University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations

2016

Search Space Representation of Near Field 3D Scenes using Microsoft Kinect

Ronak Shaileshkumar Patel
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Patel, Ronak Shaileshkumar, "Search Space Representation of Near Field 3D Scenes using Microsoft Kinect" (2016). *Electronic Theses and Dissertations*. 5785.
https://scholar.uwindsor.ca/etd/5785

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

المنارة للاستشارات
www.manaraa.com

Search Space Representation of Near Field 3D Scenes using Microsoft Kinect

By

Ronak Patel

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of **Computer Science**
in Partial Fulfillment of the Requirements for
the Degree of **Master of Science**
at the University of Windsor

Windsor, Ontario, Canada

2016

© 2016 Ronak Patel

www.manaraa.com

Search Space Representation of Near Field 3D Scenes using Microsoft Kinect

by

Ronak Patel

APPROVED BY:

_____

Dr. Behnam Shahrrava
Electrical and Computer Engineering

_____

Dr. Dan Wu
Computer Science

_____

Dr. Scott Goodwin, Advisor
Computer Science

_____

Dr. Boubakeur Boufama, Co-Advisor
Computer Science

24 May, 2016

www.manaraa.com

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

www.manaraa.com

ABSTRACT

Augmented Reality (AR) systems allow users to experience reality with extra data. These systems can be used in various applications such as real-time informatics and games. An important requirement for many games is the ability to find a path from one point to another with minimum cost.

In recent years, pathfinding algorithms have evolved tremendously, and researchers have created numerous variations and techniques that have improved game experience. AR games can be played in an immersive way using the real world as a game world, however, the information perceived by these systems do not provide suitable search spaces (and search graphs) for pathfinding algorithms.

This thesis proposes a novel method which generates a search space representation from the perceived information from an AR system, specifically Microsoft's Kinect. The generated search space can provide a basis to apply existing knowledge of pathfinding to augmented reality.

www.manaraa.com

DEDICATION

To my awesome papa, loving mama and dashing sister

www.manaraa.com

ACKNOWLEDGEMENTS

I would like to thank my supervisor and advisor Dr. Scott Goodwin for selecting me as his student and guiding me throughout my master's study. His guidance and knowledge were helpful for me from beginning to end. I would like to thank my co-supervisor Dr. Boubakeur Boufama for helping me in my thesis.

I also want to thank my thesis committee for giving their time in reviewing my thesis. I would love to thank Mrs. Karen Bourdeau, for her continuous support and assistance. I am also thanking my friend and colleague Halen Whiston for helping me in improve the writing of my thesis. I am so thankful to University of Windsor and its staff for being so supportive of international students.

www.manaraa.com

TABLE OF CONTENTS

www.manaraa.com

www.manaraa.com

www.manaraa.com

LIST OF TABLES

www.manaraa.com

LIST OF FIGURES

www.manaraa.com

www.manaraa.com

www.manaraa.com

www.manaraa.com

LIST OF ALGORITHMS

www.manaraa.com

LIST OF ABBREVIATIONS/SYMBOLS

Navmesh – Navigational Mesh

AR – Augmented Reality

3D – Three Dimensional

2D – Two Dimensional

PTAM – Parallel Tracking and Mapping

SLAM – Simultaneous Localization and Mapping

ø - The angle of surface unification

www.manaraa.com

CHAPTER 1

Introduction

1.1 Thesis Claim

This thesis proposes a new technique for creating a search space representation from a 3D scene using the Microsoft Kinect depth sensor. The generated search space can be used as a map or search graph for pathfinding in augmented reality games.

1.2 Pathfinding

Pathfinding algorithms solve the problem of finding a shortest (or least cost) path between two points in a given searchable environment. These algorithms are essentially graph search algorithms.

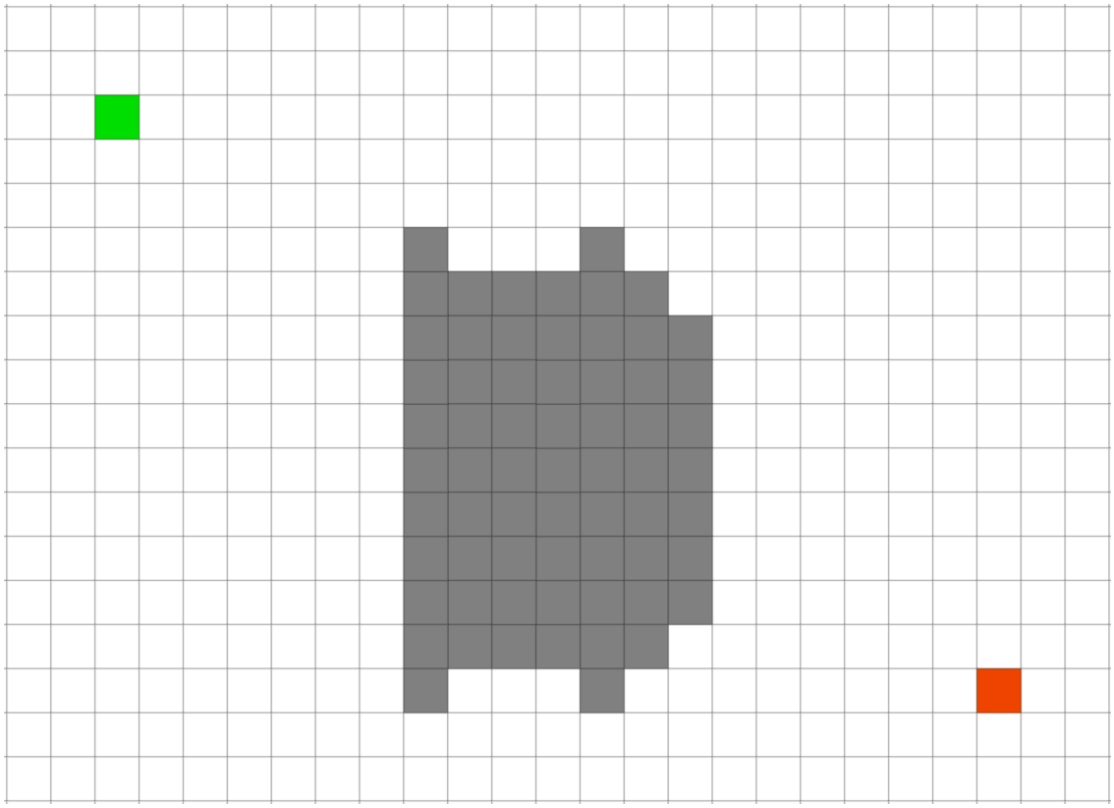


Figure 1.1 Pathfinding environment with initial setup

www.manaraa.com

For example, consider point S as a starting point and point G as a goal point and assume these points are located on some searchable environment (e.g. map). Pathfinding algorithms use knowledge of the environment, typically encoded in the form of a search space representation (or search graph) and possibly some additional heuristic information, to return the shortest (obstacle avoiding) path. We will provide more details in a later chapter but for an overview, here is some valuable information.

Figure 1.1 shows an initial setup of pathfinding scenario. The green dot shows starting position and red dot represents a goal position. Grey blocks are considered as a walls or obstacles. There are lots of ways to represent the environment but for simplicity, here we consider a grid representation.

This initial setup is passed to a pathfinding algorithm which finds a path (typically, shortest or least cost) from the start position to the goal position avoiding all obstacles. For different scenarios, we can implement lots of variations and different mechanisms in the pathfinding setup. For example, the above setup uses only a four-way transition on the grid; alternatively, one might use eight-way or diagonal transition.

In Figure 1.2, the cyan blocks represent nodes that have been explored (*closed list*) by the pathfinding algorithm, green blocks correspond to nodes on the search frontier (*open list*) by the pathfinding algorithm, and the yellow line represents the path found on the map. Though there are many pathfinding algorithms and search space representations in the literature, the above is typical of many of them.

Pathfinding Algorithm

Search Space Representation

www.manaraa.com

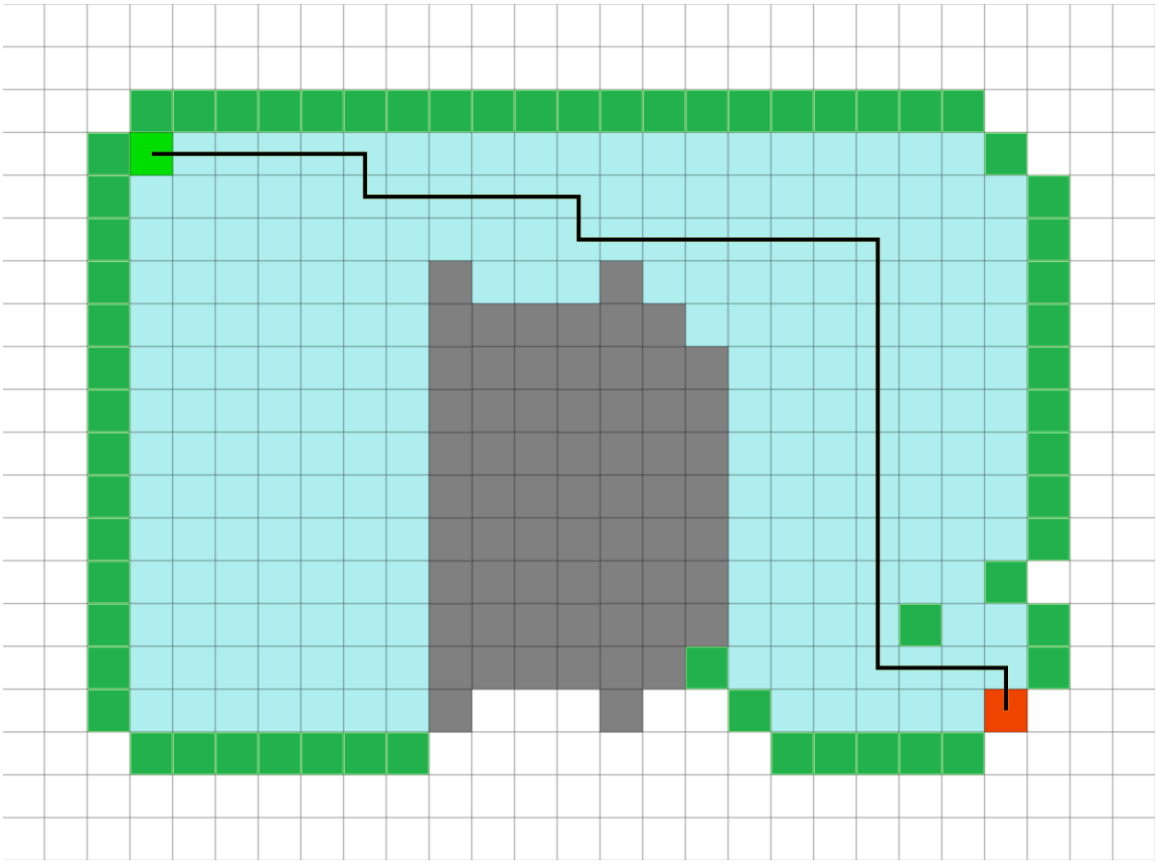


Figure 1.2 Output of pathfinding process

The most well-known pathfinding algorithm is A* [13], usually pronounced as *A-star*. In general, pathfinding algorithms are graph-search algorithms. The most common search space representations are grid-based, waypoint-based, and navmesh-based. We will look at these in the next section.

1.2.1 Search Space Representation

Search space representations are commonly referred to as maps, and they are a spatial representation of a search space. Games are created in a computer-generated synthetic environment. In such environments, the game creator has full control over all aspects of the game design. These environments commonly consist of 3D models of objects and

www.manaraa.com

hidden layers of search space. Search spaces can make game experiences more user-friendly. It is also useful to implement AI game characters that find a path using search space. Search spaces are graphs at their core. There are many different types of search space representations available to the game designer but for a simple introduction, we will consider the search spaces below:

- Grid representation
- Navmesh representation

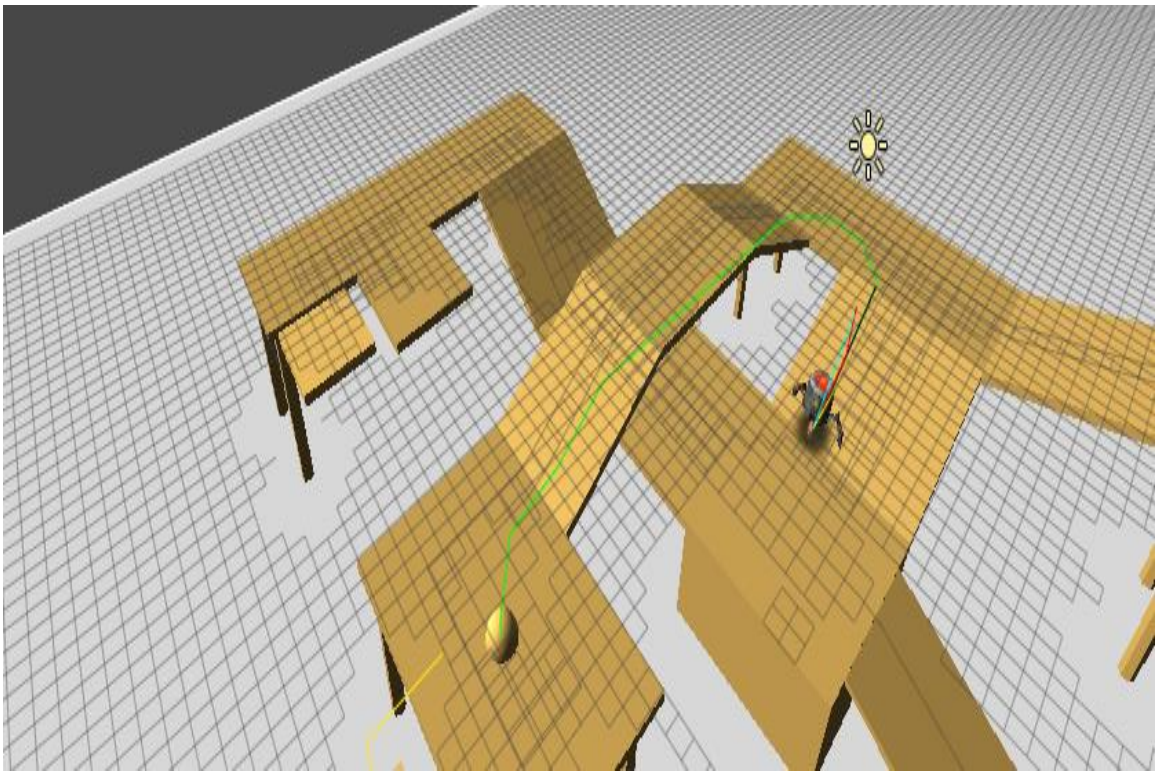


Figure 1.3 Grid search space representation [1]

Figure 1.3 shows simple grid representation for a search space. There are lots of variations in grid representation that are available, like a hexagonal grid. A square grid has an equal size of square pieces called tiles. A square grid is made of equally distanced

www.manaraa.com

horizontal and vertical lines. Each tile in the square grid is considered as a node in an equivalent graph, and each side is considered an edge. In this case, they are called four-way connected grids. It is really important to understand that this graph representation is (normally) not visible to users, though, it is treated as a layer of map design and used in pathfinding. In some cases, nodes are connected to other nodes with eight different ways which include corners. This kind of representation is called an eight-way connected grid.

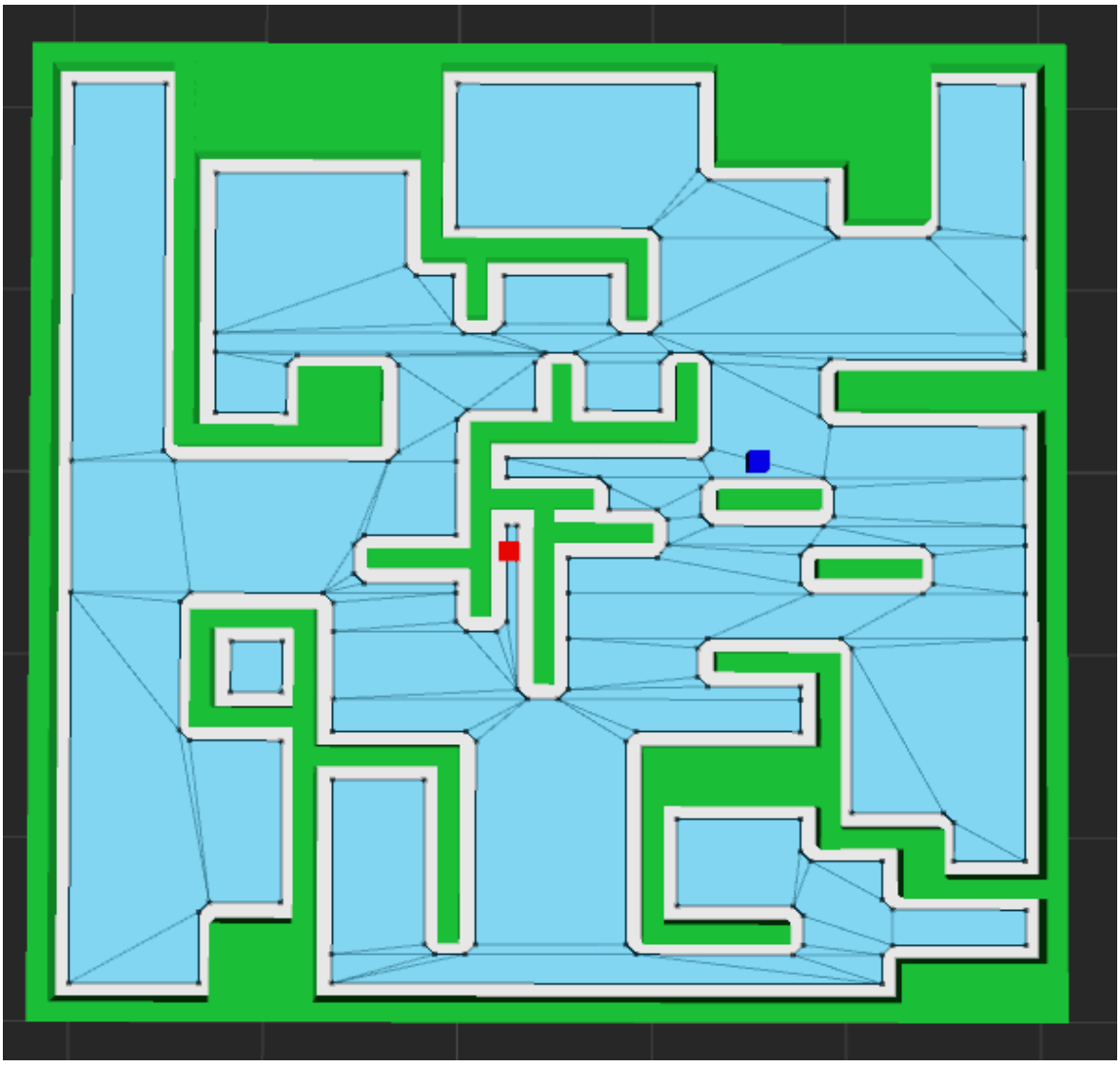


Figure 1.4 Navmesh search space representation [2]

www.manaraa.com

A navigational mesh, commonly referred to as a navmesh is an interconnected set of polygons representing a search space. The polygons used in a navmesh are *convex*. For pathfinding purposes, each polygon is considered as a node in the graph and a connection between two polygons is considered as an edge between two nodes.

As shown in Figure 1.4, the green portion represents the obstacle, the blue portion represents the searchable (walkable) space, and the white portion represents the radius of the game character (it ensures there is room to move without scraping the walls). The searchable space is divided into convex polygons which are navmesh tiles. We will discuss more about navmesh in later chapters as this thesis mainly focused on navmesh search space representation.

1.3 Depth Sensors

Classical cameras can capture black and white or color photos. According to the pinhole camera model, photographs are a two-dimensional representation of the three-dimensional scenes. In technical terms, this kind of camera is a type of RGB sensor because they capture photo frames in Red, Green and Blue color combination.

Human eyes and brains are trained to observe these photographs with depth perception, but computers are not and this is one of the classical problems in computer vision. If we need to teach computers how to process taken frames, we need to calculate depth. There are lots of hardware and software solutions to measure depth, but below are the two main types of hardware for depth perception:

- Active depth sensors
- Passive depth sensors

www.manaraa.com

Figure 1.5 Laser range finding sensors

Figure 1.5 shows laser range finding sensors from different manufacturers. Laser range finding sensors are classified as active depth sensors. Researchers have developed various methods and algorithms to calculate depth using these sensors. For example, the *Time of Flight* method which calculates depth using the round trip time taken by a laser beam to travel from source to source. The biggest disadvantage of these systems is cost and complexity of operation. There is another kind of active depth sensor which uses structured light to detect depth. One notable example of this is Microsoft Kinect.

Figure 1.6 Microsoft Kinect depth sensor [3]

The biggest advantages of Microsoft Kinect sensor are its low price, accuracy, and strong developer community. Many researchers have used Kinect in their projects to detect

www.manaraa.com

depth and so do we. We will discuss more about Kinect in later chapters as it is mainly used in experiments.

The stereo camera system is the best example of a passive depth sensor. It uses a triangulation technique to determine depth. These systems are cheap and easy to use, but the biggest disadvantage is their accuracy. The depth perception using this type of sensor is not as good as active depth sensors.

Figure 1.7 Stereo camera system

1.4 Application of Research

The proposed technique in this research can be used in various applications, but we have focused more on Augmented Reality as it is motivation for this thesis.

