DISASTER SCENE RECONSTRUCTION: MODELING, SIMULATING, AND PLANNING IN AN URBAN DISASTER ENVIRONMENT

By
Scott Herman
Honours Bachelor of Science
in the Program of Computer Science,
Lakehead University 2011

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Science
in the Program of
Computer Science

Toronto, Ontario, Canada 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

ABSTRACT

**DISASTER SCENE RECONSTRUCTION:
MODELING, SIMULATING, AND PLANNING IN AN URBAN DISASTER ENVIRONMENT**

Scott Herman

Master of Science, Computer Science, Ryerson University, 2014

Urban disasters are characterized by buildings collapsing. The rubble of collapsed buildings forms a chaotic, unplanned and unmapped environment in which emergency first responders must find the surviving occupants who are now trapped and hidden in the dangerous rubble. The more knowledge that search teams have pertaining to the resulting environment the better they are equipped to plan and rescue survivors. Our research demonstrates that simulations can be used to inspect urban disaster-related terrain remotely and safely within hours of the actual disaster.

The Disaster Scene Reconstruction system allows for the creation of accurate 3D models and a simulation providing custom functionality --such as virtual structural inspection and-- providing first responders the ability to plan actions in the simulated environment. The goal of this research is to demonstrate that the functionality we developed can be used to provide accurate information to users and potentially assist search and rescue planning efforts.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Alex Ferworn for his direction and support, both technically and financially. His passion for developing technologies that could save lives is what inspired me to pursue this thesis research topic.

I would like to thank my peers in the N-CART research lab, Jimmy Tran, Alex Ufkes and Christopher Kong. They have provided much support and guidance during my time at Ryerson.

Finally, I would like to thank my fiancé Erika Freund for being there every step of the way and a constant source of support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3D	Three-dimensional
API	Application Programming Interface
BoO	Base of Operations
BPA	Ball Pivoting Algorithm
CLI	Common Language Infrastructure
CRASAR	Center for Robot-Assisted Search and Rescue
CUDA	Compute Unified Device Architecture
DSR	Disaster Scene Reconstruction
EMS	Emergency Medical Services
FEMA	Federal Emergency Management Agency
FOV	Field of View
GPGPU	General-Purpose Computing on Graphics Processing Units
GUI	Graphical User Interface
HUSAR	Heavy Urban Search and Rescue
IR	Infrared
JS	JavaScript
LRF	Laser Range Finder
MC	Marching Cubes
N-CART	Network-Centric Applied Research Team
NIST	National Institute of Standards and Technology
ODE	Open Dynamics Engine
OPP	Ontario Provincial Police
PC	Personal Computer

PCL	Point Cloud Library
POI	Point of Interest
PPE	Personal Protective Equipment
RCMP	Royal Canadian Mounted Police
RGB-D	Red/Green/Blue – Depth
RRP	Reference Rubble Pile
SAR	Search and Rescue
SCSC	Summer Computer Simulation Conference
SS	Structural Specialist
TF	Task Force
TFT	Time-for-Trial
UAV	Unmanned Aerial Vehicle
UDK	Unreal Development Kit
NIST	National Institute of Standards and Technology
USAR	Urban Search And Rescue
USARSim	Unified System for Automation and Robot Simulation
VCG	Visualization and Computer Graphics Library

CHAPTER 1 INTRODUCTION

Imagine you are spending a Saturday afternoon at the shopping mall. You notice dust falling from the ceiling. Suddenly, a few pieces of ceiling tile break free and fall to the ground in front of a small group of people who laugh and comment about how things aren't made the way they used to be. Then the ground starts to shake, more debris falls, the people scatter as it becomes obvious the roof is collapsing. Everything goes dark. When you regain consciousness you cannot see anything but can hear the muffled screams and moaning of people around you. You realize you are pinned in place by something very heavy making it difficult to breathe. You begin to panic. This may be the end. This is a disaster.

It is human nature to believe this type of scenario will never happen but evidence suggests that these types of events happen all around the world. A similar event occurred in 2012 at Elliott Lake, Ontario [1] when the town`s only mall collapsed leading to two fatalities who died in circumstances not too dissimilar from our scenario.

In response to the risks associated with this type of disaster most nations have developed mechanisms to respond to large-scale urban disasters requiring specialized skills and equipment. Urban Search and Rescue (USAR) teams are deployed to locate, medically stabilize and rescue trapped "patients"[1]. However, finding victims within the rubble formed from collapsed buildings is often problematic because of the dangers of secondary collapse. Searching requires careful planning to allow access to the locations where victims may be hidden in the debris.

Search specialists and structural engineers are often unable to survey the affected area without putting themselves at risk due to the instability or inaccessibility of the rubble. Increased

[1] In this work we use the term "patient" in the USAR context—meaning any person trapped in rubble, in need of rescue.

understanding—or "situational awareness"—of the rubble environment allows these people to better decide how to proceed with searching for and extricating patients.

Recent events that have led to urban disasters are not hard to find. These include the 2010 Haiti earthquake [2], the 2011 Tōhoku earthquake and tsunami [3], the 2001 Twin-Tower collapses (see Figure 1.1) [4], and the Algo Centre Mall collapse [1].

Figure 1.1: The resulting rubble from the September 11, 2001 attacks on the Twin Towers of the World Trade Center, New York, USA [4].

