computes a single result. Common examples include sums, average, min, max, and product. In sequential terms, implementing reduction is very simple. Using sum as an example, algorithm 1 shows how this might be implemented.

Data: n element array X
Result: sum of elements in X
i = 0;
sum = 0;
while *i* ¡ *n* **do**
    sum += X[i];
    i++;
end

Algorithm 1: Sequential Sum

Notice that in the sequential algorithm, the accumulator's value at each iteration depends on the value in the previous iteration. To perform a parallel sum, the associative property of the operator can be exploited to break group the operation into a structured set of binary operations that can be performed in parallel (Figure 2.1).

This parallel reduction method will work for any associative binary operator.

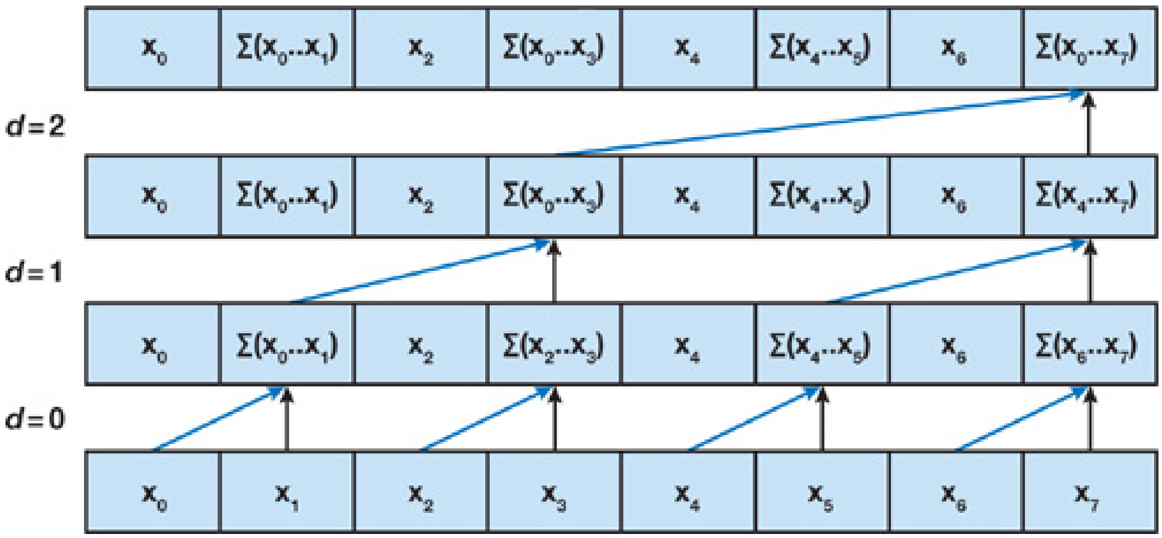

Figure 2.1: Parallel Reduction. Reproduced from[22]

2.2.2 Scan

The scan operation is similar to a performing a cumulative sum operation on an array. There are two forms of scan: exclusive and inclusive. Usually the term scan refers to an exclusive scan.

Given an associative binary operator $\oplus$ with identity value I and an array of n items $[a_0, a_1, \ldots a_{n-1}]$, an exclusive scan returns the array:

$$[I, a_0, (a_0 \oplus a_1), (a_0 \oplus a_1 \oplus a_2), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})]$$

An inclusive scan returns the shifted result:

$$[a_0, (a_0 \oplus a_1), (a_0 \oplus a_1 \oplus a_2), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})]$$

The sequential implementation of a sum scan is trivial and outlined in Algorithm 2.

Data: n element array X
Result: array Y contains exclusive scan of X
i = 1;
Y[0] = 0;
while *i* ¡ *n* **do**
  Y[i] = Y[i-1] + X[i-1];
  i++;
end

Algorithm 2: Sequential Sum

To perform a parallel scan in a work efficient manner, first a reduction is performed on the array as in Figure 2.1. Then, the last element of the array is set to 0, and a series of swaps and sums are performed as shown in Figure 2.2.