1.4.1 Augmented Reality

Augmented Reality [14] means the experience of reality with extra data. This extra data is added to the user's experience of reality, and hence the experience is termed augmented reality. Humans perceive reality using their senses granted by their eyes, ears, skin, nose, etc. We gather information using our eyes as visuals or a stream of images. Different senses perceive information in different forms like how ears perceive sound. The computers use different sensors to perceive the environment such as cameras for visual

www.manaraa.com

information. This thesis mainly focuses on visual augmented reality systems, although there are different augmented reality concepts available like audio augmented reality, haptic feedback augmented reality, etc. Visual augmented reality, which is also commonly referred to as simply Augmented Reality (AR), mostly deals with the visuals of the surrounding environment. The diagram below show what an AR system looks like.

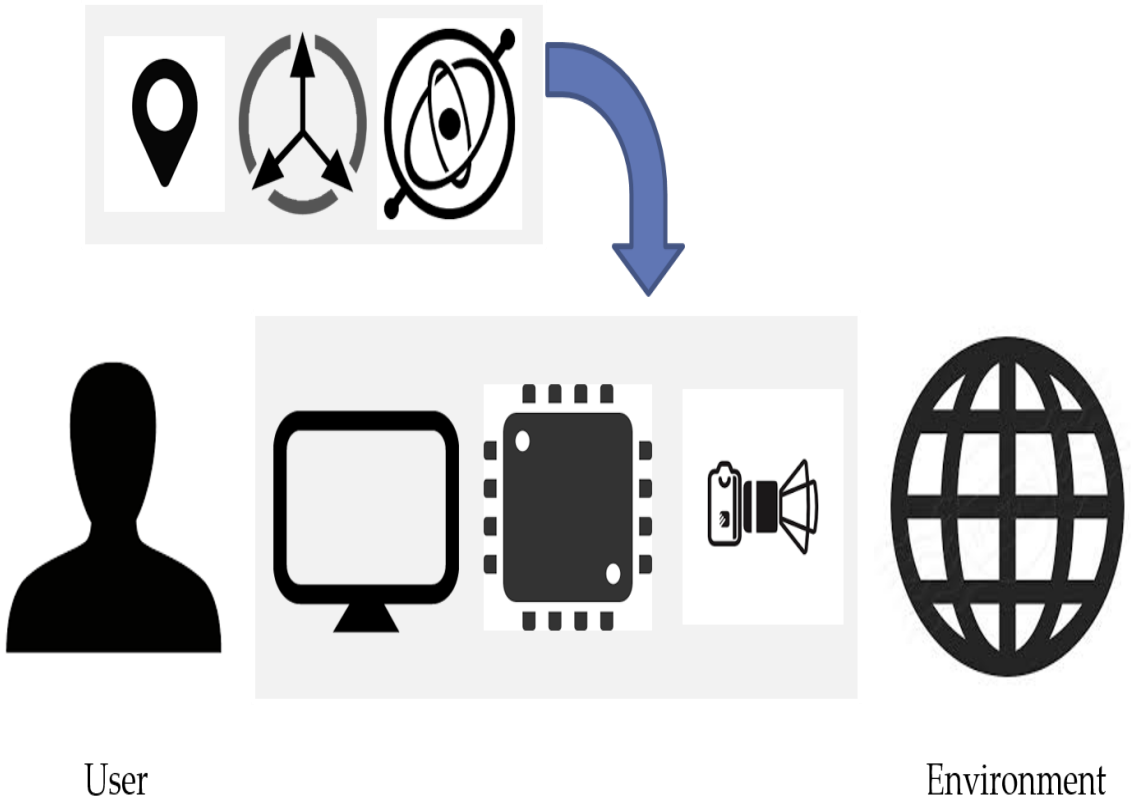


Figure 1.8 A simplified augmented reality system

As shown in Figure 1.8, an AR system at its bare minimum, consists of a visual receptor (camera), a processing unit (computational unit) and display. On one hand, we have the environment to perceive and on the other hand, we have a user who will experience the AR. In some cases, various sensors are used like magnetometer, gyroscope, and accelerometer which we will discuss later on. A camera normally catches visuals from

www.manaraa.com

the environment and transfers them to a computation unit in the form of frames. A computational unit processes those frames and decides what to augment on top of that and then transfers visuals to a screen with augmented data.

Augmented reality can bifurcate into two types: marker-less AR and marker-based AR. The marker-based AR system relies on 2D or 3D markers to create environment awareness, like barcodes or 3D objects. Marker-less AR does not depend on any markers or environment knowledge. Instead, they collect information and produce environment awareness during execution.

1.5 Problem Domain

It is really important to understand the relationship between pathfinding and augmented reality before going further. Both of these concepts are used in game development, although pathfinding handles object navigation within an environment, whereas AR is focused on the immersion factor.

Games in augmented reality generally use marker-based AR concepts where they don't need environment information, which is the reason why they don't need real-time pathfinding. If you play immersive games by considering real world as your map, then that scenario is the same as virtual games. Given that virtual games need pathfinding and because of the lack of search space representation in markerless augmented reality games, developers cannot apply existing knowledge of pathfinding to AR games. To fill this gap between pathfinding and augmented reality, one needs to create search space representation to make pathfinding knowledge applicable in the context of AR.

www.manaraa.com

1.6 Motivation

Most available augmented reality mobile games use marker-based augmented reality concepts. It should be noted however, that marker-based augmented reality produces a limited immersive game-playing experience. It also requires complex arrangements, like setting up markers before playing games, and it is also costly and time-consuming. In the end, this type of setup delivers a less engaging experience, which is counter-intuitive in the context of AR. On the other hand, marker-less augmented reality games can be played in a more immersive way by using the real world as a map for games. One needs to collect environment or scene-specific information using some vision technology.

Most SLAM algorithms are used to create 3D reconstruction (models, environment, etc.) from the scene, but this reconstruction can be used further to create search space representation. To the best of our knowledge, search space representations are not used in AR games because of the early stage of augmented reality. As many people believe, necessity is the mother of invention; there is no need to create search space because most games are marker-based, but as technology evolves, more complex games will be released on the market.

Pathfinding research has carried out numerous ways to determining a path in games. This existing knowledge is also applicable to augmented reality games, but it is not included due to the lack of search space representation. Hence, the main motivation of this research is to reduce the gap between augmented reality and pathfinding in the context of games. Our aim is to produce a search space representation, specifically a navmesh, from a 3D scenes reconstructed from the output of a Kinect device. Our work takes a

www.manaraa.com

reconstructed 3D scene as input, produces useable navmesh and extracts associated search graphs, which could then be used by existing pathfinders.

1.7 Thesis Outline

Chapter 1 stats our thesis claim and then introduces the basic thesis-related topics. It gives an overview of pathfinding and augmented reality. The thesis motivation is then described.

Chapter 2 focuses on background literature review and related work. It deals with explanation of existing work and some of the very important research carried out in the past. It describes various important algorithms and techniques from the KinectFusion algorithm to automated navmesh generation.

Chapter 3 explains useful concepts to understand further chapters. Chapter 4 is dedicated to our approach. This chapter explains all the concepts developed in this thesis. The experiments are presented in Chapter 5. The conclusion and future work are given in Chapter 6.

www.manaraa.com

CHAPTER 2

Background and Related Work

2.1 SLAM

SLAM algorithms help to build point cloud of a 3D scene. We will see details of point cloud in later chapters but for now, it is a way of representation for a 3D scene. As we are working with Microsoft Kinect sensor, we have to look for depth-based SLAM algorithms. One of the most well-known of these algorithms is *Dense Planar SLAM* [4] which uses an RGB-D sensor to map the environment. The Dense Planar SLAM represents surfaces using bounded planes. As shown in Figure 2.1, it has detected various planes from the scene and used them in various applications.

Figure 2.1 Dense Planar SLAM in action [4]

www.manaraa.com

Dense planar SLAM [4] can also associate same planes registered on different frames. Normally, the whole scene is not covered with a fixed camera's field of view, so we have to move the camera to capture the whole scene. That also comes with one more problem called registration. When we make point cloud using the RGB-D sensor, we also need to integrate different point clouds from different scenes.

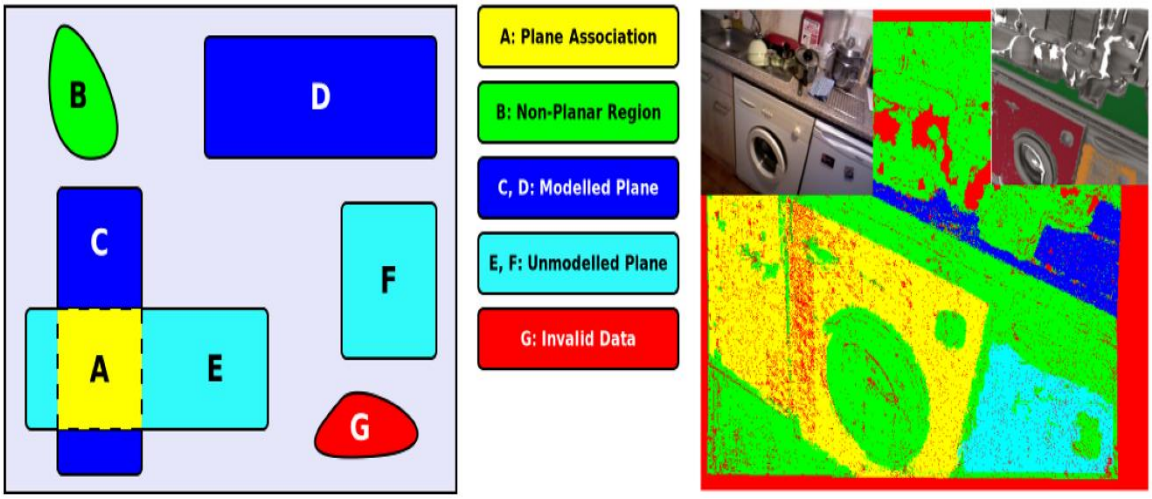


Figure 2.2 Data Association in dense planar SLAM [4]

As shown in Figure 2.2, there are various cases of data association in dense planar SLAM [4]. We will see in later chapters how detecting planes work.

The second important SLAM algorithm is KinectFusion [5], that can reconstruct a 3D scene using Microsoft Kinect depth data. KinectFusion [5] takes raw depth data from the Kinect sensor, estimates pose, predicts surfaces and then reconstruct the whole 3D scene. The generated reconstruction is made out of triangular meshes (also known as a face or tile). Each mesh has a tile index and a normal vector which represents its direction. For each new frame, the algorithm has to find a new pose and integrate reconstruction. KinectFusion [5] can integrate two different reconstructions from two different frames.

www.manaraa.com

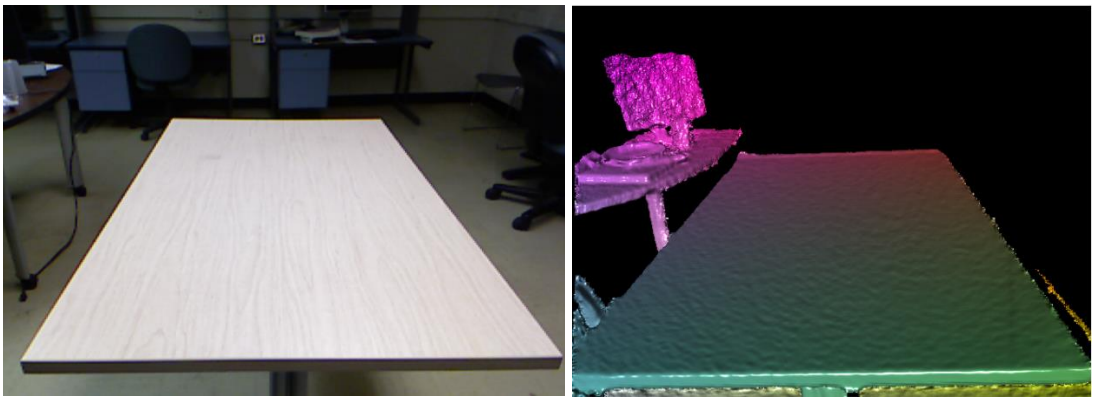


Figure 2.3 KinectFusion [5] in action

Figure 2.3 shows KinectFusion [5] results, the left side is the RGB frame of a 3D scene, and the right side is the output from KinectFusion [5]. Figure 2.4 shows zoom in of the meshes, where triangular meshes are used for 3D reconstruction.

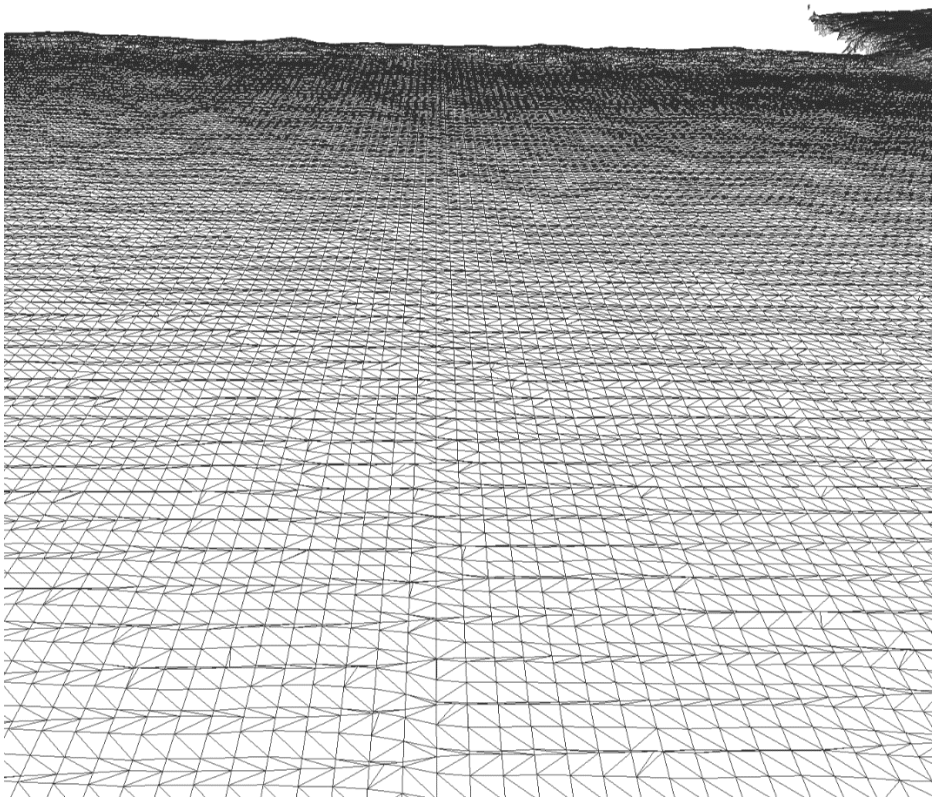


Figure 2.4 Near view of reconstruction

www.manaraa.com

KinectFusion has also been an inspiration for this thesis. We have used the reconstructed triangular meshes concept to generate the high-level search space navigational mesh.

2.2 Search Space

Although there are many different search spaces available to use, depending on implementation needs, we have used navmesh. The reason for this choice, as well as the pros and cons, will be discussed in later chapters. Navmesh was first discussed in a book called AI Game Programming Wisdom [15], followed by Game AI Pro [16] and since its origin, the basic concept of Navmesh has remained unchanged. One more paper [6] describes grid based map generation for robot navigation. They used a normal RGB camera to perceive visual data and proposed an algorithm which can create a grid-based search space. This work is quite different from this thesis as they have used a monocular camera which does not provide the depth information. Furthermore, they have used a grid representation which has many limitations compare to navmesh.

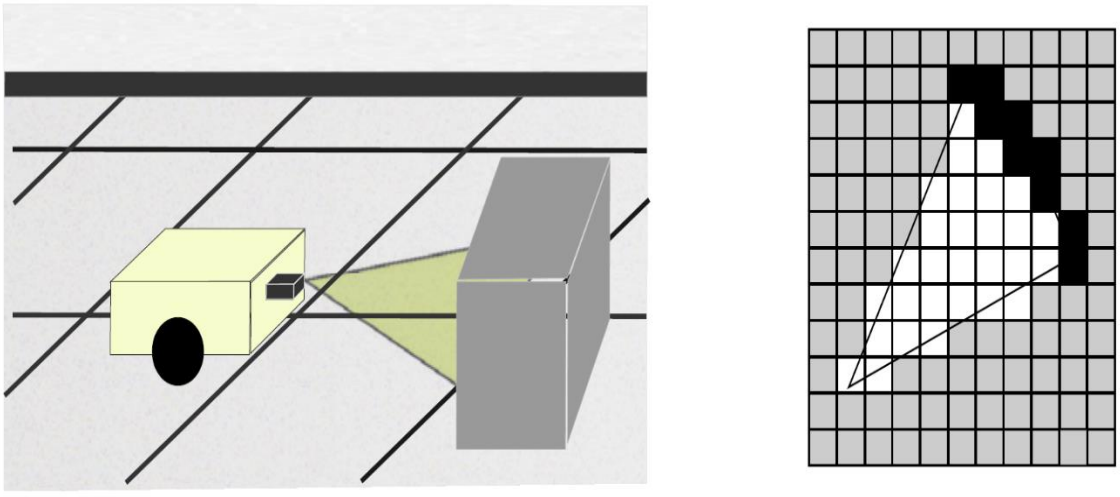


Figure 2.5 RGB camera based grid generation for robots [6]

www.manaraa.com

Xiao and Hao have explained in their paper [17] how pathfinding works in navmesh. Their work is relevant to this thesis because while creating navmesh search space representation, it is important to understand the characteristics of pathfinding used in it.

2.3 Automatic Search Space Generation

There are various techniques and tools available to generate an automatic navmesh, but normally when games are created game designers want to create search space manually. The manual process produces better results compared to automatic but it is quite tedious, as well as a time-consuming process. To remedy this, researchers have designed many new automatic search space generation algorithms.

These algorithms don't have reasoning like humans do so their generated search spaces are not that accurate, but they are good enough to save game designers time and effort. Stuart Golodetz has described an automatic navmesh generation method in his paper [7]. He provides a new approach which is free from the problem of agent's geometry consideration during navmesh generation. He has introduced a basic navmesh generation technique called *Brush Unioning* [7]. Figure 2.6 shows algorithm for brush unioning.

Another paper [9] explains the concept of suboptimal navmesh generation. The approach is called *Automatic Navigation Mesh Generator* (ANavMG) which has introduced convex relaxation. This thesis follows a similar approach to generate a navmesh which does not lie in NP-Hard, as demonstrated by ANavMG. Another paper [8] of the same authors has proposed a near-optimal GPU-based navmesh generation. They have used the 2D abstraction of the 3D game environment to generate a navmesh.

www.manaraa.com

```
function union_all
:   (brushes: Vector<Brush>) → List<Polygon>

var result: List<Polygon>;

// Build a tree for each brush.
var trees: Vector<BSPTree> :=
  map(build_tree, brushes);

// Determine which brushes can interact.
var brushesInteract: Vector<Vector<bool>>;
for each b_i, b_j ∈ brushes
  if j == i then
    brushesInteract(i, j) := false;
  else
    brushesInteract(i, j) := in_range(b_i, b_j);

// Clip each polygon to the tree of each brush
// within range of its own brush.
for each b_i ∈ brushes
  for each f ∈ faces(b_i)
    var fs: List<Polygon> := [f];
    for each b_j ∈ brushes
      if brushesInteract(i, j) then
        fs := clip_polygons(fs, trees(j), i < j);
    result.splice(result.end(), fs);

return result;
```

Figure 2.6 Brush unioning algorithm [7]

The disadvantage of this method is that it cannot generate a navmesh in a continuously changing environment, but it is useful in a static environment. This thesis is influenced by the works of Roman and Nuria [8] [9] as they used 3D object normals and navmesh. We will see more details about 3D normals and their association in later chapters.