In this thesis, we present work to demonstrate our intent to provide an accurate visualization tool for data collected at a disaster scene that can be used to enhance the situational awareness of first responders who are in transit (in bound) to the scene or are working at the scene (committed). An accurate 3D model of the scene will allow searchers to view the scene from a variety of perspectives and in safety. Potentially, this could help to reduce the time it takes to find and rescue trapped victims. In addition the tool may provide rescuers an additional

method of identifying entry points and spotting dangerous or unstable regions that require additional support (shoring).

Previous work has shown that using an Unmanned Aerial Vehicle (UAV) with an attached RGB-Depth sensor can create dense 3D point cloud models of rubble from a safe distance [5]. We collected our test data at a mock-disaster environment provided by the Ontario Provincial Police (OPP). The terrain is challenging to traverse on foot and is unstable. We scanned and recorded the area using a sensor-equipped UAV to create a 3D point cloud model. Flying over the terrain allowed a large area to be surveyed quickly and safely, requiring no human engagement on the ground. The resulting point cloud can be viewed within a 3D modeling program, such as Meshlab [6]. However, a user of this visualization can easily become disoriented as the point cloud itself obscures their ability to imagine the actual rubble view.

The use of game engines for serious simulation is becoming more common [7-9]. A modular design allows functionality to be applied to other domains by sharing assets. This allows the rapid creation of simulations with different capabilities and data. With the inclusion of physics engines, many simulations can be created quickly and, perhaps more importantly, accurately. Applying these physics engines to simulated urban rubble model data can lead to simulating interactions with an unsafe environment in a physically accurate way.

As these models can be produced quickly, it should be possible to create them from data gathered locally (at the disaster site) and transmitted to a distant USAR Task Force. The time it takes for an in-bound USAR Task Force to arrive at a disaster site could be, and often is, many

hours[2]—often wasted. With accurate models, this time could be used to concurrently plan for an actual operation.

Current point cloud viewers have limited toolsets to interact with and learn from data sets in a USAR context. We propose that it is possible to create 3D models of real-world terrain from collected point cloud data within a practical time-frame of hours to support search and rescue operations. Using a game engine and accurate backend physics engine we can simulate real world tasks, a capability only possibly with 3D mesh models. The ability to interact with the models will provide better situational awareness and support the efficient allocation of scarce resources at the scene of an urban disaster.

1.1 Problem Definition

The rubble created from an urban collapse is unmapped, unstable and dangerous. The potential search space could be vast and determining the most critical areas in which to search first is problematic but crucial to save the lives of trapped victims. Areas of interest can be dangerous to access and require time to reach safely. Situational awareness of the terrain allows USAR teams to make informed decisions about where to search for trapped survivors. Search dogs have the ability to find humans using air scenting, but face the same dangers as humans when traversing the terrain and cannot communicate what they are experiencing while traversing rubble. Ground robots can be used on rubble with a variety of sensors to remotely inspect areas and enter confined spaces. However, ground robots normally have very poor cross-rubble performance and are a poor alternative to dogs when actually looking for victims. Clearly, alternative inspection methods that can cover large areas quickly with little danger would be very useful when planning search operations.

[2] It took 14 hours by bus for CAN-TF3 to respond to Algo Centre Mall Collapse.

Locating trapped patients within the rubble often requires first responders to manually inspect the terrain and enter the collapsed structure. This can only occur when a structure has been declared safe for human entry. “Shores” are supporting structures that are put in place by first responders along a search path in order to stabilize areas within a damaged building that may be unstable and, therefore, dangerous. Before a shore is placed, the unstable terrain must be manually measured in order that an existing expedient temporary shore can be placed or a permanent shore locally fabricated. The measurement process is inherently dangerous as it exposes first responders to the unstable structure. In addition, the measurement delays construction of a shore until a human can actually get to the site to make it. The shore is then constructed and placed allowing further entry. As more areas require support, the process is repeated. All of this typically takes many hours. As trapped people in rubble are inevitably in peril, the delay to search operations may result in a consequent reduction in rates of survival for trapped victims.

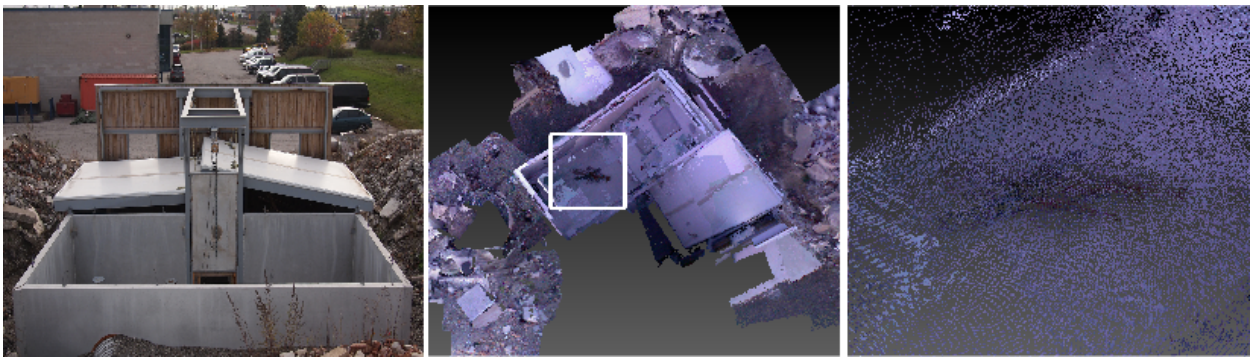


Figure 1.2: From left to right, a mock disaster environment, the created point cloud model highlighted with an area of interest, and a close inspection of the highlighted area.