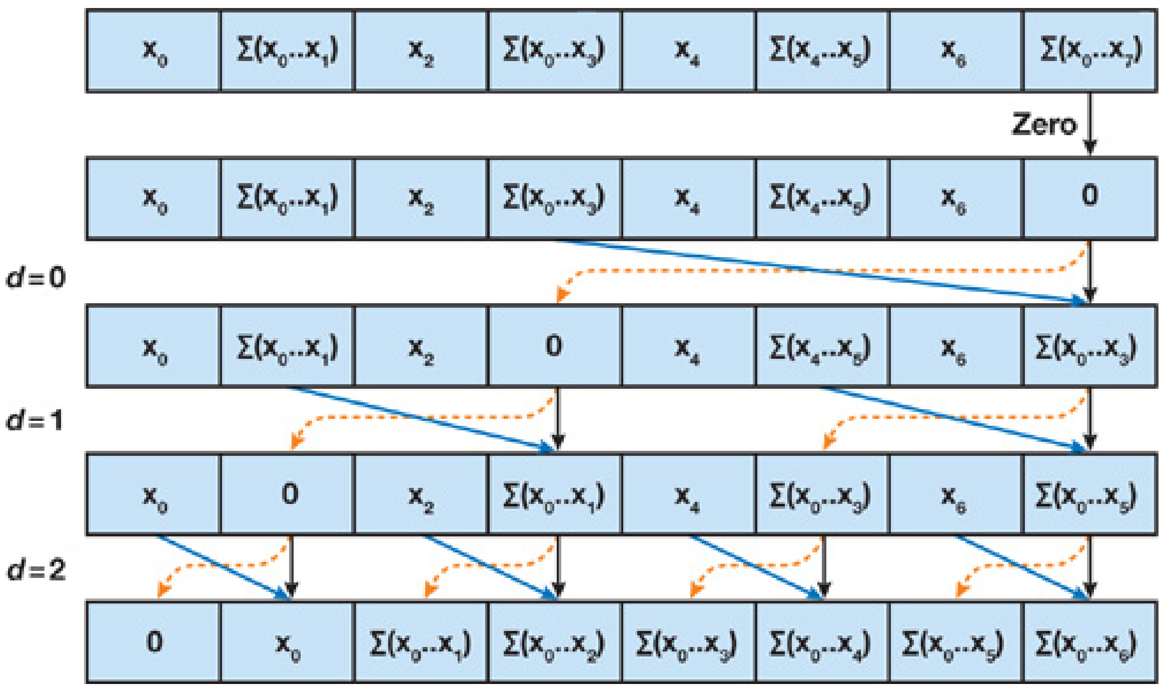

Figure 2.2: Down-sweep phase of parallel sum scan. Reproduced from[22]

2.3 Stream Compaction

In parallel computing, stream compaction is a useful tool for selecting a subset of data and compressing it into a coherent memory block. For example, stream compaction could be used to transfer all of the odd elements in array A of Figure 2.3 and place them in order compressed at the start of array B.

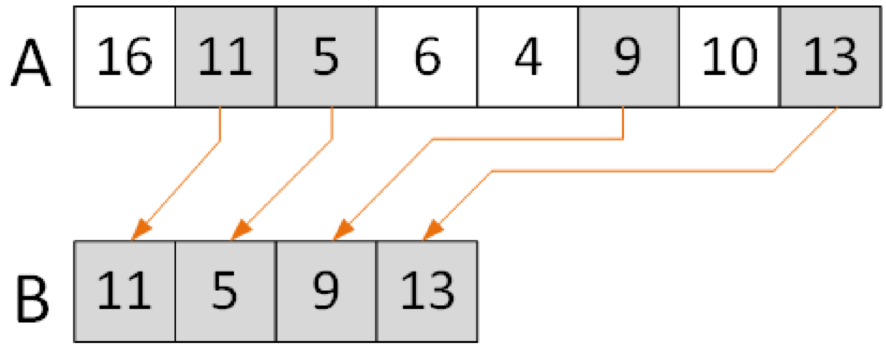

Figure 2.3: Example of stream compacting odd valued integers

Accomplishing this is surprisingly simple. As shown in Figure 2.4, a secondary array of flags is created. This array has a 1 for every odd number in array A and 0 for all others. An exclusive scan is performed on the flag array. Notice that every flagged element now has a unique zero-based index associated with it. The flagged elements are then "scattered" by the computed index. The scatter function is outlined in Algorithm 3.

```
Data: input array A and scanned flag array F, both of length n
Result: output stream compacted array B
foreach i=0...n-1 in parallel do
    if isFlagged(A[i]) then
        B[F[i]] = A[i];
    end
end
```

Algorithm 3: Sequential Sum

2.4 Programming with CUDA

CUDA is a C/C++ based programming language created by NVIDIA to expose the general compute functionality of their GPU processors.