ANavMG uses a concept called convexity relaxation which allows system to generate more efficient navmesh. Figure 2.7 shows convexity relaxation concept which is relevant to surface unification used in this thesis.

www.manaraa.com

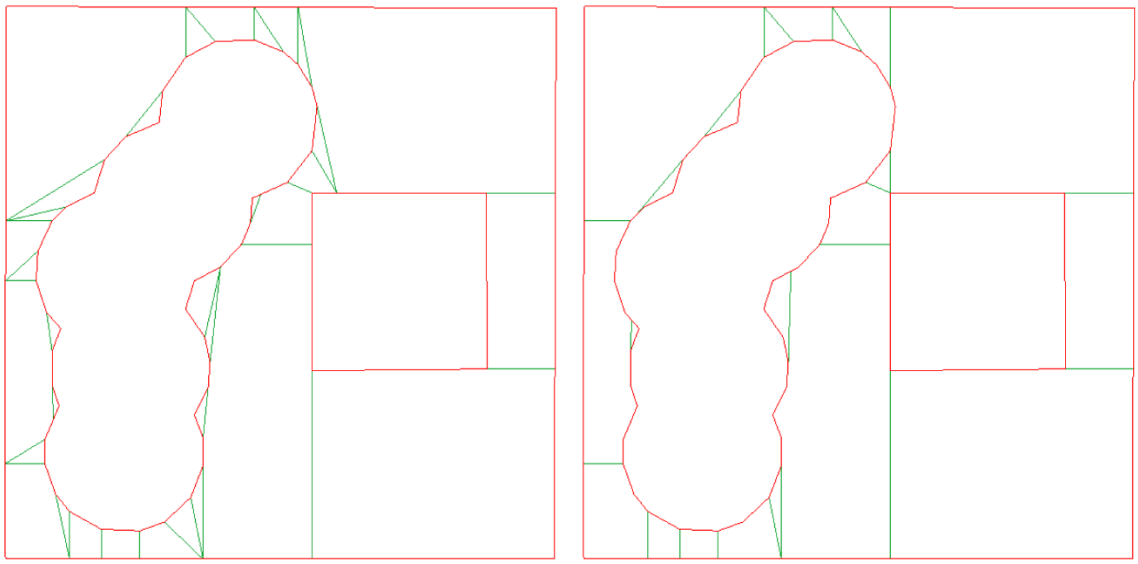


Figure 2.7 Convexity relaxation of ANavMG [8]

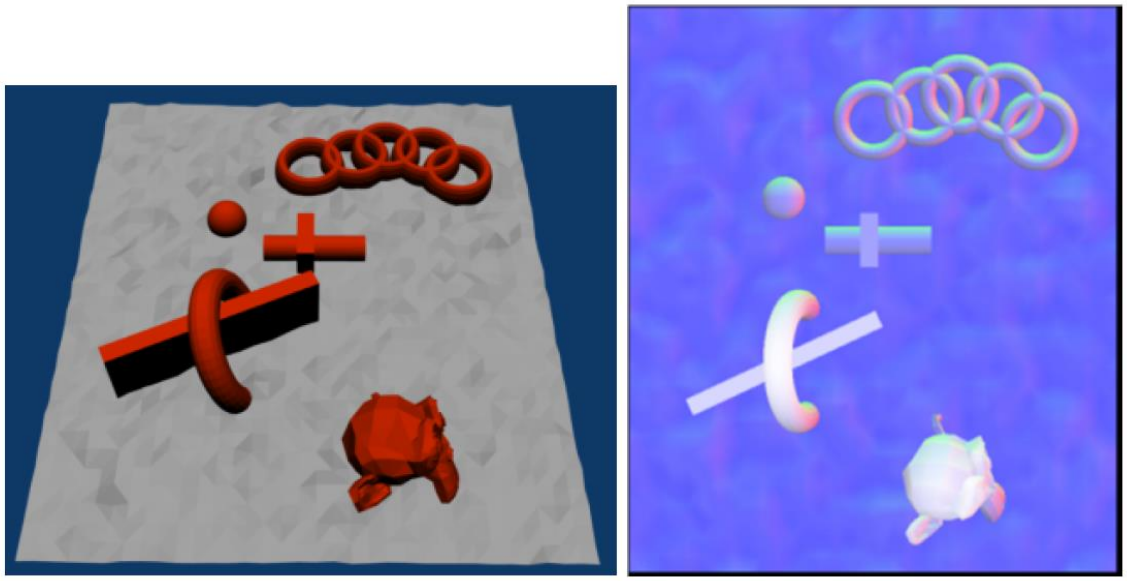


Figure 2.8 2D abstraction of 3D environment [8]

Figure 2.8 shows how 2D abstraction works. The left side of the figure is the 3D scene, and the right side of the image shows its 2D abstraction. Figure 2.10 shows a generated navmesh from 2D abstraction.

www.manaraa.com

ANavMG can work with single layered and multi layered environments. It is more suitable for systems like premade game world. The model and environment used in game world must be without any distortion and need to be smooth. This thesis has introduced a method for rough, on-the-fly generated 3D scene. Figure 2.9 shows flowchart of ANavMG algorithm for single layered environment.

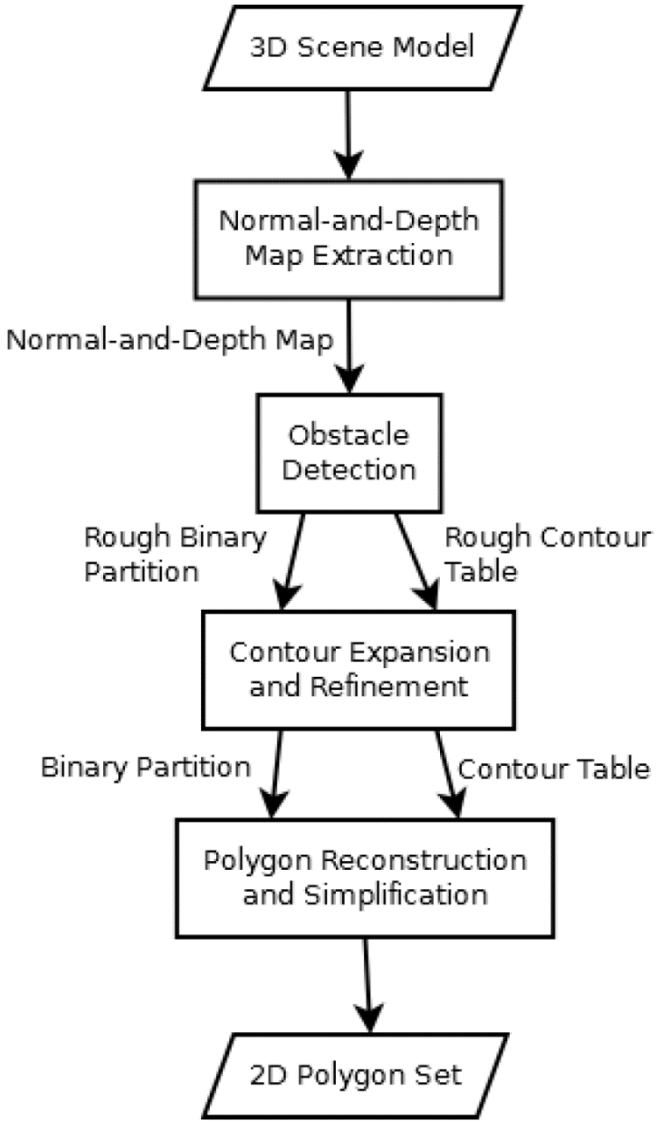


Figure 2.9 ANavMG execution flowchart [9]

www.manaraa.com

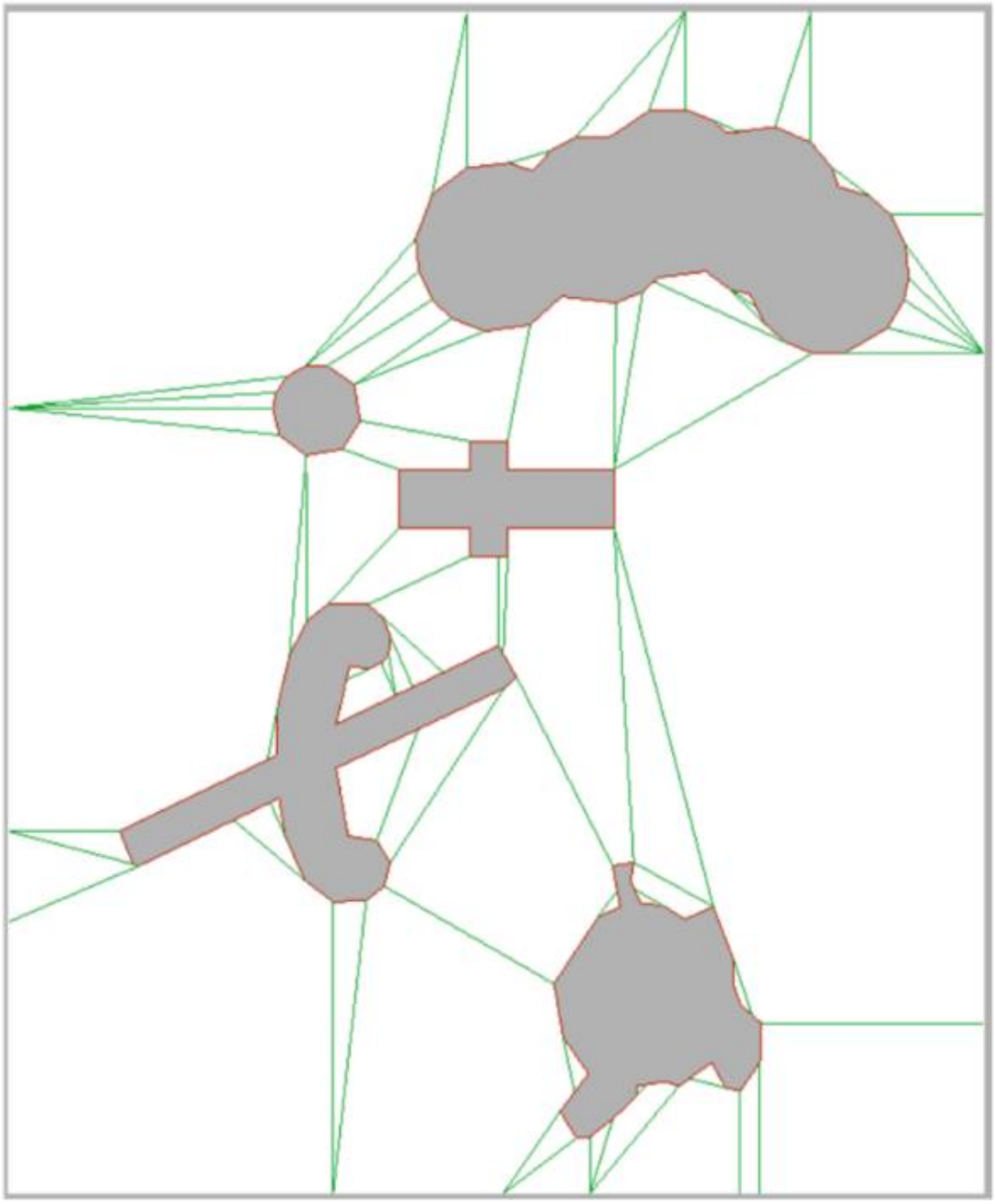


Figure 2.10 Generated navmesh of Figure 2.8 [8]

One more, important algorithm has been introduced by Hale and Youngblood in their paper [10]. This algorithm is called *Adaptive Space Filling Volumes 3D (ASFV3D).* This algorithm seeds world-space with a series of unit cubes and then it automatically

www.manaraa.com

subdivides it, to convert cubes into higher-order polyhedrons. Figure 2.11 shows ASFV3D algorithm.

```
StillGrowing = true ;
/* Populate the world with the initial
   user defined seeds */
if NegativeSpaceRegions.isEmpty() then
    seedWorld();
while StillGrowing do
    StillGrowing = false ;
    for NegativeSpaceRegion in World do
        for Face in NegativeSpaceRegion do
            Face.Translate(Face.Normal);
            if Face.isNotColliding() then
                StillGrowing = true ;
            else
                /* A collision has occurred.
                   First check to see if it
                   possible to increase the
                   order of the polyhedron */
                if Face.isSplittable() then
                    /* Adapt the face to
                       follow the surface it
                       intersected.  Insert an
                       additional face at the
                       point of collision
                       which shares edges with
                       every face that
                       contained the vertices
                       that are being split.
                       */
                    Face.SplitPoint();
                    /* Lock the growth of the
                       newly created vertices
                       to lie on the equation
                       of the plane they
                       intersected */
                    Face.ConstrainPoint();
                    StillGrowing = true ;
                else
                    /* The collision cannot be
                       handled by splitting */
                    Face.Translate(-1 *
                    Face.Normal);
                    Face.canGrow = false;
List seedPoints = new List ;
/* Find places to place new Seeds in the
   world */
seedPoints.Append(World.Seed());
if not seedPoints.isEmpty() then
    /* Restart growth algorithm on the new
       points */
    ASFV3D(seedPoints);
/* Run combining and clean up procedures
   */
World.combineConvexShapes();
World.removeColinearPoints();
World.RemoveDegenerateFaces();
```

Figure 2.11 ASFV3D algorithm [10]

www.manaraa.com

CHAPTER 3

Concept Description

3.1 Navmesh as a Search Space

Navmesh is a type of search space representation, widely used in game AI and pathfinding applications. It is made of interconnected convex polygons. Each polygon in navmesh is sometimes called a tile or face. Let's assume we have a 2D environment with few obstacles in it. Figure 3.1 shows an example of a 2D environment with simple navmesh where black blocks are obstacles and black lines are boundaries of navmesh cells.

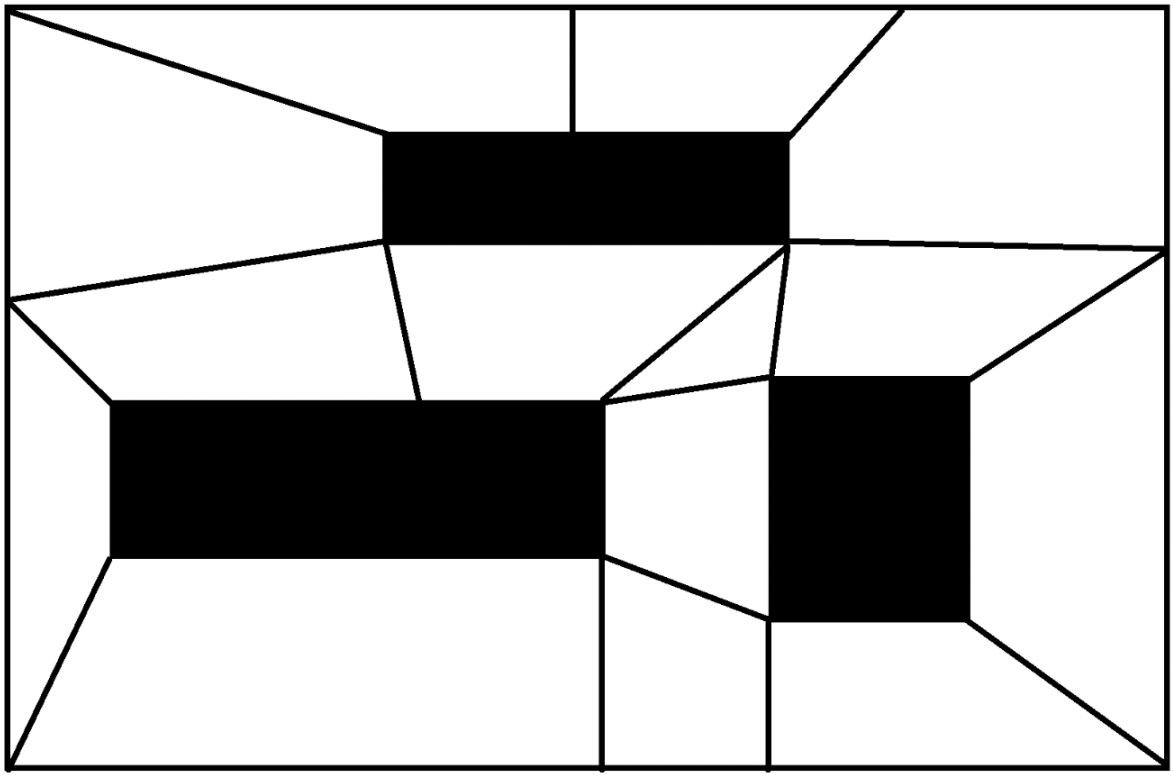


Figure 3.1 Navmesh example with obstacles

www.manaraa.com

As we know, pathfinding algorithms are graph search algorithms which require a graph as an argument to the algorithm so this visual representation must break down into a graph for calculation.

There are numerous ways to convert a navmesh into a graph. For example, denoting a tile as a node, edge as a node, etc. We have used an approach called *edge as a node*. Each polygon in navmesh has an edge, which is considered as a connection between two polygons. The edge as a node approach takes a midpoint of an edge and denotes it as a node on the graph. Nodes are connected with each other in the same way as navmesh tiles are connected.

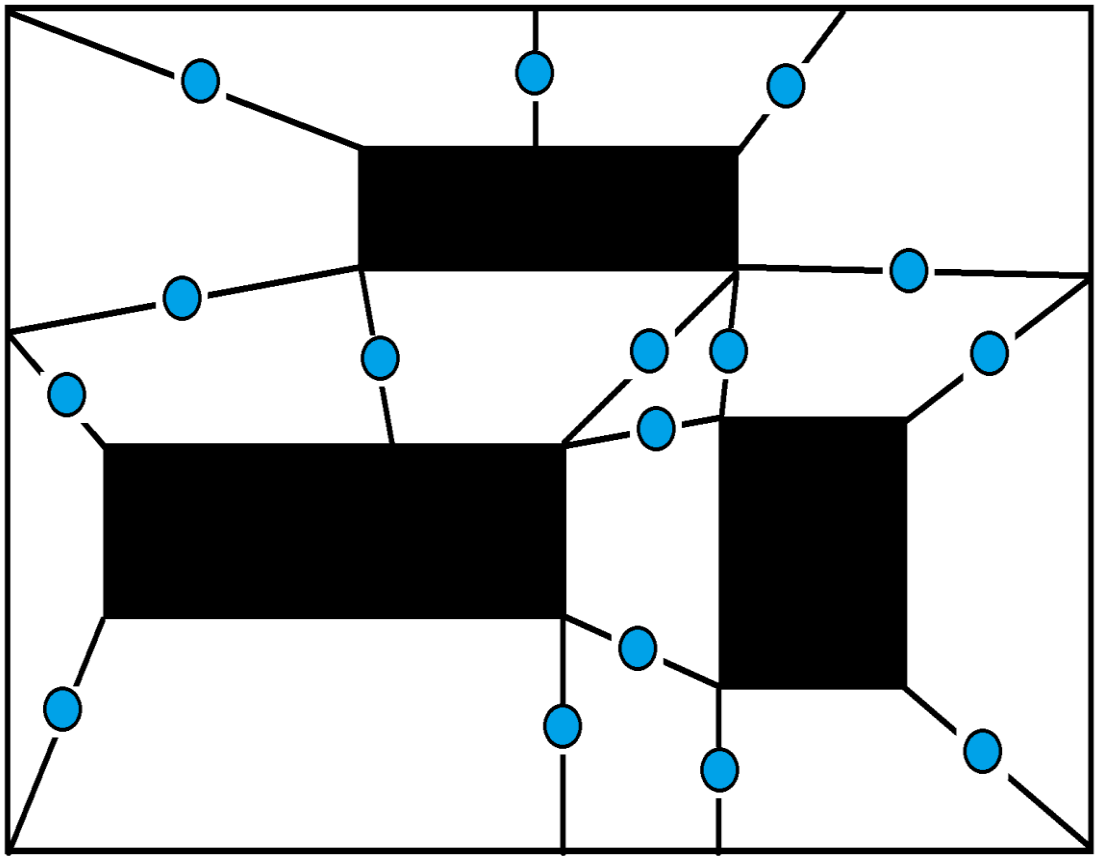


Figure 3.2 Edges as a node in Navmesh

www.manaraa.com

Figure 3.2 shows the edge as a node approach where blue dots represent each node on the graph. An *agent* is a 3D or 2D model that will follow the path after pathfinding is applied. The agent can move anywhere inside the convex polygon. Pathfinding is required to find a path across different polygons. The method used for movement inside a polygon is known as *ray casting*. According to the ray casting method, if we put a light bulb in any place inside the polygon, the ray originating from the bulb will reach every possible place inside the polygon. That is the main reason why we have used convex polygons instead of non-convex polygons. During that free movement inside each polygon, the agent doesn't need pathfinding mechanism. Unlike grid search space, navmesh representation requires both mesh and graph for pathfinding calculation. Normally, when graph representation boils down to the graph, the grid is not required for pathfinding.

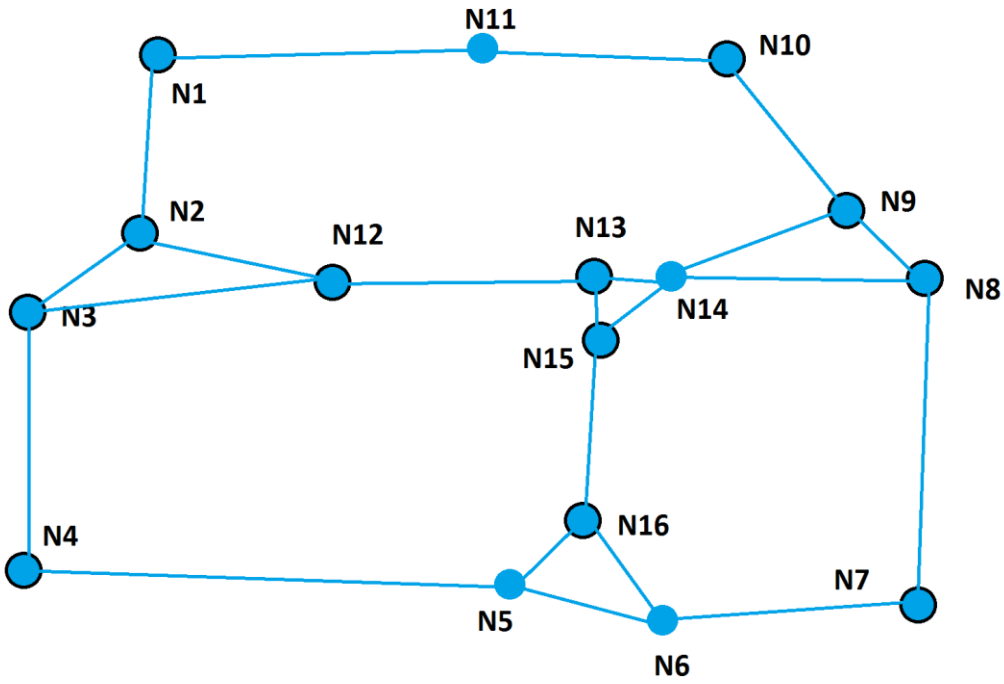


Figure 3.3 Graph representation of navmesh

www.manaraa.com

Figure 3.3 represents a graph made out of navmesh. Neither graph nor navmesh is visible to users; it is just a logical layer for pathfinding. We will see more details about it in later topics. There are a couple of methods to represent graphs, for example, edge list, adjacency matrices, etc. We have used adjacency list for this thesis. Table 3.1 shows the adjacency list of navmesh graph of Figure 3.3.