As shown in Figure 1.2, previous work [5] creates point cloud models of real-world disaster environments. These models contain hundreds of thousands of points and must be viewed in specialized software. Humans are good at detecting patterns in relatively sparse visual

data and point clouds provide enough context so that most people can “fill in the blanks” to identify a scene. However, on closer inspection (zooming in) the points begin to appear further apart and the viewer loses the cues that a dense set of points provided. The visual acuity provided by our ability to detect a pattern is lost. In addition, functionality which might assist planning for an additional ground search is missing in simply viewing point clouds—most notably the ability to interact with components of the cloud in ways similar to the actual terrain. This is not to say that point clouds are not useful. They are a building block of a 3D model and can be further processed to become a useful asset.

We wish to extend the current viewing capabilities of point cloud data by creating a 3D model and importing it within a game engine. Leveraging a backend physics engine real-world tasks can be simulated, which is currently not possible in available point cloud viewers.

1.2 Objectives

The aim of this thesis is to demonstrate that it is possible to create accurate 3D models of post-disaster terrain within hours after arriving, inspect and process the models within a game engine simulation that provides an experience analogous to being at the disaster site, and allow functionality that would support actual operations at the site. The system is intended to be used at the disaster scene or en-route.

1.2.1 Contributions

To the best of our knowledge, we are the first to develop a system that allows for the rapid creation of real-world disaster models in simulation from collected data. Our system provides customized functionality that allows users such as structural engineers or first responders to virtually examine simulated disaster rubble in unique ways. After arriving at a disaster site we can provide our system to a search team within a practical time-frame. The simulation is created in less than five hours from the point of scanning, creation of a 3D model, and the use of the model within the game engine. An earlier revision of our system was published at the Summer Simulation conference [10].

We determined what techniques are effective at processing point cloud data by applying various algorithms to reduce the size of the data set and create accurate 3D models compatible for use within a common game engine (Unity). As a result we created the Model Creation Pipeline which is used with our data collection technique. This pipeline is modular and can have better performing methods swapped or added to increase overall performance.

We conducted a series of experiments to accurately predict the task of traversing rubble within a simulation and validated our results against actual rubble traversal. We claim that our "Time Trial Mode" can be used by USAR personnel to estimate traversal times across the varying terrain of disaster rubble within seconds under some conditions. Also, it allows inspection of the terrain to determine if it is fit for traversal.

We have developed an automated method within the Disaster Scene Reconstruction system to support the creation of virtual support systems (shores). We have verified that our virtual technique called "Shoring Mode" can be used to create virtual shores that are physically

accurate with the benefit of providing the list of materials and measurements needed to actually construct them in reality.

Finally, we provided a series of code modules and a framework that can be used by other researchers and developers to extend the functionality present within the DSR simulation. As environments are modeled and interacted with, the information contained within the "scene" can be reloaded and referred to. This will allow a method of archiving past disaster events which, to our knowledge, has never been done before.

1.3 Thesis Organization

This chapter serves as an introduction to four additional chapters. Chapter 2 presents an overview of emergency management generally and the strategies, techniques and technologies used in USAR operations specifically. A literature review of related work in point cloud data processing and surface reconstruction techniques is also presented. Lastly, the topic of serious games is introduced along with the game and physics engines that are commonly used in the creation of them.

Chapter 3 details the environments and techniques used to collect our data. The Model Creation Pipeline is presented and how it is used to create 3D models from point cloud data. Next, the Disaster Scene Reconstruction system architecture and functionality used to inspect and interact with the modeled terrain is detailed.

In Chapter 4, experimental results pertaining to the modeling of terrain are presented. Next, the DSR system is used to verify the accuracy of the implemented functionality. This is followed by an analysis and discussion.

Finally, Chapter 5 presents a summary of this thesis, the results that were obtained, and discusses some ideas for future research.

CHAPTER 2 BACKGROUND

This chapter begins with a general introduction to disaster and emergency management followed by a more specific discussion of Urban Search and Rescue and the Task Force system. Next, an introduction to rescue robots and the available sensors that can be employed are presented. Examples of simulations currently being employed for disaster-based research will be reviewed. The chapter also includes a discussion of "point cloud" data and how it can be used in developing 3D simulations. Finally, game and physics engines will be introduced and examined as research tools.

2.1 Disasters and Emergency Management

A **disaster** is "a sudden calamitous event bringing great damage, loss, or destruction" [11]. Disasters can be caused naturally by events such as hurricanes, earthquakes and avalanches, or can stem from human influenced events such as terrorist attacks, industrial accidents or public policy mistakes. A disaster may result in many human casualties, severe property and environmental damage, and social unrest. Whatever their cause, disasters throughout history have had tragic consequences.

Natural events such as the 2011 Tohoku earthquake and tsunami off the coast of Japan resulted in over 15,000 casualties with many more people left homeless [12]. Human influenced events such as the September 11th World Trade Center attacks in the United States [13], a similar attack on the U.S. Pentagon building and the 2013 Rana Plaza collapse in Bangladesh [14] have resulted in thousands of people killed outright, severely injured and traumatized. Local resources may require aid of regional or national government agencies to assist when dealing with the disaster. Government agencies and enterprises of all sizes plan for disasters by mitigating

potential risk and developing action plans that allow for reaction to events, continuity of operations and recovery.