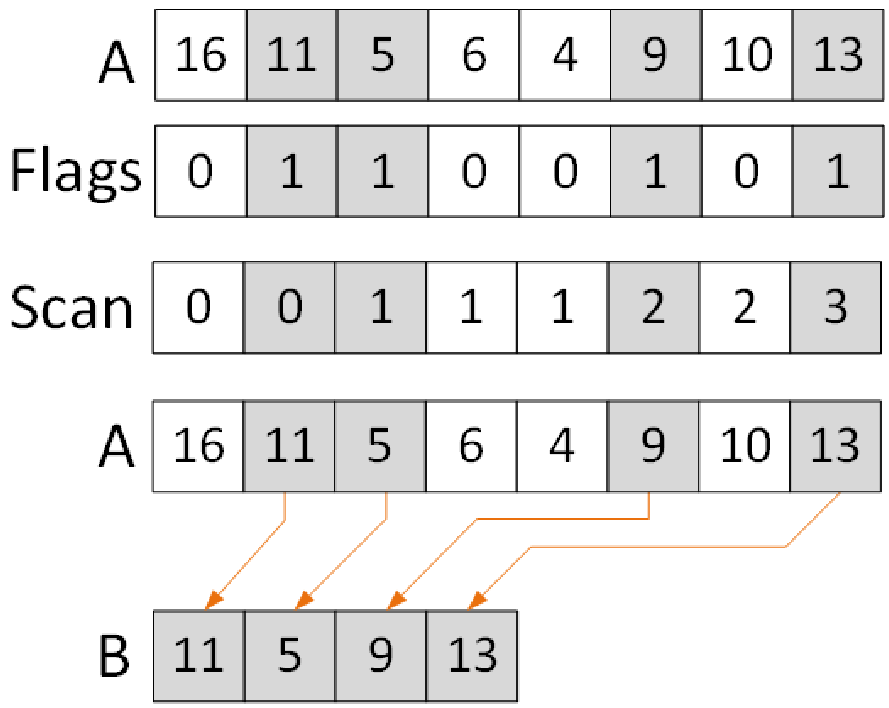

Figure 2.4: How stream compaction works (Scan and Scatter)

2.4.1 CUDA GPU Architecture

CUDA divides the computer into host and device. The host refers to the CPU and all memory it has access to, and the device refers to the GPU and its memory banks. Memory transfers between the host and GPU are started and managed by the host.

A piece of code that runs in parallel on the device is called a kernel. Figure 2.5 shows how kernels are organized. Each individual running process is called a "thread". Threads are organized into "blocks" of up to 1024 threads on modern architectures. Blocks are then grouped into "grids". Within each block, threads are executed in groups of 32 threads called "warps". Every thread in a single warp executes simultaneously on a single processor core. The device memory architecture is depicted in Figure 2.6. Global memory, constant memory, and texture memory are all accessible by any thread in the kernel. The primary difference between these memories . Every thread also has access to a limited amount of memory which is shared between all threads in a block. Shared memory is much faster than global memory access, and can be used to accelerate kernels that need to share data between threads locally. Finally each thread has access to its own local register memory.