N1	->	N2, N11
N2	->	N1, N3, N12
N3	->	N2, N4, N12
N4	->	N3, N5
N5	->	N4, N6, N16
N6	->	N5, N7, N16
N7	->	N6, N8
N8	->	N7, N9, N14
N9	->	N8, N10, N14
N10	->	N9, N11
N11	->	N1, N10
N12	->	N2, N3, N13
N13	->	N12, N14, N15
N14	->	N8, N9, N13, N15
N15	->	N13, N14, N16
N16	->	N5, N6, N15

Table 3.1 Adjacency list of navmesh

www.manaraa.com

There are a few reasons why navmesh is more popular than any other search space. The navmesh requires less memory for storage and it is also light on pathfinding computation when compared to grid and waypoint. Unlike grid, it provides best-area coverage of the game environment. The proposed method in this thesis will generate a search space as an adjacency list as an output.

3.2 Point Cloud

The point cloud is a way of representation for 3D scenes which is made up of 3D points. Points in point cloud are defined as three parameters that show its x, y and z-axis coordinates in 3D space.

$$P = [x, y, z]$$

Many different techniques can generate the point cloud but in this thesis, we have used RGB-D Microsoft Kinect Sensor.

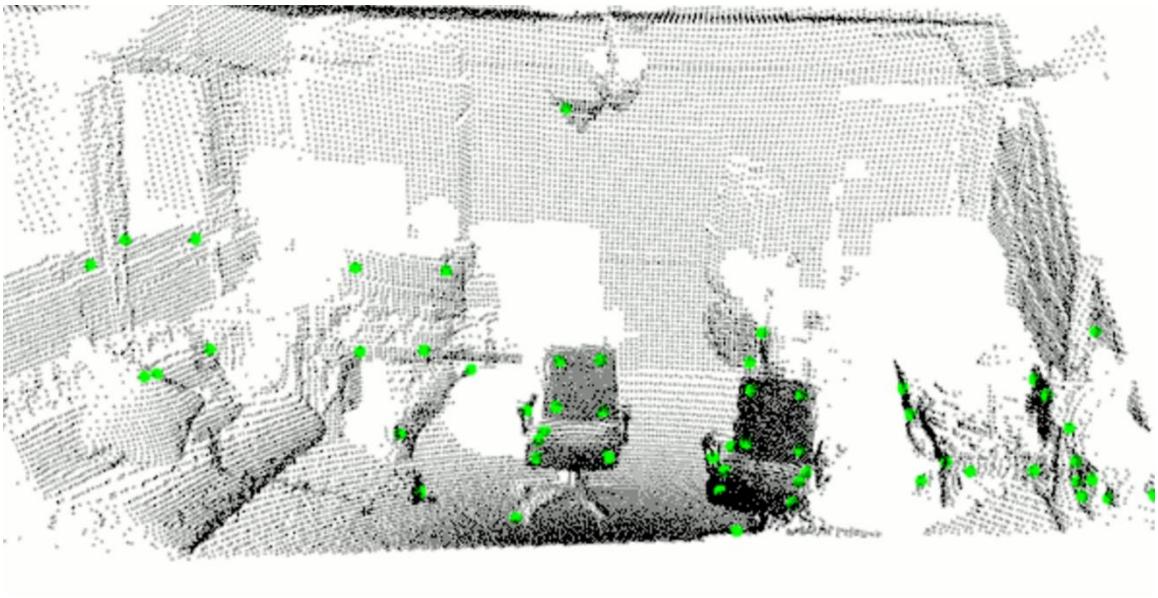


Figure 3.4 Point cloud of a 3D room scene [11]

www.manaraa.com

Normally, the point cloud is measured considering the camera as a point of origin. So, the camera coordinates are [0, 0, 0] and rest of the points in point cloud is mapped accordingly. The point cloud visualization system works somewhat different compared to point cloud recording system.

The visualization contains two camera points, first [0, 0, 0] which is used for recording the point cloud, and the second is used for viewing through the camera, which changes its coordinates according to the spectator's needs. Point clouds can be classified into two categories, *sparse point cloud* and *dense point cloud*. We have used dense point cloud in this thesis because of its accuracy and effectiveness despite its high cost and complexity.

Sparse point cloud has a very low density of points in a particular space. The well-known approach which generates sparse point cloud is parallel tracking and mapping [18]. Dense point cloud has very high number of points in a particular space. Normally, the sparse point cloud is created using RGB (monochrome or color camera) sensors, and dense point cloud is created using RGB-D (depth sensor like Microsoft Kinect) sensors.

The more points we have, the better navmesh we can generate. It is a trade-off between density and accuracy. The main idea behind this research is to reduce the number of meshes in recent 3D reconstruction algorithms. If we use low-density point cloud SLAM algorithms, we will have a low number of generated polygons, but it will also produce less accurate navmesh.

The monocular SLAM also has low accuracy for determining points in point cloud. Hence, augmented reality experience with monocular SLAM is not smooth and immersive.

www.manaraa.com

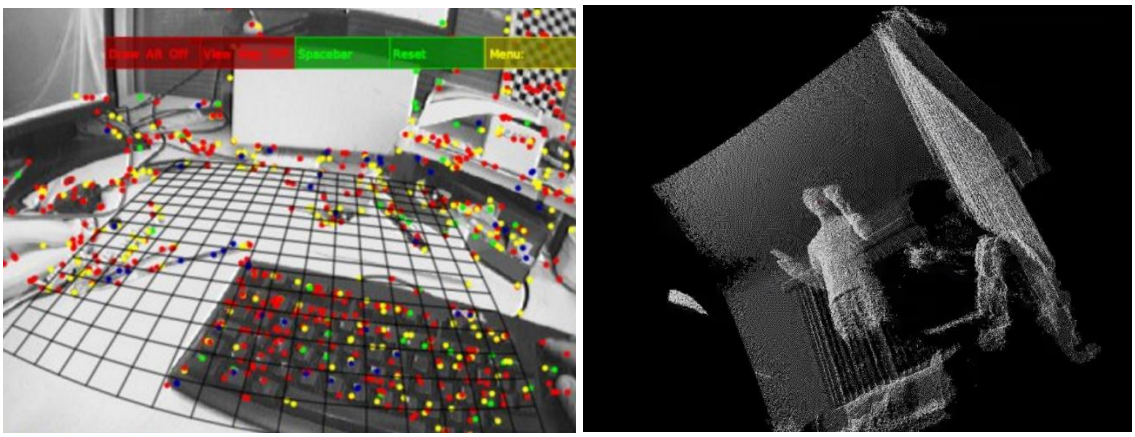


Figure 3.5 [Left] Sparse [Right] Dense point cloud

According to the pinhole camera model, a point [x, y, z] of the 3D space (real world coordinates) can be determined using two points [x1, y1] and [x2, y2] of 2D space (image) using triangulation method. If we follow this, then it will take a large of time and CPU cycles to produce a point cloud of a small scene. Hence, we have used Kinect sensor which provides us with the depth of a particular point in the frame and thus we can build point cloud easily.

3.2.1 Normal of Point in Point Cloud

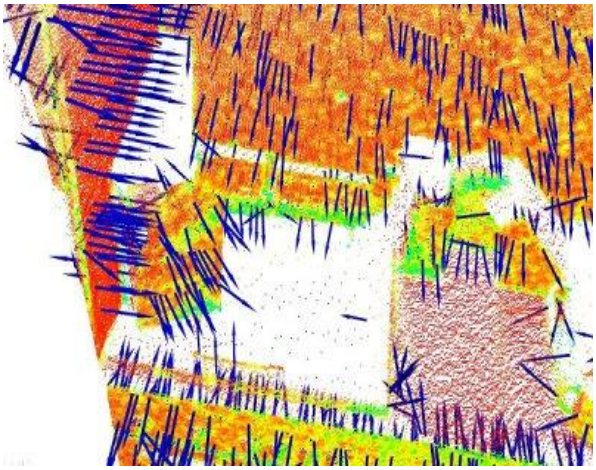


Figure 3.6 Normal of point in point cloud using PCL [11]

www.manaraa.com

A point normal is also referred to as a surface normal because it shows the orientation of the surface where the point resides on. As described in point cloud library documentation [11], finding the normal of a point is approximating normal of a surface tangent plane. So, basically the normal of a point is described as,

$$\vec{N} = a\hat{\imath} + b\hat{\jmath} + c\hat{k} \quad \text{OR} \quad \vec{N} = [a, b, c]$$

Point normal shows the direction of the tangent to the surface where that point is located. The same concept can be applied to surface normal which we will see in the next few topics. The surface normal is more important to this thesis as we have used surface normals in surface unification process. The determination of surface normal and surface construction is not a part of this thesis. We have used the KinectFusion [5] algorithm to generate surface reconstruction and surface normal. It is provided in the form of a one or two-dimensional array with the surface indices. In this thesis, we have used point cloud library to determine normal of the mesh surface and KinectFusion for surface reconstruction, and reconstructed surface normal prediction.

3.3 Search Space as a Logical Layer

In game AI, the physical layer is something you can interact with. Although computer games are virtual, and it does not have existence in the real world because they only exist in computer memory. The physical layer of the game involves an agent that can also see and interact with the game environment. For example, 3D or 2D models used in games. On the other hand, the logical layer is something the agent cannot see or interact with. For example, search space representation. Even though the logical layer exists with the

www.manaraa.com

physical layer, it is only used in computation. The physical layer provides the context for agents and the logical layer provides the context for computation.

As shown on figure 3.7, the game world is the base and the rest of the layers are set on top of it. Wherever game designers want AI interaction, they have to implement a logical layer. In this thesis, we have worked on a logical layer of this aspect, especially the graph representation.

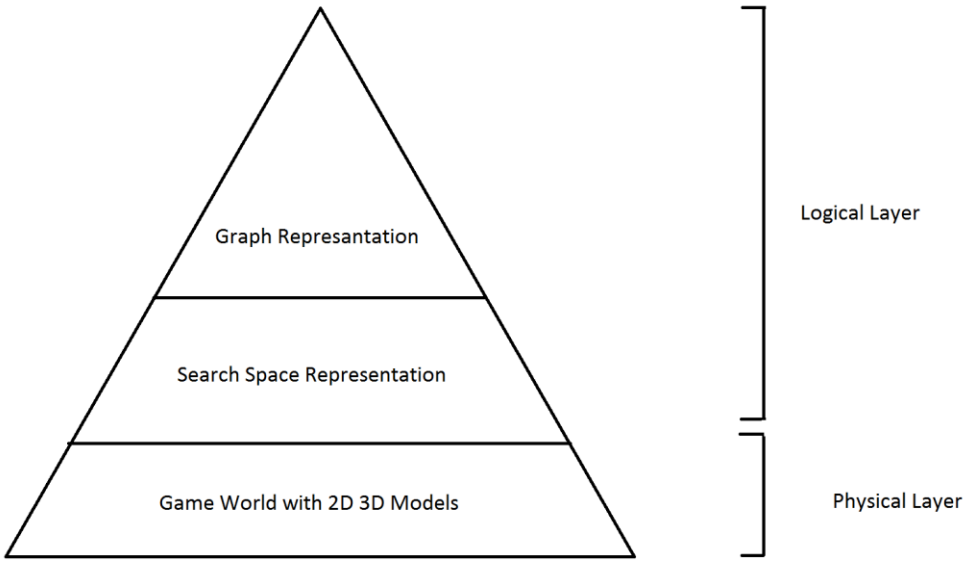


Figure 3.7 Hierarchy of different layers

The logical layer is also made of two parts, logical representation layer, and logical computational layer.

The logical representation layer is used for representation and understanding, such as search space. In some cases, search spaces are not used in the logical computational layer. For example, in grid search space, the actual grid representation is used only for game designers to give them a better understanding of the environment, but its logical

www.manaraa.com

computation layer is only used for game AI computation. The logical representation and computation layer of navmesh are both used during game AI computation. These layers are merged to create a whole game and pathfinding experience.

www.manaraa.com

CHAPTER 4

Thesis Approach and Explanation

4.1 Surface Unification

The 3D reconstruction algorithms like KinectFusion [5] builds a surface of the 3D scene. The surface unification process can be applied to those surfaces that are built up. The problem with 3D reconstruction algorithms is that, its output is not suitable for game AI pathfinding. It generates a dense mesh from a small 3D scene. That much detail is unnecessary as a search space in pathfinding. Surface unification solves this problem and generates a less dense navmesh. The next few paragraphs discuss some parts of Microsoft Kinect, but for more details, refer to chapter 5.

4.1.1 Different Surface Types

The KinectFusion [5] has generated the meshed 3D reconstructed surfaces. The generation of navmesh depends on surface and surface normal. As discussed in the previous topic, surface normal is derived using point cloud library [11]. For example, if a surface plane (which is also a small mesh in a large mesh network) *S1* has normal vector *N1* and surface plane *S2* has normal vector *N2*.

$$\text{For surface S1, } \overrightarrow{N1} = a1\hat{\imath} + b1\hat{\jmath} + c1\hat{k}$$

$$\text{For surface S2, } \overrightarrow{N2} = a2\hat{\imath} + b2\hat{\jmath} + c2\hat{k}$$

The angle between two surface normal vectors is defined as the *Angle of Surface Unification*. The angle of surface unification is different for every single consecutive pair

www.manaraa.com

of meshes [19] in 3D reconstruction. The angle of surface unification between two normal vectors is defined as,

$$\emptyset = \cos^{-1}\left(\frac{\overrightarrow{N1} \,.\, \overrightarrow{N2}}{\left\|\overrightarrow{N1}\right\| \,.\, \left\|\overrightarrow{N2}\right\|}\right)$$

Where, $\overrightarrow{N1} \,.\, \overrightarrow{N2} = a1.a2 + b1.b2 + c1.c2$

$$\left\|\overrightarrow{N1}\right\| = \sqrt{a1^2 + b1^2 + c1^2}$$

$$\left\|\overrightarrow{N2}\right\| = \sqrt{a2^2 + b2^2 + c2^2}$$

The angle of surface unification $\emptyset$ is discussed more in the *Experiment and Result* section but for now, it is important to look at different cases of surface orientation.

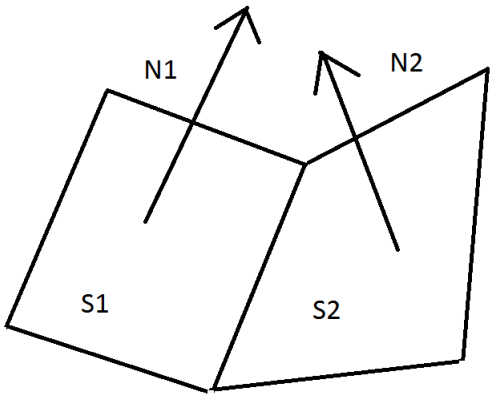


Figure 4.1 Concave surface orientation

There can be three possible surface orientations: Concave, convex and flat. The concave orientation has nothing to do with a convex polygon of mesh in navmesh; it is just a type of surface structure and orientation relative to other meshes. As shown in Figure 4.1, the

www.manaraa.com

angle of surface unification generated by a concave surface orientation will be between 0 and 180 degrees.

$$0° < \emptyset_{concave} \leq 180°$$

As shown in Figure 4.2, the angle of surface unification generated by a convex surface will also be between 0 and 180.

$$0° < \emptyset_{convex} \leq 180°$$

Concave and convex surfaces are handled in different ways which we will examine in the next few topics.

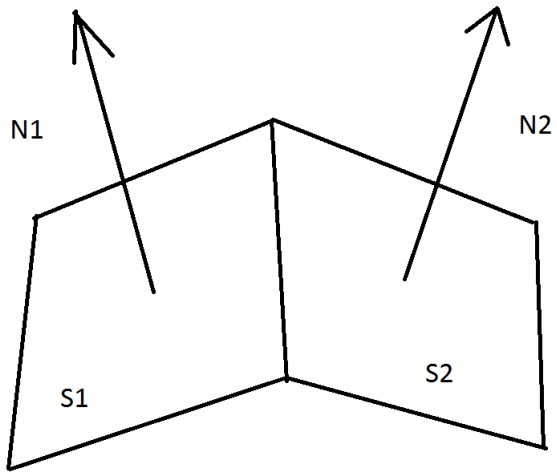


Figure 4.2 Convex surface orientation

Surface Unification algorithm is required for all kinds of surface orientations, but a flat surface is directly eligible for the convex test. The flat surface has 0 angle of surface unification which means it is ready for unification and can be directly transferred for the convex test. As shown in figure 4.3, the angle of surface unification is 0.

www.manaraa.com

$$\emptyset_{flat} = 0$$

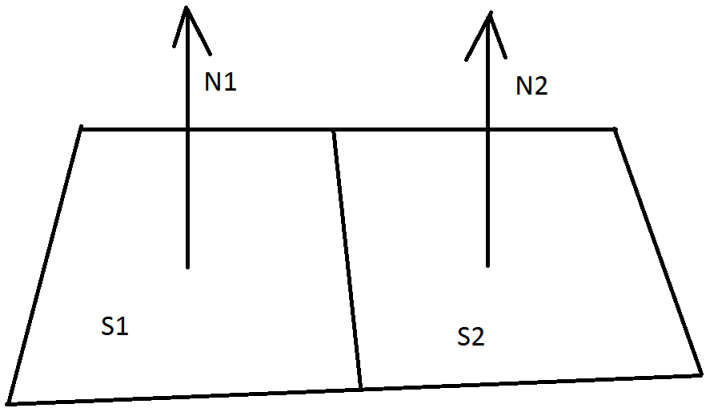


Figure 4.3 Flat surface orientation

4.1.2 Surface Unification without Registration

Surface unification algorithms are divided into two different categories: *With registration,* and *without registration,* but we have focused more on the without registration case. Microsoft Kinect gives depth data as a frame. Each pixel in the frame represents the depth of a particular pixel related to its RGB frame. We have constructed the point cloud frame using Microsoft Kinect depth data. We will see more details about depth data, Microsoft Kinect and point cloud generation in the *Experiment and Result* section. In this method, registration is not required, so surface unification is run on every frame. To compute surface unification on every single frame is not feasible so it is used as a base concept and sometimes used during research work. As shown in Figure 4.4, point cloud frame is a 480 * 640 * 3, 3-dimentional array, which holds x, y and z coordinates of each frame pixel respectively. We have used 640 * 480-pixel frame from

www.manaraa.com

Microsoft Kinect, if it is in another resolution, then point cloud frame is made of that dimension. The dimension does not matter as they simply show the size of the array.

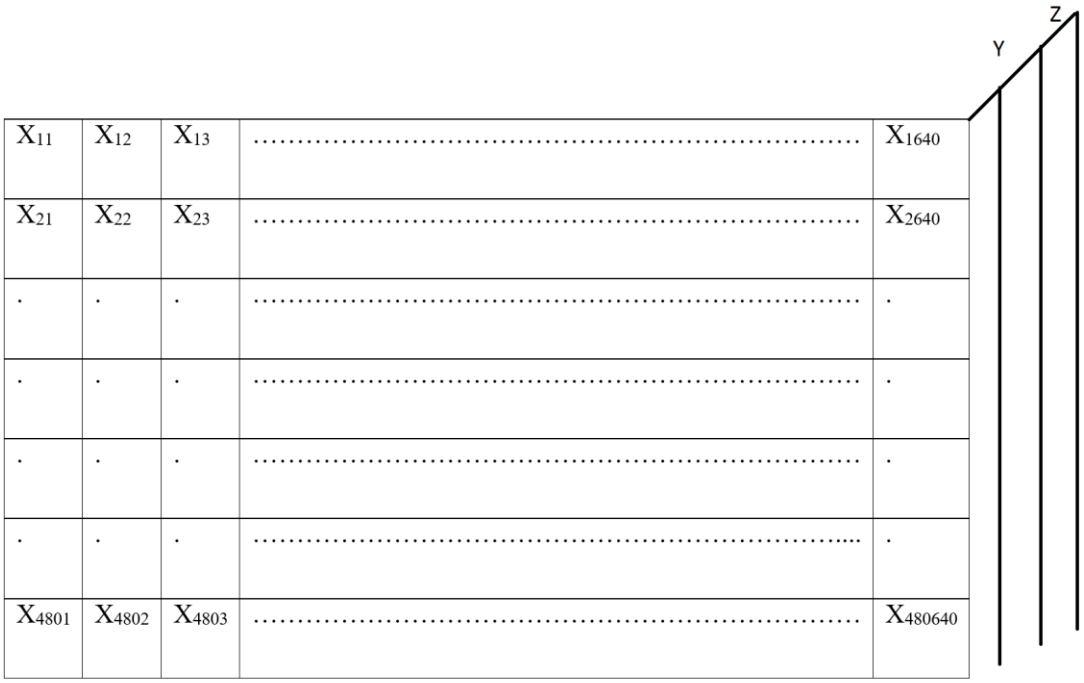


Figure 4.4 Point cloud frame

The format of point cloud frame is described below; each index holds a depth in millimeter.

$$X_{row\ column}$$

KinectFusion [5] used Microsoft Kinect depth data to build point cloud then to build a 3-dimensional surface reconstruction. We have used depth data to build point cloud and passed this point cloud to KinectFusion. Before reconstruction we extract each surface reconstruction frame for the surface unification process. We do not need full reconstruction as it is similar to the physical layer of search space. The array index is used as an index of each point in the point cloud. Figure 4.5 shows the intermediate

www.manaraa.com

output of KinectFusion [5]. The dimension of each surface reconstruction frame is also the same as each respective point cloud frame without registration case and hence, it is also 480 * 640 pixels. The surface reconstruction frame is only a two-dimensional array as it is just showing the mesh of a particular scene frame. KinectFusion generates only a three-vertex triangular mesh. As you can see from Figure 4.5, a particular number is repeated only three times. Note that this is not an actual reconstruction frame and is only being used as an example.