An **emergency** can refer to two different descriptions. A minor emergency may involve a residential fire or car collision. These events occur more regularly and usually involve a low number of casualties and damage. Serious emergencies refer to events that happen in the future having broader implications. For example, public notification of an oncoming hurricane allows time to prepare. A **hazard** is an event that poses a threat to human life, property, and the environment. For example fires or gas leaks.

To minimize damage and react accordingly when a disaster occurs, emergency management preparations must be made. Emergency managers have the skill, knowledge, and experience pertaining to disaster management to reduce liabilities as the event unfolds. Emergency management is an ongoing effort "—which all individuals, groups and communities manage hazards in a collective effort to avoid or reduce the impact of the hazards [15]."

Figure 2.1: Emergency Management Cycle [15].

The **Emergency Management Cycle** is a "best-practices" strategy for minimizing risk before, during and after disasters, as shown in Figure 2.1. If no effort is spent in preparing for disasters, preventable hazards can turn into emergencies. Taking appropriate actions at each stage of the cycle allows for greater preparedness, earlier warning systems, reduction in damage, and the prevention of other disasters. There are four stages in the emergency management cycle:

- **Mitigation:** Preventing or reducing the severity of disasters. Examples of mitigation are strengthening structures against storms or insuring against economic loss.
- **Preparedness:** Preparing, learning from past disasters, and creating specific actions plans to be followed. For example, a stored emergency supply kit can make the difference in survival.
- **Response:** Allocating emergency services to the disaster area during and after an incident to protect human life and property damage.
- **Recovery:** Attempt to restore an affected area back to normal. Examples include providing medical aid, rebuilding structures, and repairing infrastructure.

During the response phase of the Emergency Management Cycle, emergency response services are immediately deployed to the affected area. The term "first responders" are personnel that generally include firefighters, police, and emergency medical services (EMS) first arriving to the site. If a disaster is small enough, the local response teams are adequate. If a disaster is large enough (i.e., high magnitude earthquake, terrorist attack, etc.), specialized response teams can be called upon to assist with the relief effort.

2.2 Urban Search and Rescue (USAR)

USAR is a specific term for a specialized group of skills and activities related to Search and Rescue (SAR) operations in areas dominated by numerous buildings of complex construction in close proximity—normally found in cities or similar urban areas. USAR organizations provide "an integrated multi-agency response which is beyond the capability of normal rescue arrangements to locate, provide initial medical care and remove entrapped persons from collapsed structures and other environments in a safe and expeditious manner" [16]. The organization of specialized USAR teams has become accepted practice in many countries that tend to suffer from urban disasters [2, 13]. These teams are often called Task Forces (TFs) and are typically organized and available as regional or national resources that are available for deployment where needed. As USAR is a relatively new discipline, TFs have only been in existence since the mid-1980s with the first Canadian TFs being formed in 2006. Among other professions, USAR TFs personnel consist of firefighters, policeman, EMS, doctors, structural engineers, radio operators, canine handlers, and heavy equipment operators [17]. When a disaster is formally declared, an USAR TF may be activated and dispatched to assist in the response to it.

The components that make up a TF are:

- **Search**: locating survivors.
- **Rescue**: extricating or otherwise retrieving survivors.
- **Technical**: inspecting structural components, operating specialized equipment and heavy machinery.
- **Medical**: provide medical care to survivors and other persons as required (including members of the TF).

A typical organization chart for a TF is shown in Figure 2.2.