2.4.2 Optimizing CUDA Code

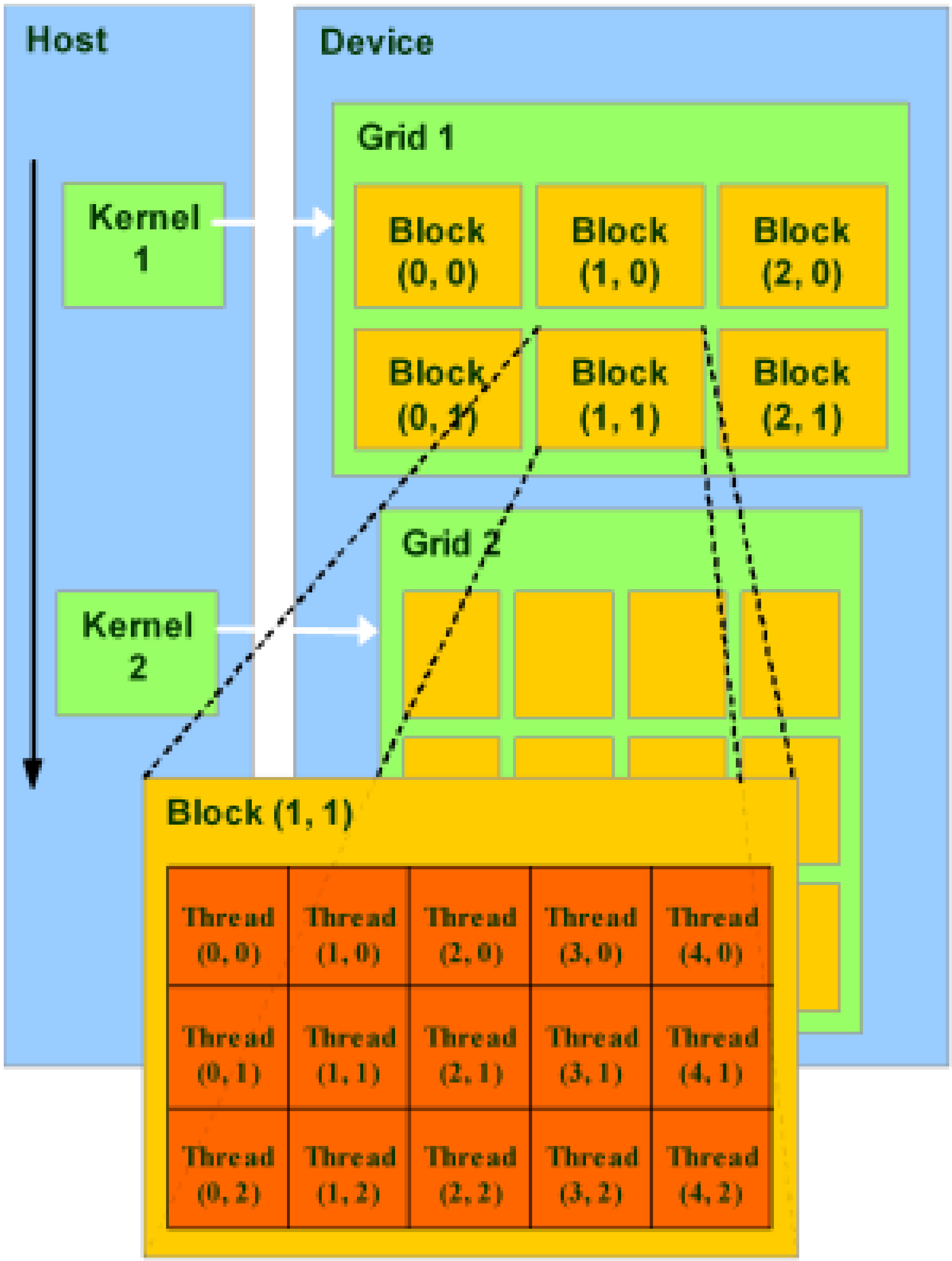

Figure 2.5: CUDA Kernel Organization

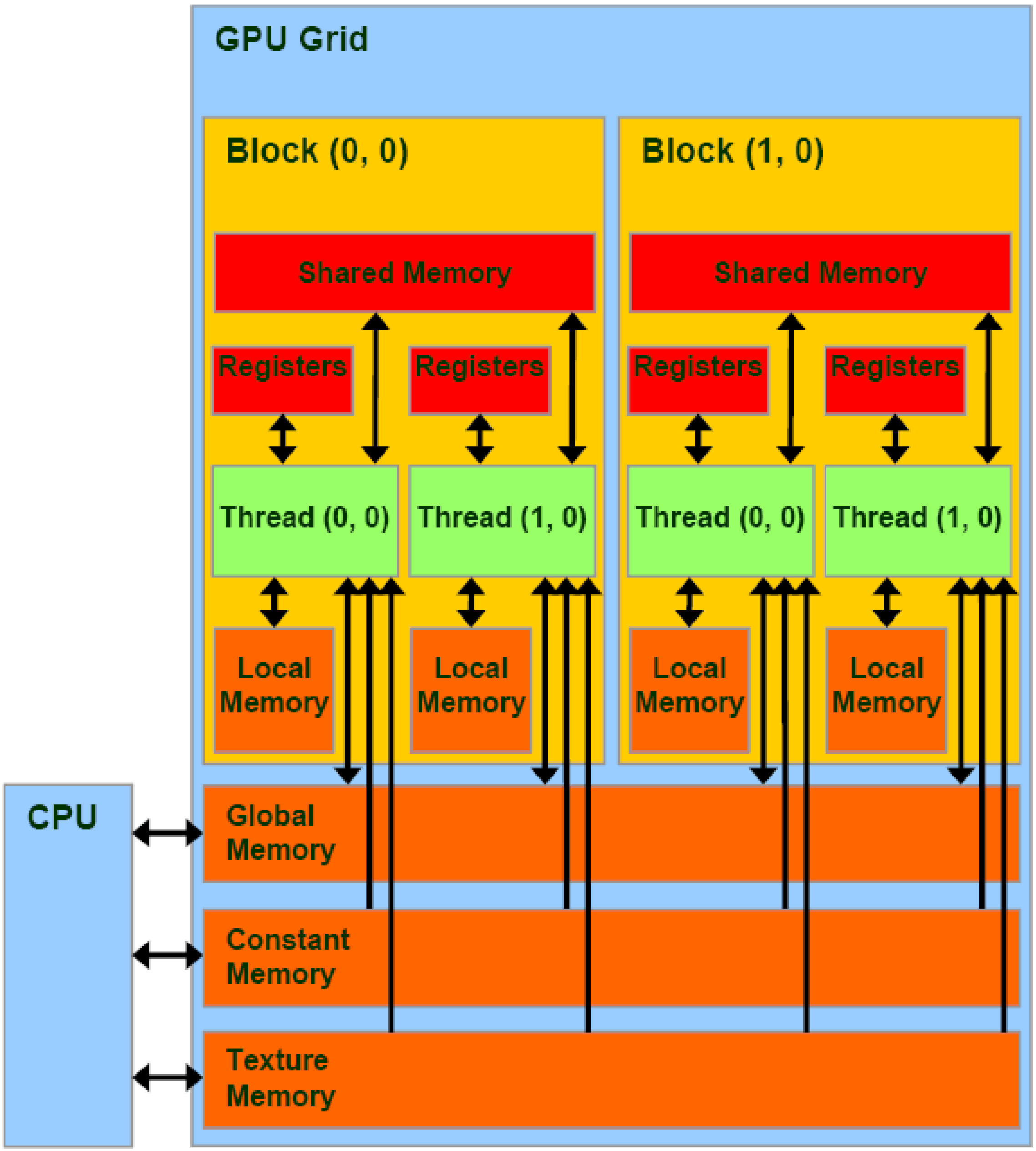

Figure 2.6: CUDA Memory Organization

Chapter 3

Problem Formulation and Approach

3.1 Problem Specification

3.2 High Level System Design

3.3 Plane Detection and Meshing Pipeline Design

Chapter 4

Implementation

4.1 RGBD Framework

4.2 Preprocessing

4.3 Plane Segmentation

4.4 Mesh Generation

Chapter 5

Performance Analysis

Chapter 6

Conclusions and Future Work

Bibliography

[1] *Robust real-time visual odometry for dense RGB-D mapping*, May 2013.

[2] Alexey Abramov, Tomas Kulvicius, Florentin Wörgötter, and Babette Dellen. Real-time image segmentation on a gpu. In *Facing the multicore-challenge*, pages 131–142. Springer, 2011.

[3] Pilar Arques, F Aznar, M Pujol, and Ramón Rizo. Real time image segmentation using an adaptive thresholding approach. In *Current Topics in Artificial Intelligence*, pages 389–398. Springer, 2006.

[4] Morten Bojsen-Hansen, Hao Li, and Chris Wojtan. Tracking surfaces with evolving topology. *ACM Trans. Graph.*, 31(4):53, 2012.