1	1	2	……	136
1	2	2	……	136
.	.	.	……	.
.	.	.	……	.
.	.	.	……	.
.	.	.	……	.
125	125	124	……	229

Figure 4.5 Surface reconstruction frame from KinectFusion [5]

The number on each cell shows the index of the polygon, and as it is a triangular polygon, three vertices are part of each polygon. For example, index (1, 1), (1, 2) and (2, 1) shows a polygon with index 1, and it is located at location (1, 1), (1, 2) and (2, 1) on the frame. This can only be possible without the registration process as it processes each frame during surface unification.

www.manaraa.com

Figure 4.6 shows, the surface normal of each polygon. The surface normal collection is a two-dimensional array in which the first column shows an index of each mesh and the consecutive three columns show the x, y and z components of the normal vector. Again, the numbers used in this image are only for example purposes.

1	5	2	3
2	2	2	2
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
158	8	6	4

Figure 4.6 Surface normal collection

In the surface reconstruction frame, each cell has a list of participating points from point cloud in mesh building. That list is sorted and represents indices of the polygon. We have converted the surface reconstruction frame in such a way that each pixel in the surface reconstruction frame is matched with exactly the same pixel in point cloud frame. The surface unification algorithm uses that list of participating points and its corresponding polygon indices to unify the surface. The intermediate list of participant polygon indices can have a minimum of one element and maximum as much as needed. There is a simple process for extracting comma separated values which has been implemented.

www.manaraa.com

The surface reconstruction frame and surface normal collection are very important for the surface unification process. Below is the algorithm for surface unification without registration frame.

```
Algorithm SurfaceUnification
     Input: double surfacerecon[][], surfacenorm[][]
     Output: surfaceunify[][] and surfacenormunify[][]

inter[]
while i < surface_unification_index do
     for j = 1 to 480 step 1 do
      for k = 1 to 640 step 1 do
       inter <- list_extract(surfacerecon[j][k])
       if inter.length == 1 then
        surfaceunify[j][k] <- inter[0]
        surfacenormunify[j][k] <- find(inter[0],
        surfacenorm)
       else
       for d = 0 to inter.length step 1 do
        if Ø(inter[d], inter[d+1]) < Ø_test, Ø_obstacle
         nextindex <- mesh.next
         surfaceunify[j][k] <- nextindex
         surfacenormunify[nextindex][] <-
         average(find(inter[d],surfacenorm),
         find(inter[d+1],surfacenorm))
        else
         surfaceunify[j][k] <- inter[d]
         surfacenormunify[nextindex][] <-
         find(inter[d],surfacenorm)
        end if
       end if
       end for
      end for
     end for
end do
```

The single dimensional array named *inter* used in the algorithm is responsible for holding the intermediate list of participant polygon indices. The two-dimensional arrays surfacerecon and surfacenorm holds the surface reconstruction frame and reconstructed surface normal respectively. This algorithm gives two two-dimensional arrays named

www.manaraa.com

surfaceunify and surfacenormunify which holds new unified logical surface structure. It holds only logical level information because it does not have any relation with the original surfacerecon and surfacenorm arrays, and it also does not interact with physical layer arrangements.

The $\emptyset_{function}$ is a function which represents angle of surface unification. The $\emptyset_{test}$ is determined after a couple of experiments and testing and will be discussed more in the next chapter. The angle of surface unification function takes two vectors as arguments and returns the angle between them in degrees. There is one more function that is used called *average*, which takes two vectors as an argument and returns their average.

One of the important concepts we have used in the surface unification algorithm is surface unification index. Surface unification index simply shows the number of times we need to run the surface unification procedure. If the surface unification index is 1, then the surface unification procedure only runs once.

As we are combining polygons, it is not guaranteed that the resulting polygon will remain convex after the unification process. Hence, we run a convex test to check that the polygon will remain convex, but as we keep discarding on bases of the convex test, we cannot merge enough polygons to generate meaningful search space. To eliminate this problem, we have introduced surface unification index.

It might be possible that the updated polygon in surfaceunify metrics can merge with another neighboring updated polygon. After each iteration of the surface unification process, the number of polygons in the scene will be reduced. It is also not recommended to put a very high number as the surface unification index because as we are merging and

www.manaraa.com

averaging surface normal vectors, it is quite possible that the number of polygons is so reduced that pathfinding will not be effective anymore. It is important to find a balance between the surface unification index and the pathfinding approach in generated navmesh.

The worst case time complexity of this algorithm is O (jkd), where j represents the height of the surface reconstruction array, k is the width of the surface reconstruction array and d is the length of the intermediate list of participant polygon indices array. The j and k are part of the first nested loop which parse 2-dimentional array and d is counter for one dimensional array. Normally, $j \neq$ k and $d \lll j, k$.

4.1.3 Surface Unification with Registration

Registration is a process of combining a two-point cloud frame into a one-point cloud frame.

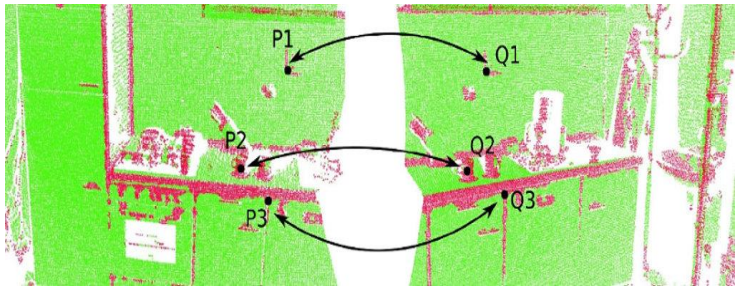


Figure 4.7 Registration using point cloud library [11]

As shown in Figure 4.7, similar points in different frames are recognized and then combined in same point cloud. Registration algorithms combine each frame data into a single set of separate data store because when frames update the content, it overwrites on top of old data. Surface unification process changes in a dramatic way when we consider registration because we cannot process raw frames in the case with registration.

www.manaraa.com

The first major change will come to point cloud, as there will be no separate point cloud frame available. It will all combine to one single file which has an ordered list of all points in point cloud with its respective x, y and z coordinates. The surface reconstruction is also built accordingly. We have not used this approach in our thesis, but it is important to mention as a future work.

4.2 Convex Test

The convex polygon is a polygon having all internal angles less than 180 degrees. If it does not, then it is called as a non-convex polygon. As we have seen previously, navmesh needs all of its polygons to be convex.

The surface unification algorithm tries to reduce the number of polygons in three-dimensional reconstruction and makes it more suitable for search space generation. During the unification process, similar or less than the threshold angle of surface unification polygons are combined and merged into a single polygon.

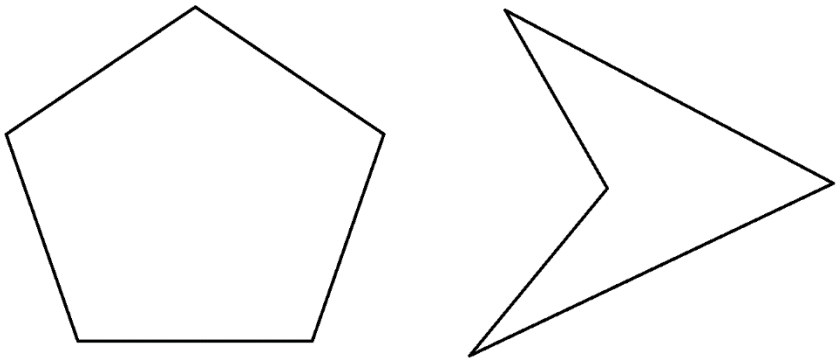


Figure 4.8 [Left] convex polygon [Right] non-convex polygon

www.manaraa.com

The surface unification algorithms guarantee the unification of similar or threshold meshes but it does not guarantee that every polygon is convex. It is required to run convex tests before confirming the result of surface unification. This test ensures that a newly unified polygon is convex, otherwise it will discard the unification and fall back to the previous stage.

```
Algorithm ConvexTest
     Input: double surfacerecon[][], pointcloud[][],
     surfaceunify[][]
     Output: boolean value showing true or flase
check <- convex_test_value
check_arr[][]
Point a[][]
same <- true; positive, negative
for j = 0 to 480 step 1 do
 for k = 0 to 640 step 1 do
  check_arr = find(check, surfacerecon[][])
 end for
end for
for x = 0 to check_arr.length step 1 do
 a = findcoordinate(check_arr[x][0],check_arr[x][1])
end for
while allpoint ∈ a
for y = 0 to a.length step 1 do
 cx1 = a[y+1][1] - a[y][1]
 cx2 = a[y+1][2] - a[y][2]
 cx3 = a[y+2][1] - a[y+1][1]
 cx4 = a[y+2][2] - a[y+1][2]
 product = cx1*cx4 - cx2*cx4
 if product < 0
     negative  = true
 else if product > 0
     positive = true
 if positive && negative = true
     return false
 end if
return true
end do
```

The convex test algorithm is the combination of convex test and data extraction. The first half of the algorithm works as a search agent and finds desired values in surfacerecon,

www.manaraa.com

pointcloud, and surfaceunify matrices. The method called *find* will find the desired indices of mesh in the surfacerecon matrix with the given convex test value. For example, if someone wants to check mesh 4, then convex test value will become 4 and pass it to *find* and *find* will return all the occurrences of mesh 4 in surfacerecon. The *findcoordinate* procedure finds a three-dimensional point in the pointcloud matrix related to its mesh index returned by the *find* method. The convex test checks for every consecutive pair of three points of the polygon. The convex test returns Boolean values which are processed by the surface unification procedure. The implementation of the convex test in this thesis is somewhat greedy from the algorithmic perspective. If the immediate result of convex test is false, then the surface unification process discards the changes. It is quite possible to wait until a few more polygons are merged in and it may become convex. The worst case time complexity of convex test is $O(jk)$ where j represents the height of the surface reconstruction array, k is the width of the surface reconstruction array. The j and k are part of the first nested loop which parse 2-dimentional array where $j \neq$ k.

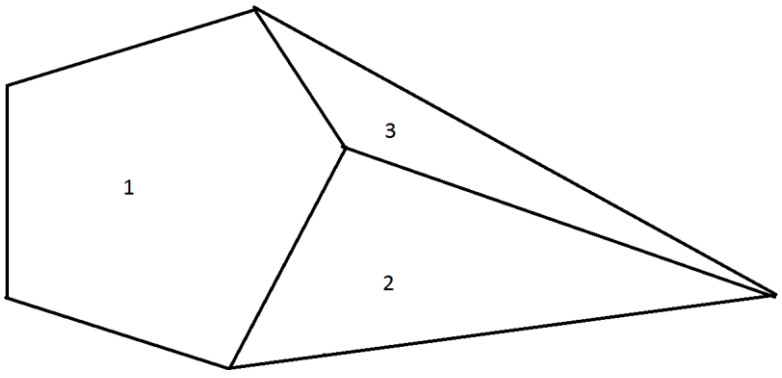


Figure 4.9 Greedy behavior of convex test

www.manaraa.com

As shown in Figure 4.9, three different polygons are connected with each other, and each polygon is convex by itself.

According to surface unification and the convex test, both cases shown in Figure 4.10 will fail as the resulting polygon is not convex. The example shows a special case of unification where convex test's greedy behavior stands out. It might be possible that this case would not come during unification.

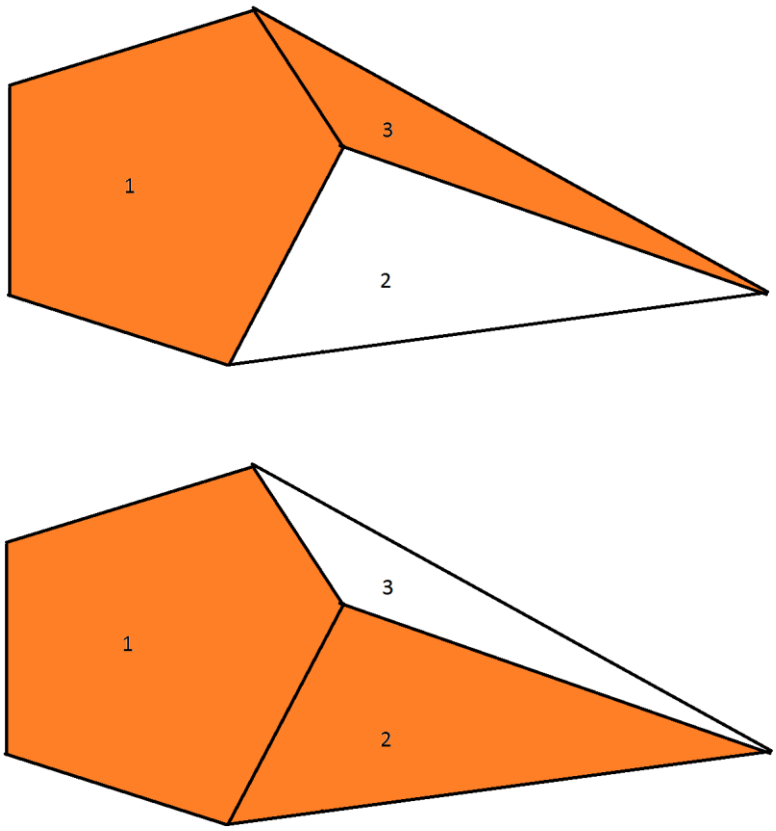


Figure 4.10 Left: Mesh 1 & 3 Merge Right: Mesh 1 & 2 Merge

The resulting polygon after the unification of all three polygons is also convex as shown in Figure 4.11.

www.manaraa.com

The convex test will not wait until all of the polygons merge because it is called on every polygon unification. It is like a tradeoff between efficiency and getting the best result. If it waits for a certain time, then the process will become very time-consuming, and if it doesn't, then we might not get the best result. It is up to the developers of the process to decide how it should be implemented.

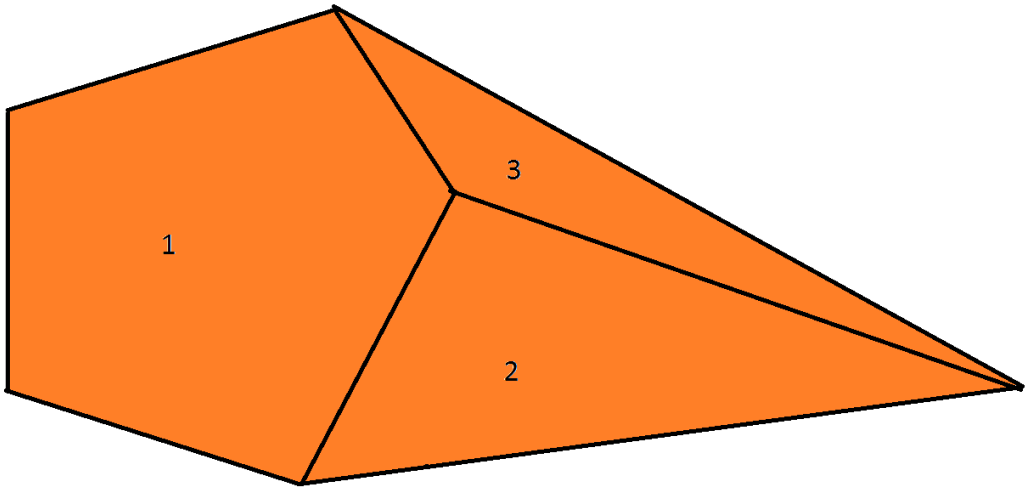


Figure 4.11 Convex polygon after surface unification

4.3 Search Space Generation

The last key substance of this thesis is the search space generation procedure which makes the whole thesis meaningful and relevant. The procedures developed to this point are mainly providing a high level framework to do search space generation calculation. Search space generation calculation will produce an adjacency list or adjacency matrix, which is one form of graphs, which in turn are a low-level representation of search space. Essentially, an adjacency list is a space efficient form of the adjacency matrix, so it does not matter from a research perspective what the output is, but it may affect

www.manaraa.com

implementation, although conversion from adjacency matrix to adjacency list is simple. Figure 4.12 shows how all procedures work together.

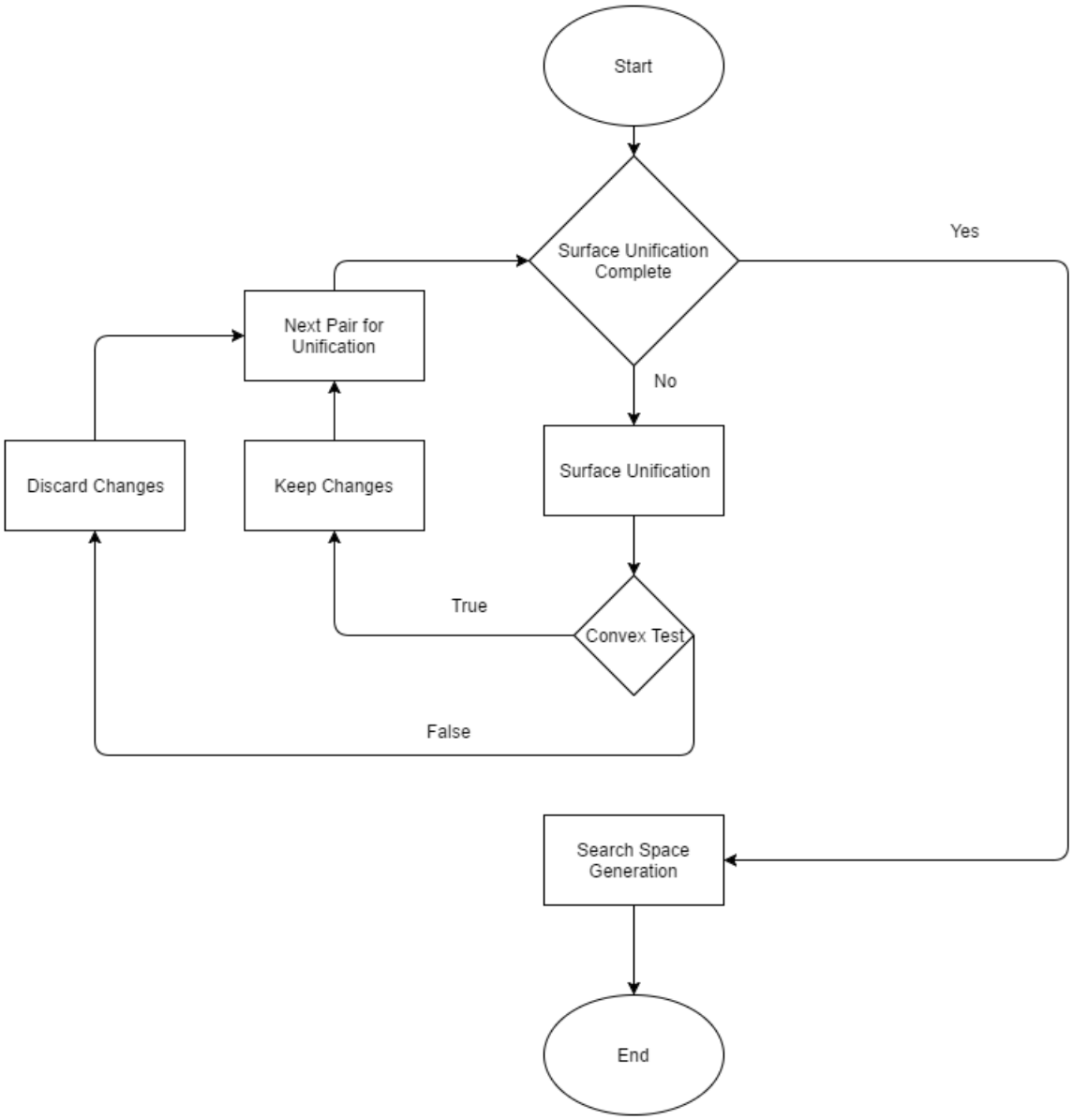


Figure 4.12 Flow chart of whole process

The total number of meshes available in a 3D scene is determined by the last number of the mesh index. The search space generation algorithm generates an adjacency matrix first and then converts it to an adjacency list. The adjacency matrix is a square matrix and

www.manaraa.com

shows connectivity between two elements. In this case, the elements are nodes and connectivity between them will become edges. The SearchSpaceGen algorithm is responsible for creating the adjacency matrix and the MatrixToList algorithm is responsible for creating the list from the matrix.

```
Algorithm SearchSpaceGen
     Input: double surfaceunify[][],surfacenormunify[][]
     Output: Adjacency Matrix searchspace[][]
     intermediate[]
i <- 0
searchspace[lastmeshindex][lastmeshindex]
for j = 0 to 480 step 1 do
 for k = 0 to 640 step 1 do
  if newdata(surfaceunify[j][k]) > 0
   for x = 0 to intermediate.length step 1 do
    for y = x+1 to intermediate.length step 1 do
    searhspace[intermediate[x]][intermediate[y]] <-
intermediate[y]
    searhspace[intermediate[y]][intermediate[x]] <-
intermediate[y]
    end for
   end for
  end if
 end for
end for
```

The search space generation algorithm uses the surfaceunify matrix which is the result of the surface unification process. The function newdata searches in the surfaceunify matrix and store data on each cell to the intermediate matrix. The intermediate is a single dimensional array which holds information about cell connectivity. The surfaceunify holds the same data as surfacerecon.