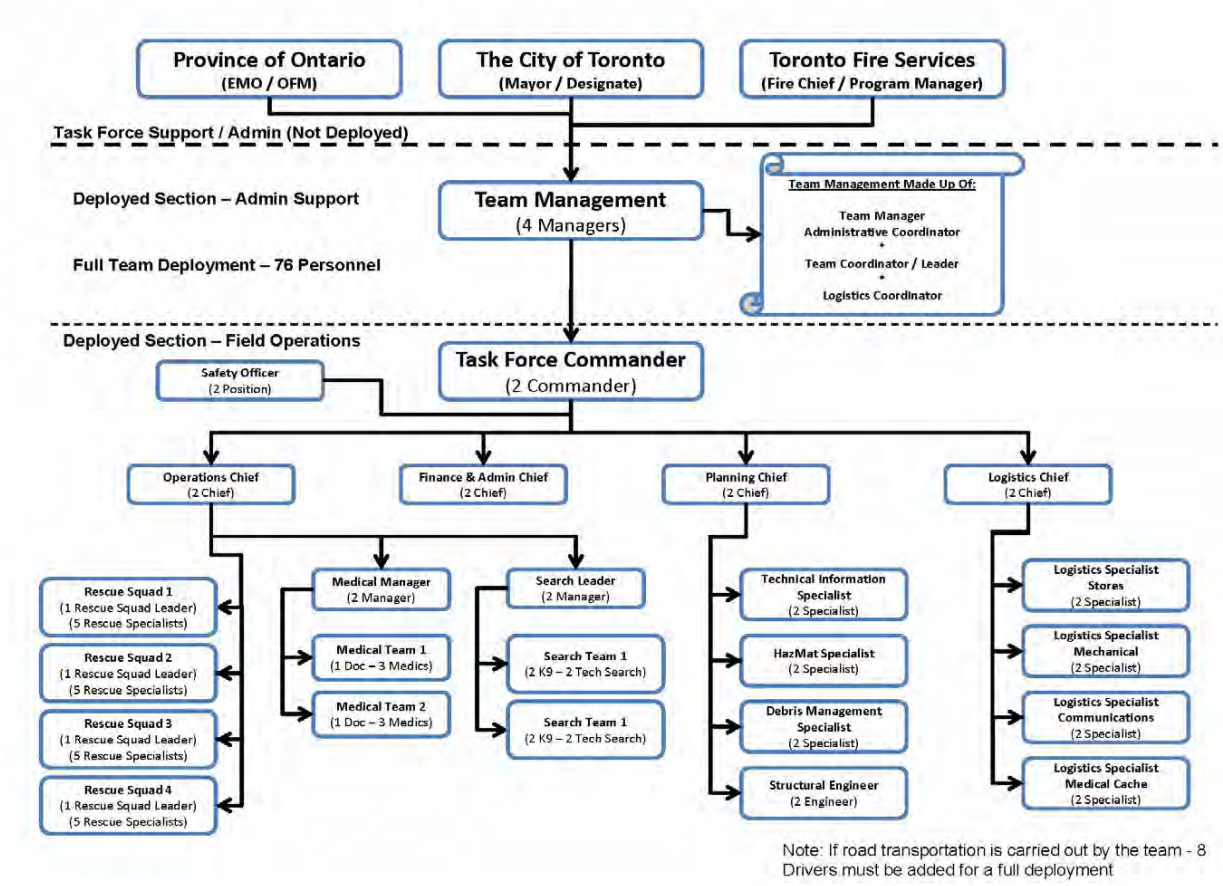


Figure 2.2: Canada Task Force 3's organizational chart [18].

Public Safety Canada maintains the Canadian Urban Search and Rescue Classification Guide [18] to define the standard tools, equipment, and supplies suitable for different USAR TF operations. The teams are classified under three levels of capability: light, medium and heavy. Light USAR TFs respond to incidents within a jurisdiction and are sustainable for up to twelve hours. Medium USAR TFs can respond within mutual aid boundaries (across jurisdictions) and are sustainable for up to 24 hours. Heavy USAR (HUSAR) TFs can respond to incidents nationally and are to be sustainable for up to 10 days with resupply every three days. Initially, HUSAR TFs carry all the equipment and supplies to sustain the team for 72 hours.

In this work, we concentrate on HUSAR operations in the Canadian context. There are currently four HUSAR TFs in Canada. They are strategically stationed across the country in four provinces:

- CAN-TF1 Vancouver, British Columbia
- CAN-TF2 Calgary, Alberta
- CAN-TF3 Toronto, ON
- CAN-TF4 Province of Manitoba

An example HUSAR TF is CAN-TF3 (TOR). Stationed in Toronto, the team consists of 76 personnel in a wide variety of roles and can be deployed within six hour notice.

2.2.1 Initial Inspection and Search

It is the task of structural specialists (SS) within an USAR TF to assess the safety of the work environment. This is done through a survey of the collapsed structures that may need to be searched. The SS recommend the appropriate level of hazard mitigation to minimize the risk to TF personnel and trapped victims during the operation. Initially, regions of obvious structural instability are identified to be triaged. The exterior and available interior of the remaining structure is inspected to determine the locations of falling or collapsed hazards. Possible voids or entry points in the debris are noted during this inspection for later exploitation during the actual search operations.

Once the building has been assessed by the SS, a crude plan is created indicating the following:

- found entry points,

- known victim locations,
- locations of hazards,
- how future hazards can be avoided, and
- possible egress route (i.e., stairs, corridors, etc.) locations based on existing building plans (if available).

The TF command staff of the USAR operation is continually provided with recommendations and updates of where additional structural support may be needed, where debris may need to be removed, and what steps to take to avoid additional hazards.

Search teams are then deployed. These teams use various search strategies, advanced sensors and canine teams to locate hidden victims. Canines teams (consisting of a USAR dog and its human handler) are often used to locate humans based on the detection of human scent by the dogs and signaled through barking [19]. Audio equipment can be used to listen for survivors who may be calling, breathing, tapping or otherwise producing sounds [20]. Increasingly, advanced sensors and rescue robots [13] may form part of the arsenal of search technology available to help locate live victims or inspect dangerous or inaccessible areas like those contaminated by HAZMAT [21] or cracks and fissures in the structure that may be too small for human entry.

Rescue operations proceed in tandem with search operations. Survivors that can be easily extracted are assisted first. Trapped survivors beneath debris may require additional effort to be reached. To rescue trapped victims, debris, including concrete, metal, and other building materials, must be carefully removed to avoid adding to the predicament of the victims beneath. Found survivors are attended by medical staff to stabilize them in order to facilitate their extrication and are then transported to a medical facility for further treatment.

2.2.2 USAR Equipment

A USAR TF typically carries a wide variety of standard equipment to accomplish its mission when deployed. This equipment ranges from medical supplies and equipment used to stabilize injured victims, tools to create structural supports, and mechanical equipment used to remove debris during search and rescue operations. All TF members wear personal protective equipment (PPE). Standard supports ("shoring") are constructed mostly of lumber. Therefore, basic tools such as hammers, levels, measuring tapes, and power tools (i.e., drills, saws, jack hammers) are used for different construction tasks associated with creating support systems out of wood. Wooden support structures are discussed further in Section 2.2.3. Other tools such as generators and lights are used to power and light a working environment so that operations can be continuous.