[5] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel poisson surface reconstruction. In *Advances in Visual Computing*, pages 678–689. Springer, 2009.

[6] Abdul Dakkak and Ammar Husain. Recovering missing depth information from microsofts kinect, 2012.

[7] Yanwen Guo, Hanqiu Sun, Qunsheng Peng, and Zhongding Jiang. Mesh-guided optimized retexturing for image and video. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):426–439, March 2008.

[8] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, pages 56–67, November 1986.

[9] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Berkeley, CA 94720, June 1989.

[10] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317. Springer, 2012.

[11] Stefan Holzer, Radu Bogdan Rusu, M Dixon, Suat Gedikli, and Nassir Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2684–2689. IEEE, 2012.

[12] Kai Hormann. From scattered samples to smooth surfaces. *Proc. of Geometric Modeling and Computer Graphics*, 2003.

[13] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using

a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[14] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[15] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[16] S. Laine and T. Karras. Efficient sparse voxel octrees. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1048–1059, Aug 2011.

[17] Lingni Ma, R. Favier, Luat Do, E. Bondarev, and P.H.N. de With. Plane segmentation and decimation of point clouds for 3d environment reconstruction. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 43–49, Jan 2013.

[18] Lingni Ma, Thomas Whelan, Egor Bondarev, Peter HN de With, and John McDonald. Planar simplification and texturing of dense point cloud maps. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 164–171. IEEE, 2013.

[19] Z.C. Marton, R.B. Rusu, and M. Beetz. On fast surface reconstruction methods for large and noisy point clouds. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3218–3223, May 2009.

[20] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.

[21] Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[22] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, 2007.

[23] A. Nüchter, K. Lingemann, and J. Hertzberg. Cached k-d tree search for icp algorithms. In *Proceedings of the 6th IEEE international Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM '07)*, pages 419–426, August 2007.

[24] Masaaki Oka, Kyoya Tsutsui, Akio Ohba, Yoshitaka Kurauchi, and Takashi Tago. Real-time manipulation of texture-mapped surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, pages 181–188, July 1987.

[25] Tuan Q Pham and Lucas J Van Vliet. Separable bilateral filtering for fast video preprocessing. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[26] Juan C Pichel, David E Singh, and Francisco F Rivera. A parallel framework for image segmentation using region based techniques.

[27] Michele Pirovano. Kinfu – an open source implementation of kinect fusion. http://homes.di.unimi.it/ pirovano/pdf/3d-scanning-pcl.pdf, 2012. PhD student in Computer Science at POLIMI.

[28] E. Puppo and D. Panozzo. Rgb subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):295–310, March 2009.

[29] Tahir Rabbani, Frank van den Heuvel, and G Vosselmann. Segmentation of point clouds using smoothness constraint. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5):248–253, 2006.

[30] Julian Ryde and Jason J Corso. Fast voxel maps with counting bloom filters. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4413–4418. IEEE, 2012.

[31] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. Hybrid simulation of deformable solids. In *2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 81–90, August 2007.

[32] Luciano Spinello and Kai Oliver Arras. People detection in rgb-d data. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3838–3843. IEEE, 2011.

[33] F Steinbrucker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722. IEEE, 2011.

[34] J Sturma, E Bylowb, C Kerla, F Kahlb, and D Cremersa. Dense tracking and mapping with a quadrocopter.

[35] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[36] Lifeng Wang, Sing Bing Kang, R. Szeliski, and Heung-Yeung Shum. Optimal texture map reconstruction from multiple views. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–347–I–354 vol.1, 2001.

[37] Jan Wassenberg, Wolfgang Middelmann, and Peter Sanders. An efficient parallel algorithm for graph-based image segmentation. In *Computer Analysis of Images and Patterns*, pages 1003–1010. Springer, 2009.

[38] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[39] K. M. Wurm, A. Hornung, M Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, USA, May 2010.

[40] Lu Xia, Chia-Chih Chen, and JK Aggarwal. Human detection using depth information by kinect. In *Computer Vision and Pattern Recognition Workshops*

(CVPRW), 2011 IEEE Computer Society Conference on, pages 15–22. IEEE, 2011.