The worst case time complexity of search space generation is O (jk) where j represents the height of the surface unification array, k is the width of the surface unification array. The j and k are part of the first nested loop which parse 2-dimentional array where $j \neq$ k.

www.manaraa.com

Each cell in surfaceunify holds the list of connected polygons, sometimes a single element, which represents a point inside the polygon, and sometimes multiple elements, which represents a point on an edge. The points on the edge have a connection with other polygons, which is represented by unordered pairs of listed elements. The adjacency matrix is normally represented as true or 1 as a connection and false or 0 as no connection, but here we have put the column number instead of 1 or true. The concept of an adjacency matrix is discussed more in section 3.1.

```
Algorithm MatrixToList
     Input: double searchspace[][]
     Output: Adjacency List adjlist
List adjlist[lastmeshindex]
i <- 0
for j = 0 to searchspace.length step 1 do
 for k = 0 to searchspace.length step 1 do
  adjlist.add(j)
  if searchspace[j][k] ≠ 0 && i >= lastmeshindex
   adjlist[i].add(searchspace[j][k])
  end if
 end for
i++
end for
```

The *MatrixToList* procedure converts the adjacency matrix into an adjacency list. We have used an array of lists to implement the adjacency list. Each cell in the array holds the first list member and *add* method adds new elements to the list. The following diagram shows the whole processing pipeline for search space generation.

The worst case time complexity of search space generation is $O(n^2)$ where n represents the dimension of search space array which is square array of n elements. It is required to parse whole array hence we have implemented double nested loop.

www.manaraa.com

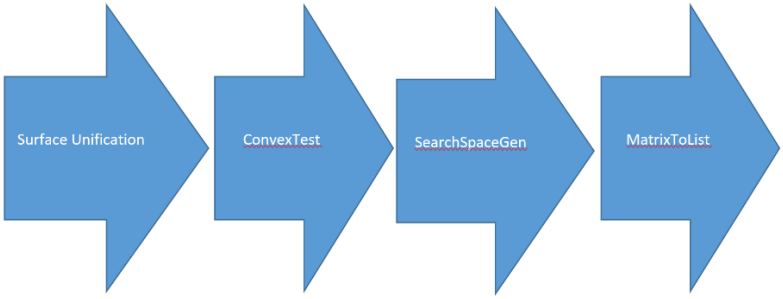


Figure 4.13 Pipeline of entire process

The adjacency matrix contains the number of columns if there is an edge between two meshes, else 0.

4.4 Example of Whole Pipeline

This section shows example of start to end pipeline processing. As shown in figure 4.14, scene of flat wall was perceived using Microsoft Kinect.

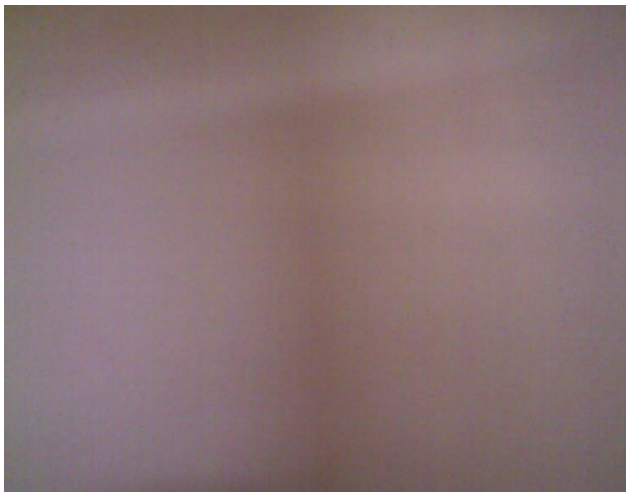


Figure 4.14 Flat wall scene as an input

The figure 4.15 shows depth map of figure 4.14 scene. The point cloud is created using Kinect generated depth map. The generated point cloud is passed to KinectFusion

www.manaraa.com

algorithm and output is shown in figure 4.16. The 3D reconstructed surface is made with triangular meshes and shown in figure 4.17. It is quite big mesh so we have considered blue outlined portion for this example. The output of Kinect fusion is forwarded to surface unification process.

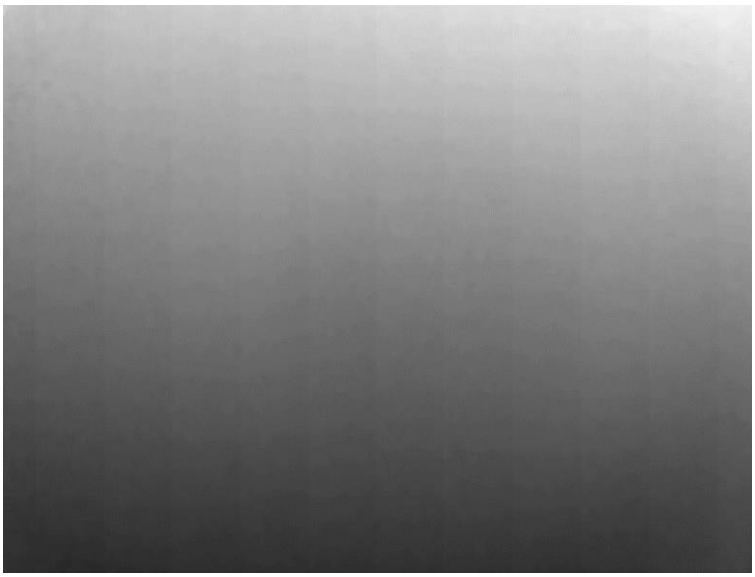


Figure 4.15 Depth map of figure 4.15

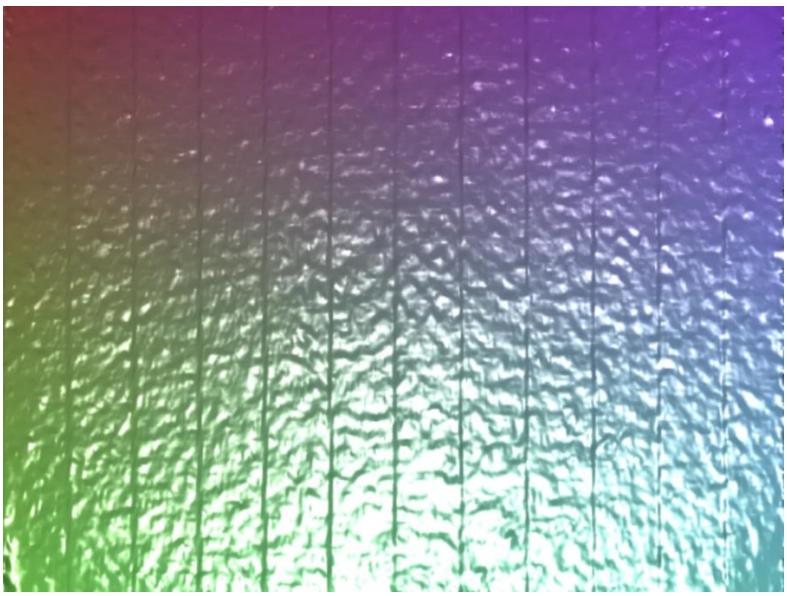


Figure 4.16 KinectFusion output of figure 4.15

www.manaraa.com

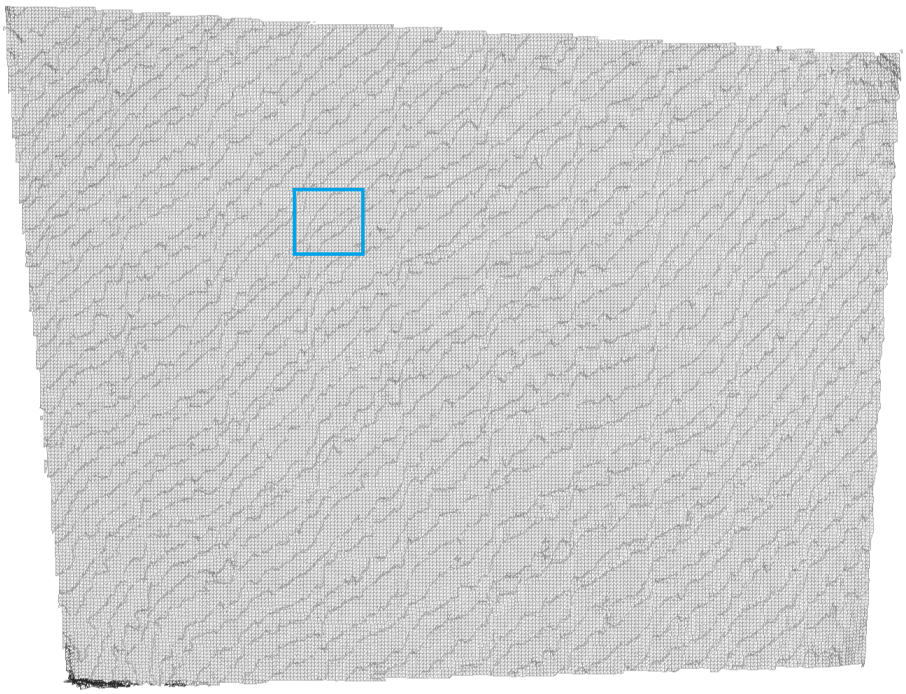


Figure 4.17 3D reconstruction mesh

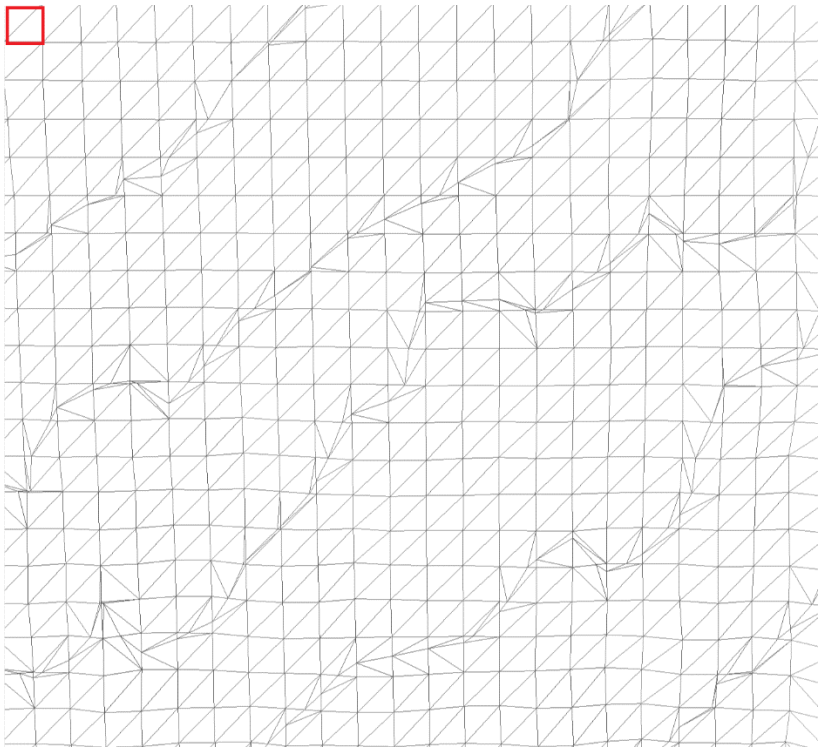


Figure 4.18 First cycle of surface unification with convex test

www.manaraa.com

Figure 4.18 shows the first two polygons considered for unification. Once they are unified, convex test will check convexity of generated polygon. In this case, it will return true and process goes on. As shown in figure 4.19, The first iteration of surface unification and convex test will generate red block as shown in the diagram. After processing whole frame the output will pass to search space generation procedure which will generate adjcency matrix and then adjacency list.

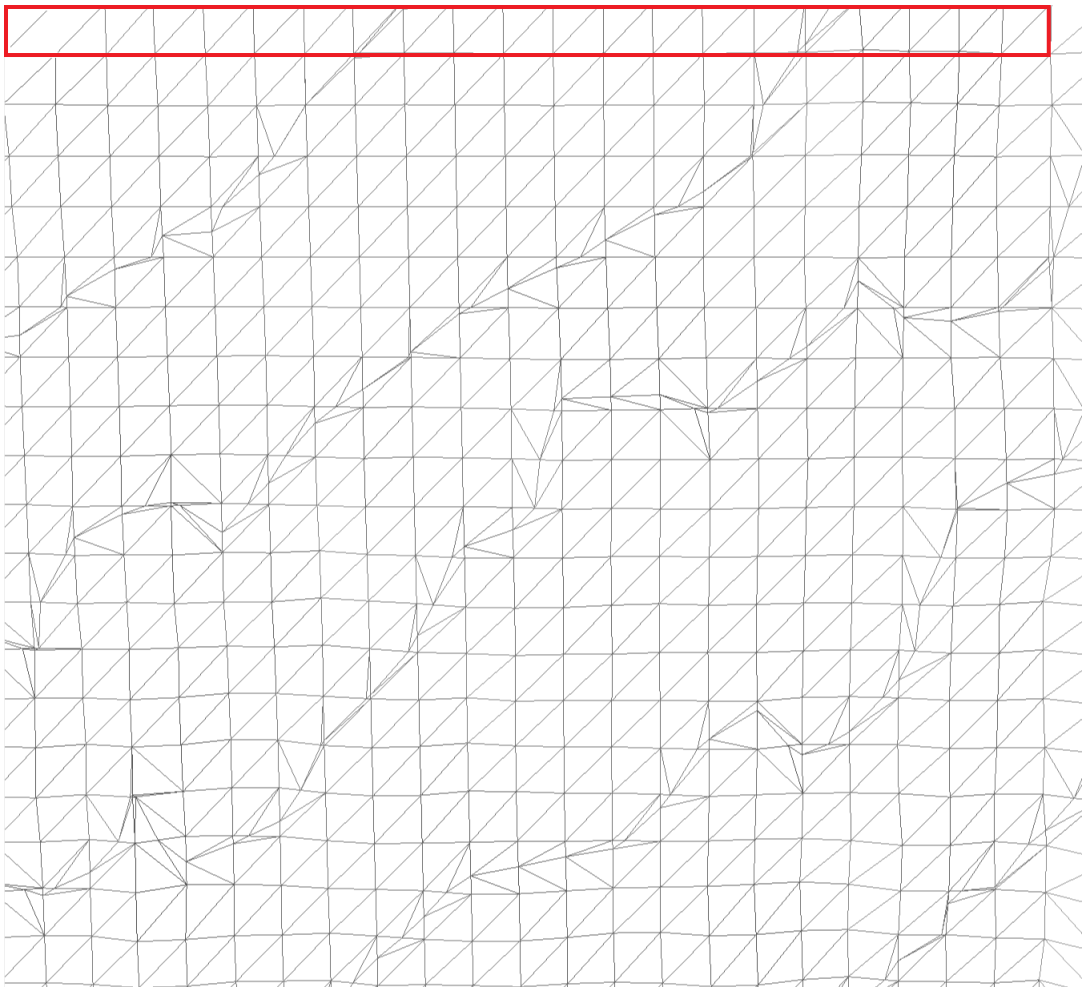


Figure 4.19 First iteration of surface unification with convex test

www.manaraa.com

CHAPTER 5

Experiments and Results

5.1 Microsoft Kinect

Microsoft Kinect is a depth sensor which can get the depth data along with visual data. Normal cameras which are often called RGB sensors can only perceive visual information in the form of pixels. Each pixel is made of different values of Red, Green and Blue respectively. The picture taken by an RGB sensor is a 2D representation of a 3D world. The world coordinate is converted to camera coordinate and camera coordinates are converted to frame coordinates using various translation, rotation and projection techniques. One of the major problems in visual processing is to determine the depth of a particular pixel using camera intrinsic and extrinsic parameters. Pinhole model can help to get the depth of a particular frame pixel but it requires minimum two points, intrinsic and extrinsic parameters.

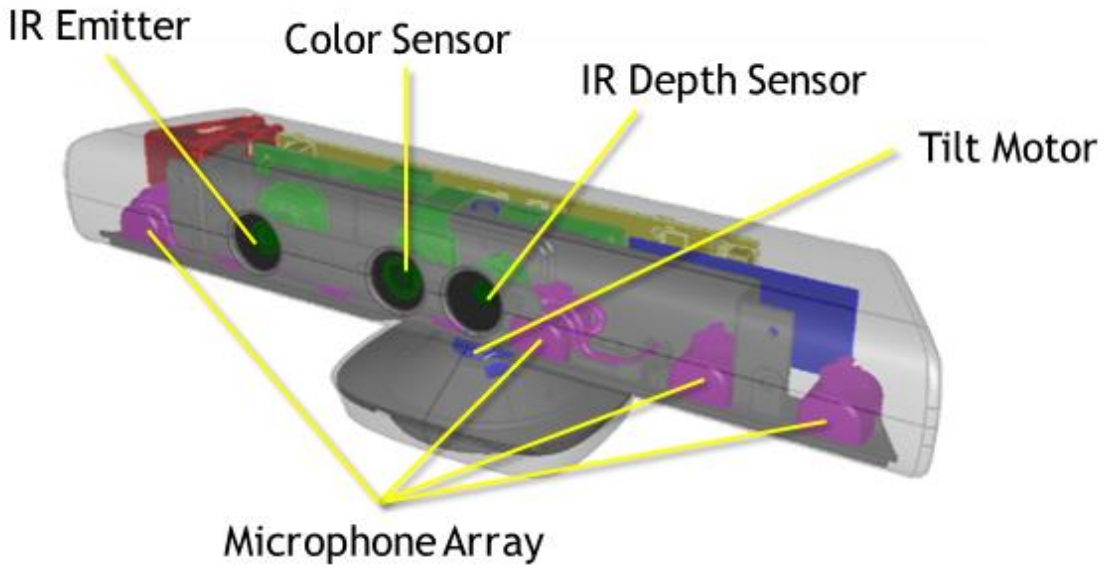


Figure 5.1 Microsoft Kinect as a depth sensor [12]

www.manaraa.com

The frame pixel is made of two coordinates x and y, but a point in the scene is made of three coordinates x, y and z. The RGB-D sensors on other hand, provides direct depth data using IR light emitters. So, there is no processing required to get depth of any pixel of the frame because Microsoft Kinect gives depth in millimeters.

The IR emitter in Kinect sensor transmits previously known pattern of IR dots and IR receiver receives that dot pattern. The difference in position of IR dots are considered and depth data is calculated.

Figure 5.2 Kinect depth data

As shown in the Figure 5.2, depth data is represented in pixel value of perceived image. The diagram shows 640 * 480 pixel frame and each pixel has its depth in millimeters. It

www.manaraa.com

is quite easy to build the point cloud using depth data. The x and y coordinates can be determined using pinhole model and z coordinate using perceived depth data. Microsoft Kinect has many versions but we have used Microsoft Kinect XBOX 360 in this thesis. Each version has minor changes in its capabilities [20].

5.2 System Specification

The minimum requirements for the experiment depends upon the selection of Microsoft Kinect Device. The hardware and software requirements are listed below.

Operating System	Windows 7
Software Development Kit	Kinect for Windows SDK

Table 5.1 Software requirement for Kinect

Processor	32-bit(x86) or 64-bit(x64) processor Dual Core 2.66 GHz or faster
Connectivity	Dedicated USB 2.0
RAM	2 GB RAM

Table 5.2 Hardware requirement for Kinect

Microsoft has released Kinect for Windows SDK for easy access to all Kinect-related functions and libraries. Although, Kinect can be programmed with other third party SDKs like OpenNI and OpenKinect, this thesis has used Kinect for Windows SDK because its implementation is in C#.NET language and we have also implemented experiments in C#.NET.

www.manaraa.com

The field of view for Kinect is 43 degrees vertical and 57 degrees horizontal. The range of depth sensing is 80 cm minimum to 4-meter maximum. The newer versions have near-view option which can sense the nearest 40 cm.

5.3 Experiment Design

The experiments are designed in such a way that four main concepts, surface unification, convex test, search space matrix generation and matrix to list conversion are covered. The experiments are performed first on example dataset. The next step of experiments is divided into two subcategories: With obstacle and without obstacle. The implementation of the experiments needs good lighting conditions as we are working with Microsoft Kinect and KinectFusion algorithm. The Microsoft Kinect is already calibrated so we have skipped the calibration procedure. The experiment setup is different for each case and it will not affect end results, as all point cloud and surface reconstruction calculations are relative to the chosen coordinate system. In this experiment, a table with the dimension of 3.50 ft * 2.25 ft * 2 ft (length * width * height) and a box with the dimension of 1 ft * 1 ft * 0.50 ft (length * width * height) have been used. The table acts as a flat surface and a box acts as an obstacle.

The result of search space generation is dependent on the 3D reconstruction algorithm because our proposed method uses the output of the 3D reconstruction algorithm as an input for the processing pipeline. As we have not considered the case with registration, sometimes the 3D reconstruction algorithm produces a reconstruction with holes meaning that the scene is discontinued from the search space perspective. We cannot consider a hole as an obstacle because even obstacles have search space. We have treated obstacles as a region of movement. This is the main reason we have removed a portion from the

www.manaraa.com

original scene and found the best reconstruction without holes. As the experiments were performed under a controlled environment, it is easy to find blocks of 100 * 100 pixel in reconstruction.

5.3.1 Example Dataset

The example dataset is specifically made for this experiment. The special reconstruction of 9 points (3 * 3 reconstruction matrix) and corresponding normal collection is used. These points are not recorded using Microsoft Kinect but used same as other recorded points. As show in below Figure 5.3, 9 points of point cloud are reconstructed using triangular mesh.

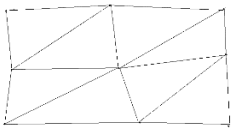


Figure 5.3 Reconstruction mesh of example data

The Table 5.3 is the reconstruction matrix of example dataset.