An incident may require a TF to remain and work for a relatively long period of time. To allow continuous operation a TF stores a cache of supplies at a Base of Operations (BoO) at the disaster scene for up to 10 days. The BoO provides a location where planning, logistics and other operations can be performed. The BoO typically includes accommodations (sleeping, eating, and washing) for the TF, command facilities and anything else that may be required for a particular operation.

The structural collapse of a building typically results in tons of debris having to be removed before USAR operations can commence. If heavy machinery is available, it is used by trained operators to carefully remove debris to prevent further damage or collapses. Other tools like high-powered cutting torches and concrete saws are used to breach walls providing passage through the debris. A full list of available equipment can be found in the Canadian USAR Classification Guide [18].

2.2.3 Shoring

In USAR, **shoring** "is the temporary support of only that part of a damaged or partly collapsed structure that is required for conducting operations at reduced risk" [20]. A structural specialist inspects a debris field to determine whether an area needs vertical or lateral support. Used as a last resort, unstable structures are shored if they cannot be avoided or removed altogether. Different shoring techniques can be used and various configurations exist that are constructed using any or all of lumber (wood), hydraulic and pneumatic jacks or struts, and high-pressure air lifting bags. By far, the most commonly employed material is lumber. For the purpose of this thesis, wooden shores are the primary focus.

The engineering concept behind all shoring systems is called the double-funnel principle (see Figure 2.3). The objective is to collect a load produced by forces on structures within rubble and transmit it through support posts for distribution to a stable supporting structure. A shore system is built to give warning signs before failure. A unique property of wooden shores is that they provide audible indications of impending failure through loud cracking sounds. This warning allows time for responders to reinforce the shore before its failure leads to an additional collapse.

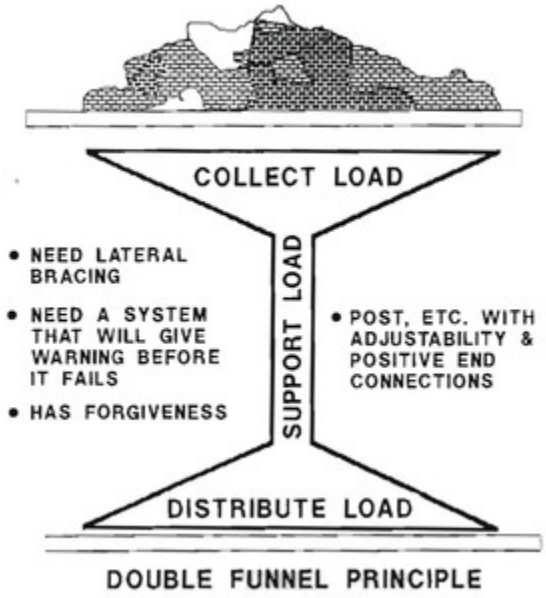


Figure 2.3: The Double Funnel Principle is the engineering concept of how a shore supports and distributes a load [20].

Various shoring options are available, each supporting specific load capacities, and are built in a specific way using defined materials to produce consistent results. Unstable areas such as a partially collapsed floor or unstable walls are example areas that may require support through appropriate shoring. The shoring method used is dependent on the direction of the load. The two categories of shores are vertical and lateral. For example, to support a load vertically two T-spot shores can be employed to support weights up to 4,000 LBS as shown in Figure 2.4 at the bottom. A laced-post box shown in Figure 2.4 top-right can support weights up to 80,000 pounds.

Figure 2.4: Shoring configurations within rubble. **Top-Left:** Window/Door shore used to provide safe entry. **Top-Right:** Laced-Post Box shore used to support the floor/ceiling above an area. **Bottom:** Two T-Spot shores supporting collapsed area.

Wooden shores are constructed using nails and wood components of varying sizes. The common lumber dimensions for various shore components are 6”x6”, 4”x4”, 2”x4”, and ¾ inch plywood. An area needing support is measured and, depending on the estimated weight to be supported, a shore will be selected for construction. Lumber is cut to certain sizes at a “cut table” and is assembled based on guidelines provided by standards such as the FEMA Field Operations Guide (FOG) [20].

2.3 Robotics in USAR

Robotics can be used for the purpose of emergency response as they provide an alternative method of inspection to USAR TFs when searching the debris. Areas unsafe or hard to reach can be surveyed remotely using robotics and their employed sensors. Information is then provided about the terrain without having to be manually inspected.

2.3.1 Ground-Based Robotics

Robots that navigate on the ground typically employ different combinations of wheels and tread tracks; see Figure 2.5 for examples. The police and army use robots to remotely inspect and diffuse possible hazards, such as bombs [22]. In USAR, collapsed buildings can be inspected to search for signs of life. For example, after the Twin Towers collapse in New York, a team of robots from the Center for Robot-Assisted Search and Rescue (CRASAR) deployed 17 robots to search the wreckage [23].

Figure 2.5: Examples of robots being used in a USAR application.

Communicating with ground robots is achieved either wired or wirelessly. The CRASAR robots were attached to 30 meter tethers. A tether ensures a constant power source and communication pipeline to the robot. The distance a tethered robot can travel depends on the

length of the tether itself. Also, tethers have been reported to snag on surrounding terrain making the robot difficult to control [23].

Different forms of wireless communication such as WI-FI and radio can be used to remotely control the robots. Terrain far away can potentially be reached without the restriction of an attached tether. During a USAR operation, obstructing debris is detrimental to wireless signal strength. Dense concrete has shown to degrade a wireless signal to the point of lost communication [24]. Very expensive robots can be lost within the rubble and unable to be retrieved due to the inherent danger.

A defining challenge faced by ground robots when navigating collapse debris is traversal. The types of terrain to be negotiated can range immensely, making terrain navigation very difficult for most styles of robots. Robots have been abandoned after becoming stuck on terrain during a rescue mission [23]. Flying over terrain can avoid most of these pitfalls.

2.3.2 Aerial-Based Robotics

Unmanned Aerial Vehicles (UAVs) are robots that are remotely controlled to fly over terrain. Militaries across the world use large UAVs called “drones” to conduct reconnaissance missions [25]. Smaller, non-lethal UAVs are becoming more common and have many advantages over ground-based robots during USAR.

By flying over an emergency site using a UAV, the obstacles faced by ground-based robotics are avoided. A UAV can be used to avoid a degrading wireless signal due to concrete. Areas that are dangerous and not easily accessed can be viewed aerially. Large areas (common in an emergency) can be surveyed to be later analyzed. The disadvantage is that confined areas

cannot be scanned such as collapsed building interiors because of difficult flying conditions. Also, weather can be a factor.

The first known report of a UAV being used in a rescue was by the Royal Canadian Mounted Police (R.C.M.P.) in Saskatchewan to locate an injured man in the woods [26]. The man was located using a four-rotor UAV with an attached thermal sensor.

In this work, a lightweight hexacopter UAV and a RGB-Depth sensor is used to collect data and create a 3D representation of an urban disaster scene [5]. The data is discussed further in Section 2.5. The used UAV and an example frame of collected raw video and depth data is shown in Figure 2.6. This previous work was used as a basis in creating 3D environments of real-world disaster terrain for this thesis.

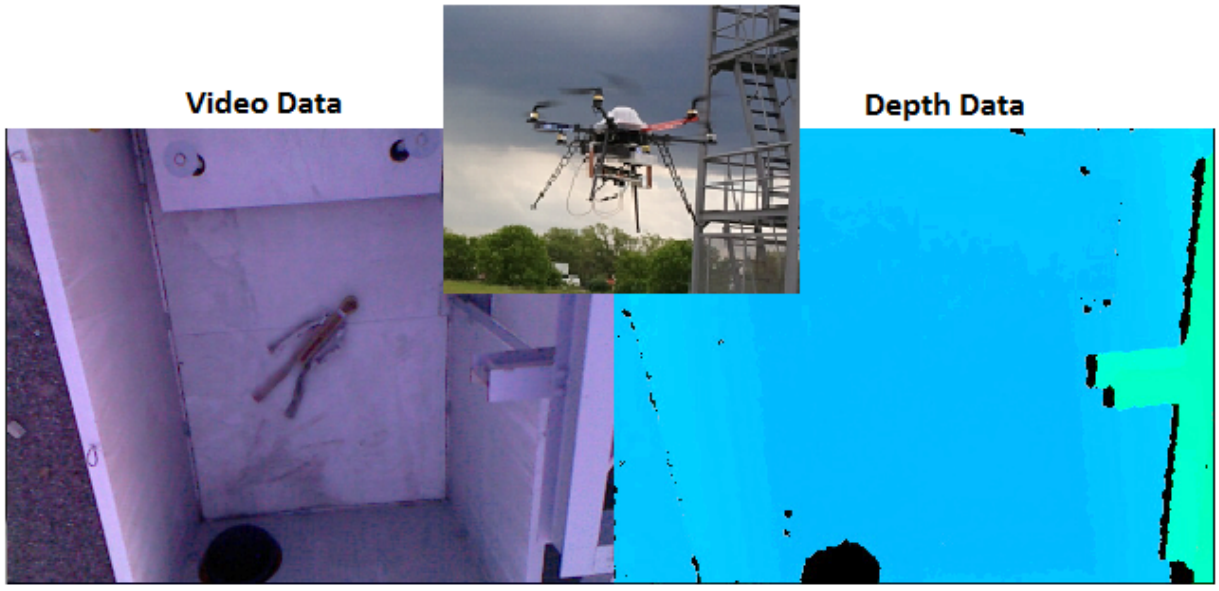


Figure 2.6: The UAV used to collect data with an attached Kinect sensor and the resulting video and depth data.

2.4 Sensors

A variety of sensors can be employed by robots to convey a better understanding of an environment. In USAR, robots searching for trapped victims are only possible because of the sensors that are employed. Examples of robots using sensors in disaster scenarios are plentiful [5, 22, 23] and typically involve the remote inspection of hazardous terrain. Different sensors available include various forms of cameras such as low-light or infrared, Laser Range Finders (LRFs) [27], and Sonar arrays [28].

Once a robot enters a collapsed structure, communicating with the robot can be difficult for a number of practical reasons. Wireless communications have been shown to quickly degrade through reinforced concrete [29]. One of the reasons that robots have not been deployed in greater numbers is that the robot/sensor package can be very expensive and would be a huge financial loss if a robot, by necessity, would need to be abandoned within an urban collapse. Various sensors can be carried by a UAV, but they are limited by their lift capacity.

With the release of the light-weight, low-power, Microsoft Kinect depth sensor [30], robot sensing has entered a new era considerably more flexible in terms of the type of data that can be collected and the costs associated with it.