1	1,2,3	3,4
1,2,5	2,3,4,5,6,7	4,7,8
5,6	6,7,8	8

Table 5.3 Reconstruction matrix for example dataset

www.manaraa.com

5.3.2 Scene without Obstacle

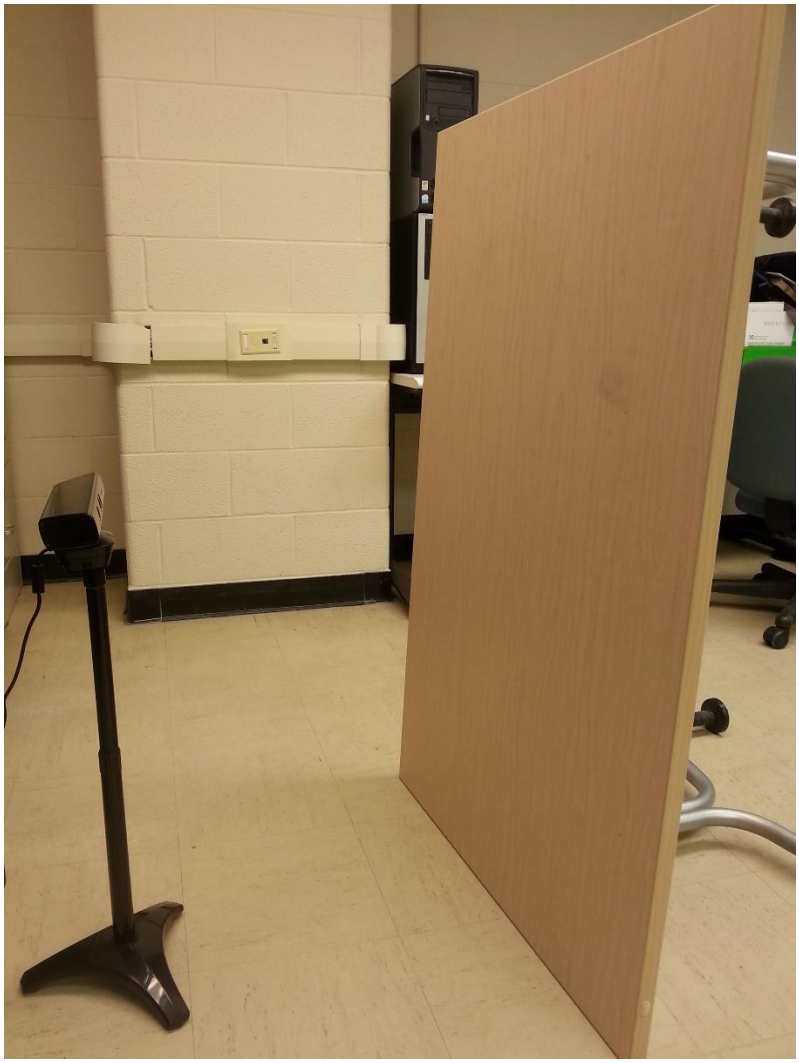


Figure 5.4 Experiment Setup for obstacle-free scene

The obstacle-free scene is arranged in such a way that the table acts as a wall and covers the whole frame. We have taken a portion of 100 * 100 pixel from the input from 640 * 480. The reason for sampling has been explained in previous topic 3.2. As shown in the Figure 5.5, the 3D reconstruction with mesh shown acts as an input for our pipeline. We have only used the first 100 pixels to the generate search space.

www.manaraa.com

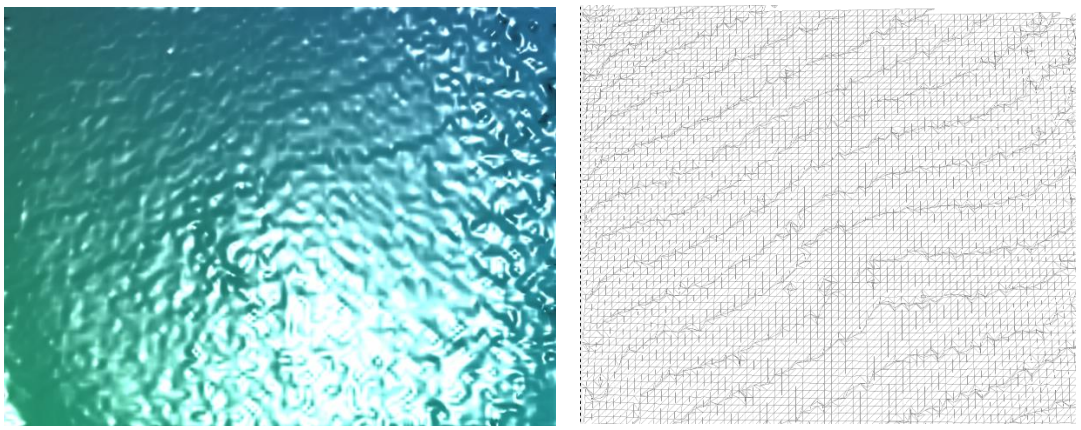


Figure 5.5 3D reconstruction Left: textured Right: meshed

5.3.3 Scene with Obstacle

Figure 5.6 Experiment Setup for obstacle scene

The obstacle scene is arranged in such a way that the depth-conceivable side of the obstacle always remains facing the Kinect (i.e. the side with the greatest surface area). As

www.manaraa.com

such, the orientation of the obstacle does matter during the experiment. We haven't used top view as an input although top view produces reconstruction without any holes. The reason for this is because we want to demonstrate the capability of search space generation procedures which can generate search space on obstacles too. Hence, we have set up Kinect at different elevations.

The height of the object also affects the output. Thus, we haven't used thin objects as an obstacle. The proposed method does not use any vision-based obstacle detection algorithms, hence, it cannot detect obstacles with very low height. We have used reconstruction parameters like angle between two polygons to differentiate between obstacle and non-obstacle space.

This reconstruction parameter-based technique has heavy dependency on quality of reconstruction. If the reconstruction has lots of error or noise, the generated search space will also be affected by it. For example, if reconstruction is not smooth or a flat surface has bumps on it, search space generation treats them as obstacles and produces different search space for it.

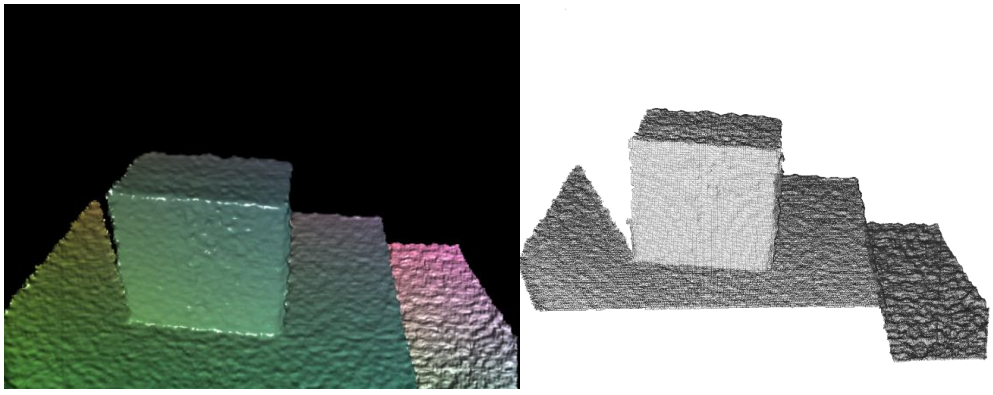


Figure 5.7 3D reconstruction Left: textured Right: meshed

www.manaraa.com

5.4 Results

The result section displays and discuss results gathered by all experiments. This section divided into three section – example dataset, obstacle-less scene and obstacle-based scene. The results for obstacle-less scene is recorded from single viewpoint as different viewpoint does not affect results. The obstacle-based scene is experimented from two different viewpoints. All experiments are performed with two different values of surface unification index. The result section also contains time to execute two major procedures – surface unification with convex test and search space generation with matrix to list.

5.4.1 Example Dataset

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	30	1	8	8	1
5	30	1	8	7	1
10	30	1	8	6	1
15	30	1	8	6	1
20	30	1	8	4	1
25	30	1	8	3	1

Table 5.4 An example dataset results

www.manaraa.com

The example dataset is created in such a way that normal collection has random normal distribution. The normal vector of any polygon is assigned as it has made angle less than 30 degrees with its neighbor polygon's normal.

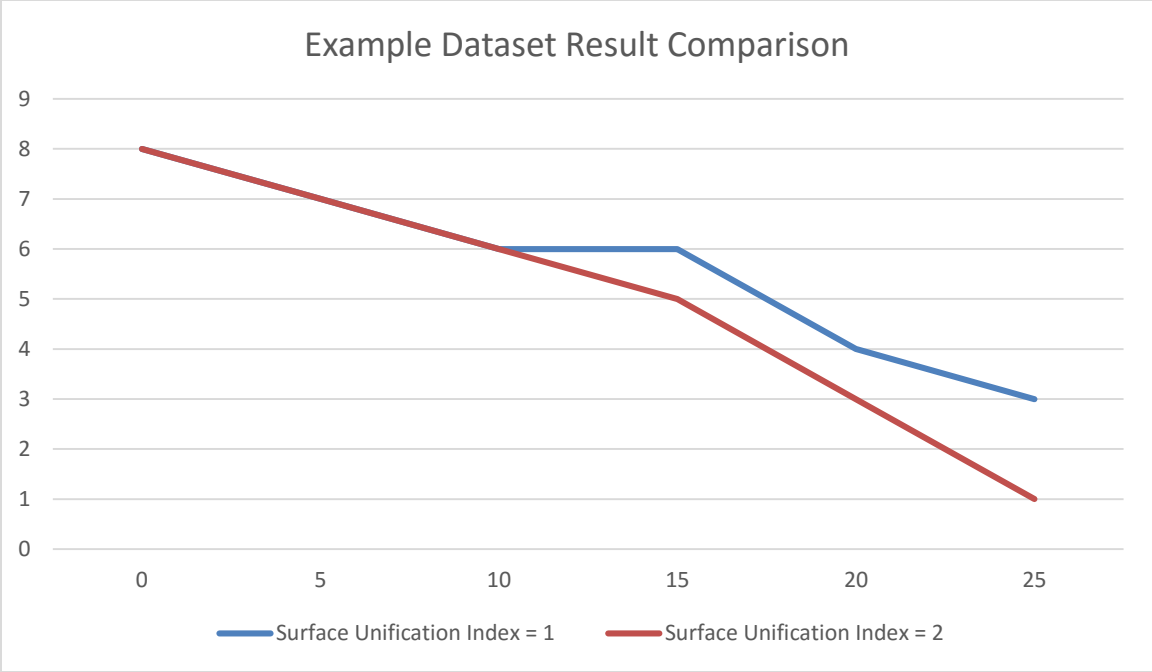
As shown in Table 5.4, all values of $\emptyset_{test}$ and $\emptyset_{obstacle}$ are in degrees. The example dataset does not have any normal whose angle of surface unification is more than 30 hence, there is only one generated search space.

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	30	2	8	8	1
5	30	2	8	7	1
10	30	2	8	6	1
15	30	2	8	5	1
20	30	2	8	3	1
25	30	2	8	1	1

Table 5.5 An example dataset results with different index

The above Table 5.5 shows results with search space unification index is equals to 2. As you can see, on second run the 25-degree case is converted into single polygon. As number of $\emptyset_{test}$ increases the change with higher index is also increases.

www.manaraa.com



Above graph shows comparison between two different surface unification index for example dataset. Table 5.6 shows execution time for example dataset experiment.

$\emptyset_{test}$	Surface Unification + Convex Test	Search Space Generation
0	0.005	0.003
5	0.004	0.003
10	0.004	0.002
15	0.004	0.002
20	0.004	0.002
25	0.004	0.002

Table 5.6 Execution time for example dataset

www.manaraa.com

5.4.2 Scene without Obstacle

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	30	1	10,106	890	1
5	30	1	10,106	768	1
10	30	1	10,106	417	1
15	30	1	10,106	158	1
20	30	1	10,106	124	1
25	30	1	10,106	107	1

Table 5.7 Scene without obstacle

We have considered 100 * 100 block in reconstruction matrix. Assume the reconstruction algorithm gives accurate reconstruction without any noise. In that case, the number of polygons available in a 100 * 100 pixel block will be almost 10,000 or more than that. The table 5.7 shows results for a scene without obstacle with various selections of angle of surface unification.

As you can see, the number of angle of surface unification rises, the number of polygons unified is decreasing which is considered a good sign from pathfinding aspect. It also

www.manaraa.com

means that the ability to detect small distortion in reconstruction or ability to detect small objects is also low with a high value of $\emptyset_{test}$.

The convex test rejection is simply showing the number of rejection for failure to be convex after unification, which is directly proportional to the number of polygons generated after unification. The surface unification index is 1 throughout the whole experiment. The higher number may generate better search space by reducing the number of polygons but it will also add a tremendous amount of time complexity (almost times the index of unification!)

The time to run this experiment depends upon the angle of reconstruction, reconstruction quality and 3D scene arrangement.

$\emptyset_{test}$	Surface Unification + Convex Test	Search Space Generation
0	0.23	0.19
5	0.25	0.17
10	0.19	0.21
15	0.20	0.22
20	0.21	0.21
25	0.22	0.20

Table 5.8 Execution time for each segment of execution pipeline

www.manaraa.com

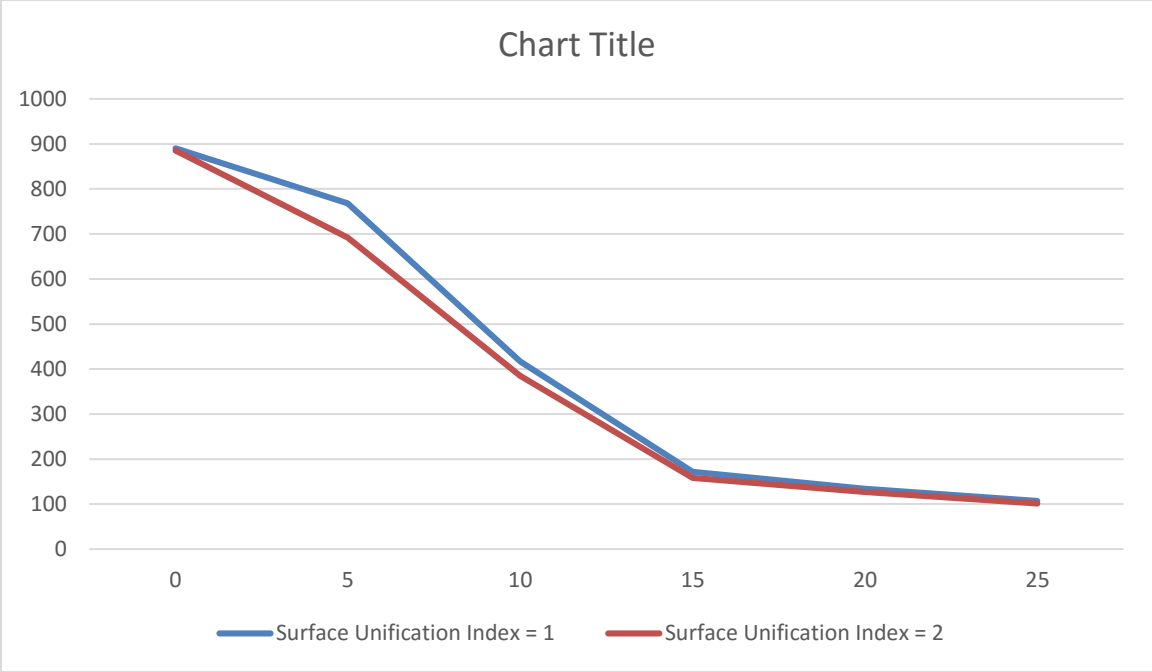
As shown in Table 5.8, the execution time for each process is different as they are processing different sizes of matrices. The highest was taken by surface unification and convex test. We have calculated the combined time for surface unification and convex test because convex test is called after every unification.

The $\emptyset_{obstacle}$ for this scene is 30 degrees which means if the angle of surface unification is 30 or more, it will detect it as an obstacle and separate the search space. This experiment does not have any angle of surface unification more than 30 so it has only one search space. The different values for surface unification index can affect execution time. The table 5.5 shows results for surface unification index equal to 2.

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	30	2	10,106	885	1
5	30	2	10,106	692	1
10	30	2	10,106	385	1
15	30	2	10,106	158	1
20	30	2	10,106	127	1
25	30	2	10,106	101	1

Table 5.9 Scene without obstacle for different surface unification index

www.manaraa.com



It is important to notice that the number of polygons after the second iteration of unification has decreased. As the angle of surface unification increases, the rate of decrement in the number of polygons after unification also increases, but since it increases the time for execution, it is not advised to increase more than 2 or 3.

5.4.3 Scene with Obstacle

The scene with obstacle is 640 * 480 pixels and has two different planes, one that the obstacle resides on and another that the table resides on. As shown in table 5.10, the number of disjoint meshes shows how many different search spaces (separated meshes) are generated from the scene. As we have discussed earlier, the search space generation pipeline generates search space on obstacles as well because the detection of obstacles is based on the angle of surface unification. As we have increased the angle of surface unification, the number of polygons generated is decreasing, which also means the ability to ignore the obstacle or distortion increases.

www.manaraa.com

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	50	1	307840	62375	21
5	50	1	307840	52257	21
10	50	1	307840	46348	21
15	50	1	307840	42522	21
20	50	1	307840	30254	21
25	50	1	307840	25698	21
35	50	1	307840	18666	21
40	50	1	307840	12650	21

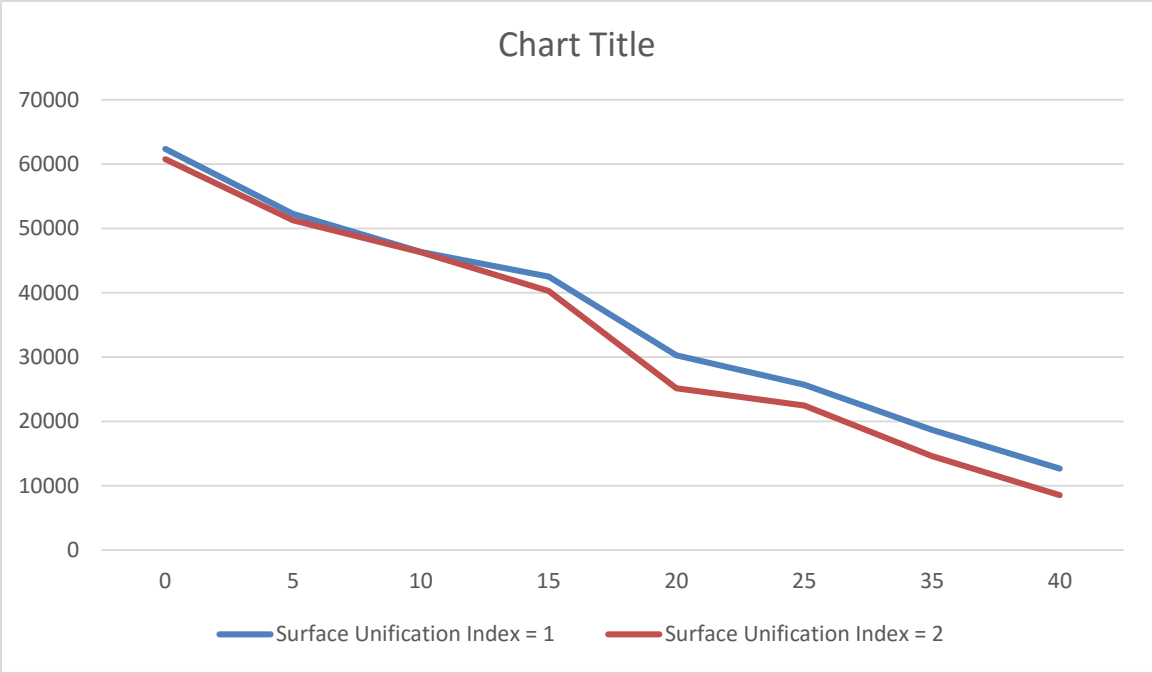
Table 5.10 Scene with obstacle from first point of view

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	50	2	307840	60751	21
5	50	2	307840	51244	21

www.manaraa.com

10	50	2	307840	46300	21
15	50	2	307840	40252	21
20	50	2	307840	25124	21
25	50	2	307840	22458	21
35	50	2	307840	14588	21
40	50	2	307840	8544	21

Table 5.11 Scene with obstacle from first point of view with different index



The above graph shows relation between different surface unification index results. As you can see from graph, as angle of surface unification increases, the number of unified polygons are decreasing. The number of unified polygons with index equals to 2 is less compared to polygons with index equals to 1

www.manaraa.com

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	50	1	307764	65642	19
5	50	1	307764	57758	19
10	50	1	307764	51477	19
15	50	1	307764	42106	19
20	50	1	307764	36758	19
25	50	1	307764	28445	19
35	50	1	307764	19984	19
40	50	1	307764	14446	19

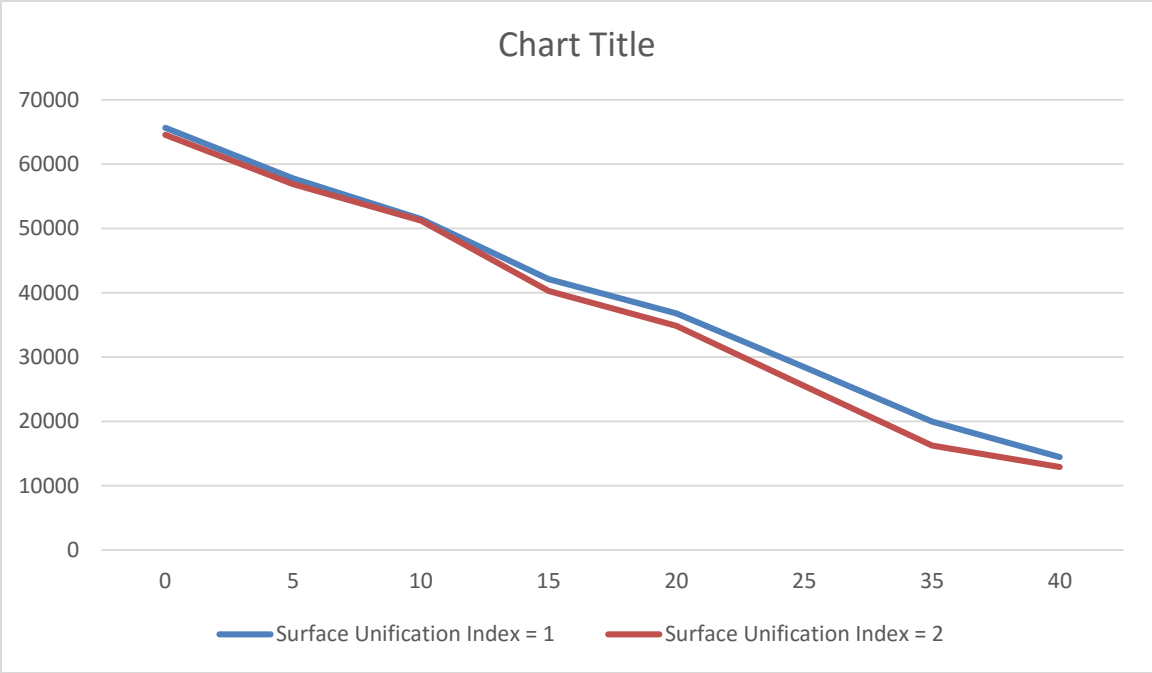
Table 5.12 Scene with obstacle from second point of view

$\emptyset_{test}$	$\emptyset_{obstacle}$	Surface Unification Index	Number of Polygons before Unification	Number of Polygons after Unification	Number of Disjoint Meshes
0	50	2	307764	64524	19
5	50	2	307764	56878	19

www.manaraa.com

10	50	2	307764	51210	19
15	50	2	307764	40254	19
20	50	2	307764	34856	19
25	50	2	307764	25476	19
35	50	2	307764	16214	19
40	50	2	307764	12878	19

Table 5.13 Scene with obstacle from second point of view



The table 5.12 shows the results from the second point of view which means the location of Kinect is shifted to a different point. There are no changes in scene arrangements or the surrounding environment. As shown in above graph, there is not much difference between first view point and second view point.

www.manaraa.com

$\emptyset_{test}$	Surface Unification Index = 1		Surface Unification Index = 2	
	Surface Unification + Convex Test	Search Space Generation	Surface Unification + Convex Test	Search Space Generation
0	0.59	0.48	1.08	0.98
5	0.55	0.49	1.05	0.97
10	0.52	0.42	1.25	0.98
15	0.58	0.43	1.16	0.95
20	0.59	0.46	1.19	0.91
25	0.56	0.44	1.02	0.92
35	0.47	0.49	1.04	1.01
40	0.52	0.43	1.08	1.02

Table 5.14 Execution time for each cycle for first view point

$\emptyset_{test}$	Surface Unification Index = 1		Surface Unification Index = 2	
	Surface Unification + Convex Test	Search Space Generation	Surface Unification + Convex Test	Search Space Generation
0	0.49	0.58	1.00	1.01

www.manaraa.com

5	0.52	0.57	1.04	1.01
10	0.50	0.55	1.02	1.02
15	0.48	0.54	0.98	1.00
20	0.51	0.52	1.03	1.02
25	0.51	0.52	1.02	1.01
35	0.51	0.48	1.00	0.99
40	0.52	0.51	1.00	0.99

Table 5.15 Execution time for each cycle for second view point

The execution time for scene with obstacle is obviously higher than that of scene without obstacle. There are several holes in scene without obstacles but the arrangements of holes do not affect surface unification. Hence, we have considered the whole scene for experiment.

The $\emptyset_{test}$ and $\emptyset_{obstacle}$ are two main type of angle of surface unification used in these experiments. The $\emptyset_{test}$ is determining factor for unification and $\emptyset_{obstacle}$ is responsible to detect obstacle. We have tested different values for $\emptyset_{test}$ and $\emptyset_{obstacle}$ to determine ideal values for the experiments. The values used in experiments are covering most of the possible configuration.

5.5 Comparison

The whole pipeline is unique in its implementation but the concepts used in each stage have correlation with other parallel research work. So, we can compare the other research

www.manaraa.com

work with appropriate section of pipeline from this thesis. The one of the important such research wok had been done by Oliva and Pelechano. They have introduced an algorithm called ANavMG which is automatic navmesh generation algorithm.

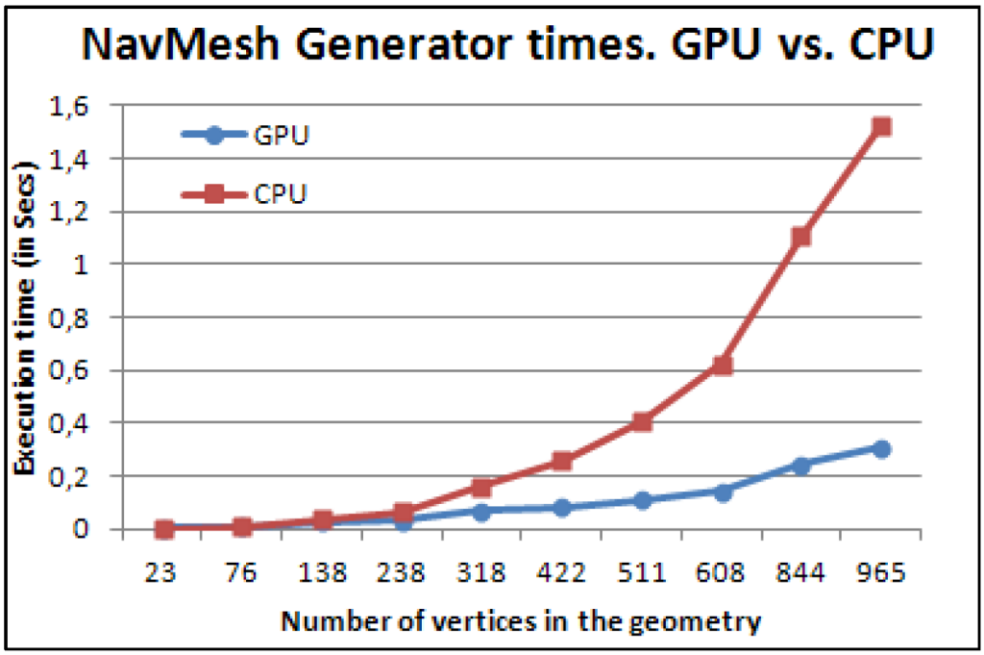
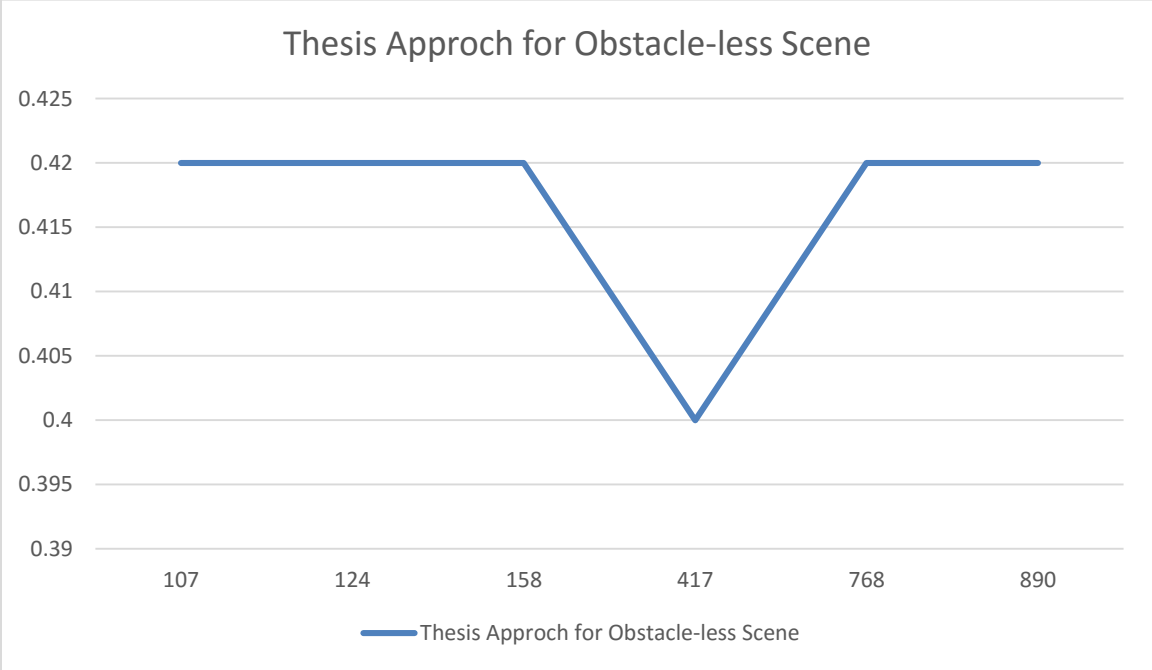
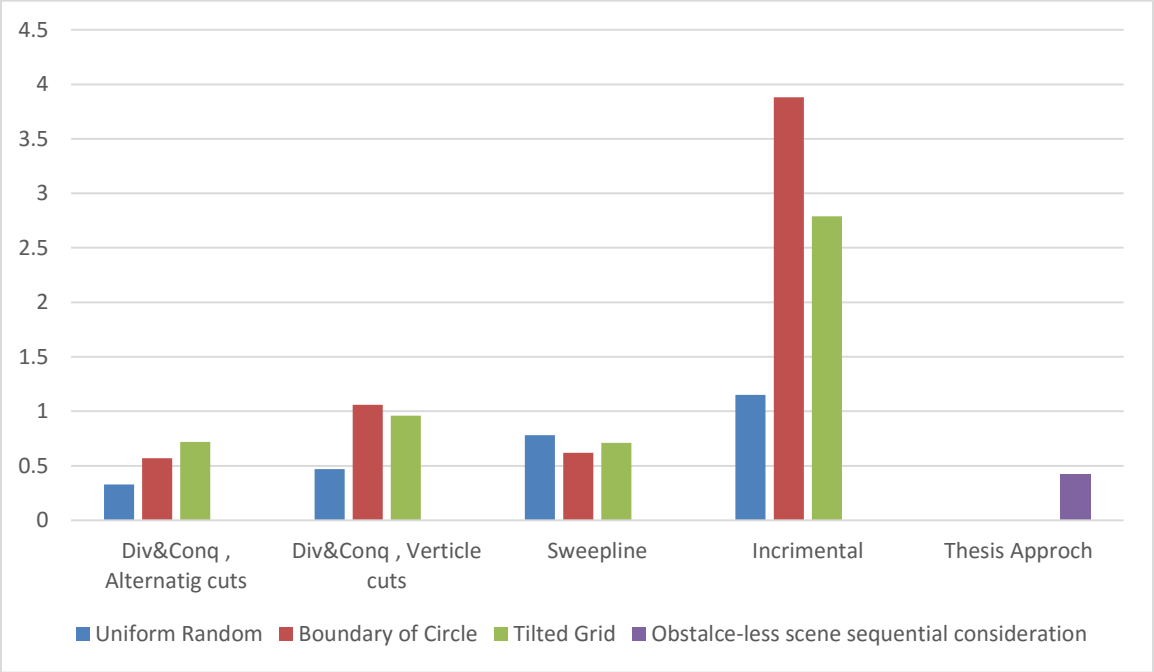


Figure 5.8 ANavMG with CPU and GPU performance



www.manaraa.com

As we can see from Figure 5.8 and following graph, the ANavMG approach has incremental time with the amount of node it has processed but thesis approach is pretty much static with the total number of vertices. In this obstacle-less scene we have used static 100 * 100 image frame. Hence, the total number of polygons are static where as ANavMG's input size is variable. The graph represent x-axis for polygons and y-axis for execution time. It looks like 417 polygons has huge drop but it is just slight change from 0.42 to 0.40 seconds.



There is one more paper which has compared different 3D scene triangle mesh generator algorithms in term of execution time [21]. Above graph shows execution time on y-axis in seconds. As we know, lower the time, better the algorithm is. Thesis approach has slight more execution time than divide and conquer method of uniform random triangulation.

www.manaraa.com

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

The thesis has explained and examined the concept of search space generation for 3-dimensional reconstructed surfaces. It can only work with the near field 3-dimensional scene because of the limitations of Microsoft Kinect sensor as we have used it as an RGB-D sensor. We have reviewed and were inspired by several popular research works such as PTAM [18] and KinectFusion [5].

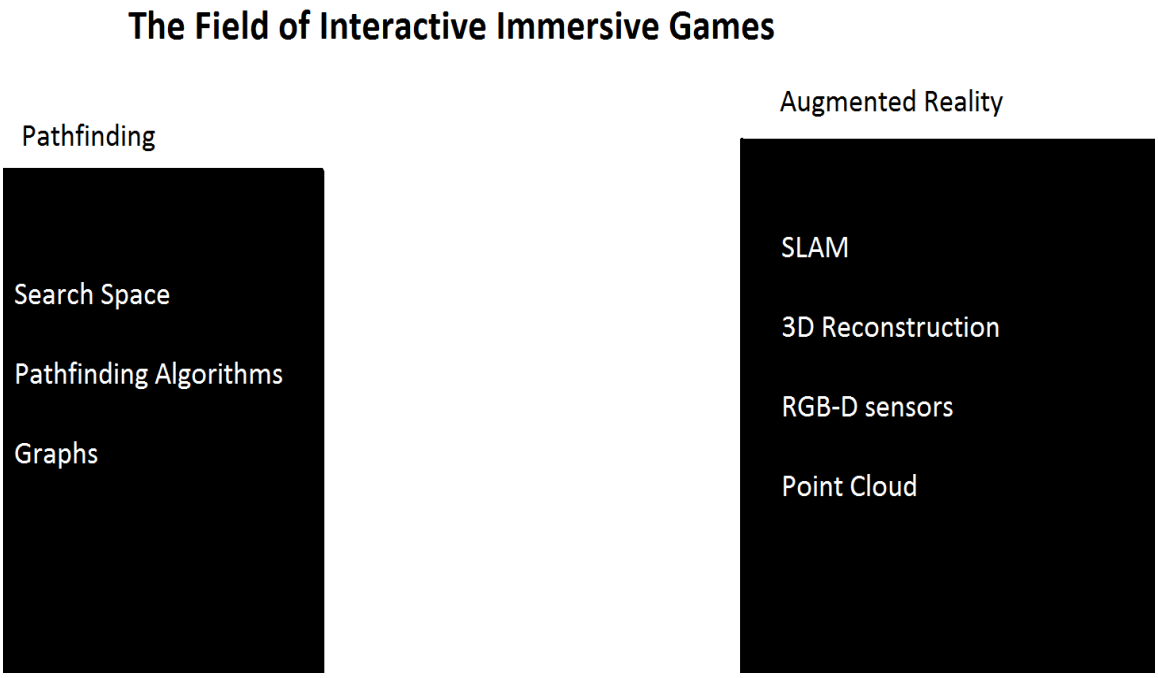


Figure 6.1 Gap between pathfinding and augmented reality

As shown in Figure 6.1, pathfinding used in game artificial intelligence and augmented reality is used in various aspects of vision-based technologies. The field of interactive and immersive games is huge and covers both pathfinding and augmented reality concepts. There is however, a gap between pathfinding and augmented reality regarding developing

www.manaraa.com

games. There is no existing framework for game developers such that they can apply knowledge of game AI to their projects, especially pathfinding. This thesis works as a bridge between these two fields and allows developers and designers to stretch existing knowledge of game AI to augmented reality. Through the concepts illustrated in this thesis, we can develop frameworks which can provide search space of 3D scenes in real time. Although we started this thesis by focusing on augmented reality as an application of the research, it may be used anywhere else depending on the requirements.

We have introduced the concept of surface unification, which is a new way to combine 3-dimensional reconstructed meshes. The next step in the process is a convex test which is a famous mathematical concept for checking convexity of a given polygon. The convex test we have implemented is configured for the whole processing pipeline. The next concept introduced was search space generation. It uses surface unification and convex test process to generate an adjacency list. The whole pipeline provides a base for search space generation framework.

The thesis also includes a handful of examples and experiments. The thesis concepts are tested over several cases and demonstrate satisfying results. The experiment setup is predefined, and a controlled environment is used as it is a similar practice in research work. We have set up a small 3D scene with one table and a few obstacles. The table represents a flat surface where the navmesh will be generated, and obstacles act as a separation point. The presented approach considers obstacles as a point where two search spaces are disjointed—one for regular flat surface and one for obstacle surface.

www.manaraa.com

6.1.1 Importance of the Thesis

Immersive gameplay experience has become so important in recent days which is beyond our imagination. After the dawn of augmented reality and RGB-D sensor like Microsoft Kinect, the new direction of game design has opened. This kind of depth sensor was previously only available to researchers and enthusiasts but is now available in the consumer market. For example, Google has introduced Project Tango [22] which has built-in depth sensor which can do lots of new things that traditional mobile devices cannot do. At the current rate of technology advancements, the day when humans will have depth perception-enabled mobile device is not far off. This type of depth sensor makes augmented reality a lot easier. The current games on the market with augmented reality features use concepts of marker-based tracking to implement augmented reality. Although they have designed games in such a way that users feel it as a markerless augmented reality, but actually it is not. This thesis provides a framework to create markerless augmented reality which can generate search space, and thus it will make the whole game experience more immersive and realistic.

6.2 Future Work

There are lots of components of this thesis where we can bring improved version. The basic important future work will involve improvement of time and space complexity of presented algorithms. The surface unification, convex test, and search space generation all use array as the main data structure. It can be made more effective by using other data structures.

www.manaraa.com

The convex test is a greedy algorithm, but we can improve the efficiency of the whole pipeline by implementing dynamic programming algorithm strategies, as well as potentially implementing machine learning to learn how many meshes should combine before running the convex test.

The major future work will be trying to implement registration-based surface unification. Registration may change the whole pipeline of processing, but it is a good improvement to make because it will make it more efficient and applicable. The implementation of registration-based search space generation will also make it process in real-time.

www.manaraa.com

BIBLIOGRAPHY

[1] "Grid Map." [Online]. Available: http://arongranberg.com/astar/docs/graph_types.php. [Accessed: 01-May-2016].

[2] "Navmesh." [Online]. Available: http://renaissancecoders.com/?p=306. [Accessed: 01-May-2016].

[3] "Microsft Kinect." [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect/develop. [Accessed: 01-May-2016].

[4] R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison, "Dense planar SLAM," *ISMAR 2014 - IEEE Int. Symp. Mix. Augment. Real. - Sci. Technol. 2014, Proc.*, pp. 367–368, 2014.

[5] R. A. Newcombe, D. Molyneaux, D. Kim, A. J. Davison, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," *IEEE Int. Symp. Mix. Augment. Real.*, pp. 127–136, 2011.

[6] A. M. Santana, K. R. T. Aires, R. M. S. Veras, and A. A. D. Medeiros, "An approach for 2D visual occupancy grid map using monocular vision," *Electron. Notes Theor. Comput. Sci.*, vol. 281, pp. 175–191, 2011.

[7] S. Golodetz, "Automatic Navigation Mesh Generation in Configuration Space," *Overload*, vol. 117, pp. 22–27, 2013.

[8] R. Oliva and N. Pelechano, "A GPU Based Method for the Automatic Generation of Near - Optimal Navigation Meshes," *Eurographics Assoc.*, 2012.

www.manaraa.com

[9] R. Oliva and N. Pelechano, "Automatic Generation of Suboptimal NavMeshes," *Motion in Games*, no. c, pp. 328–339, 2011.

[10] D. H. Hale and G. M. Youngblood, "Full 3D Spatial Decomposition for the Generation of Navigation Meshes," *Proc. Fifth Artif. Intell. Interact. Digit. Entertain. Conf.*, pp. 142–147, 2009.

[11] "Point Cloud Library." [Online]. Available: http://pointclouds.org/. [Accessed: 01-May-2016].

[12] "Microsoft Kinect Documentation." [Online]. Available: https://msdn.microsoft.com/en-us/. [Accessed: 01-May-2016].

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[14] D. W. F. van Krevelen and R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations," *Int. J. Virtual Real.*, vol. 9, no. 2, pp. 1–20, 2010.

[15] S. Rabin, *AI Game Programming Wisdom 2*. Jenifer Niles, 2003.

[16] R. Steven, *Game AI Pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2013.

[17] X. Cui and H. Shi, "An Overview of Pathfinding in Navigation Mesh," *IJCSNS*, vol. 12, no. 12, pp. 48–51, 2012.

[18] G. Klein and D. Murray, "Parallel tracking and mapping for small AR

www.manaraa.com

workspaces," *2007 6th IEEE ACM Int. Symp. Mix. Augment. Reality, ISMAR*, 2007.

[19] H. Delingette, "Simplex meshes: a general representation for 3D shape reconstruction," *Comput. Vis. Pattern Recognition, 1994. Proc. CVPR '94., 1994 IEEE Comput. Soc. Conf.*, no. June 1994, pp. 856–859, 1994.

[20] N. M. DiFilippo and M. K. Jouaneh, "Characterization of Different Microsoft Kinect Sensor Models," *IEEE Sens. J.*, vol. 15, no. 8, pp. 4554–4564, 2015.

[21] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," *Appl. Comput. Geom. Towar. Geom. Eng.*, vol. 1148, pp. 203–222, 1996.

[22] "Google Project Tango." [Online]. Available: https://www.google.com/atap/project-tango/. [Accessed: 01-May-2016].

www.manaraa.com

VITA AUCTORIS

NAME:	Ronak Patel
PLACE OF BIRTH:	Gavada, Gujarat, India
YEAR OF BIRTH:	1991
EDUCATION:	Gujarat Technological University, B.E., India, ’09-‘13
	University of Windsor, M.Sc., Canada, ’14-‘16

www.manaraa.com