Essentially, the Kinect is built with a RGB camera and infrared laser projector sensor allowing for 3D video data to be captured under proper lighting conditions. The video is captured at a resolution of 640x480 in 32-bit colour at a rate of 30 frames per second. The horizontal and vertical field of view (FOV) is 57 and 43 degrees respectively. Various researchers have used the sensor to collect spatial data in rather creative ways [31, 32]. To a certain extent, the Kinect has provided a cost-effective replacement for similar expensive sensor packages. However, the

Kinect is not without its limitations. The optimal scanning distance is between 2 and 8 meters. The infrared (IR) data can contain errors if areas in the FOV are reflective or the Sun is being shone on a surface due to IR interference.

The Kinect development community is large and growing. There are a plethora of application areas including motion and object tracking[33], gesture tracking [34], and 3D scanning [32, 35].

2.5 Point Cloud Data

Point cloud data or "points" are a set of vertices typically plotted in three-dimension (3D using XYZ coordinates and can have associated colour attributes. The point sets can be derived using a variety of sensors including Laser Range Finders (LRFs) [27], Infrared (IR) sensors [36], and the Kinect [30]. Many companies such as Toyota [37] and Leica [38] have expressed interest in point cloud-related research including 3D scene processing. They support a community of developers contributing to an open-source development library called the Point Cloud Library (PCL) [39]. Many other point cloud applications exist to either view or process data by integrating a graphic user interface (GUI). The relevant functionality available is further discussed below.

2.5.1 Point Cloud Processing

Multiple tools and Application Programming Interfaces (APIs) have been developed and are actively maintained to process point cloud data. The PCL is an open-source API project that has a community actively adding new functionality to create and process point clouds. PCL lacks visual feedback during the processing and you cannot compare models before and after requiring additional time to inspect the output.

Meshlab [5] is a freely available application that uses the Visualization and Computer Graphics Library (VCG) [40]. The functionality is similar to PCL with the added benefit of an easy to use GUI and was chosen as the primary tool in this thesis. Changes can be made to a data set and viewed immediately after processing allowing for dynamic inspection. Processing these point clouds allows developers to alter various aspects of the data, either by reducing the overall size, fixing anomalies such as stray points/redundant points, and creating a 3D model. Meshlab provides the use of many surface reconstruction techniques to create 3D models [41-43]. Section 2.5.2 describes the best performing methods.

2.5.1.1 Point Cloud Filtering

A common technique for reducing the size of the point cloud data is to remove point redundancy created from noise in sensor feedback. Many techniques exist [44, 45]. Removing points within a defined proximity to each other can reduce the file size while still maintaining the model's quality. This was first shown through stochastic sampling by Cook using Poisson Disk Sampling [45]. During Poisson Disk Sampling, a point is randomly selected (called "dart-throwing") and is compared with *k* random samples of points within a spherical annulus radius called a disk, *r* to *2r*. The point is rejected by the algorithm if it comes into contact within another already visited point. The resulting point set ensures no two points are within the defined disk area as shown in Figure 2.7. Since the points are randomly chosen, the resulting points will be different each time. The point reduction allows for further processing to be conducted much more efficiently.

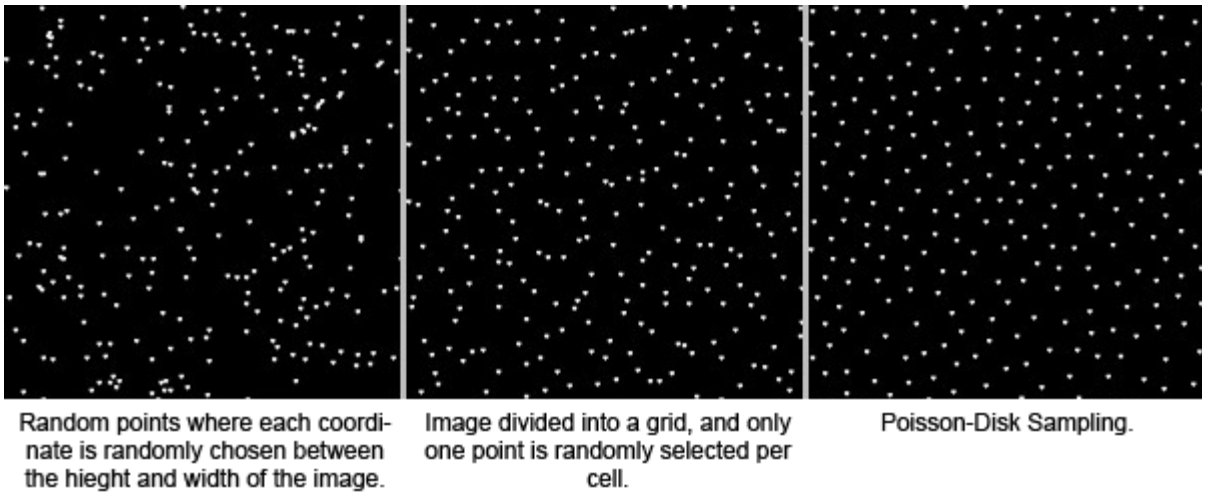


Figure 2.7: Example of point distribution methods.

2.5.1.2 Normal Estimation

Point normal estimation [46] is an important aspect in creating a 3D mesh surface. In computer graphics, a normal is computed for a surface so that lighting can be applied and be properly lit or shaded when rendered. . A normal vector is calculated from the cross product of two non-parallel edges in a convex polygon. For point clouds each point's normal is found by using the neighboring vertices to calculate a plane. Then, use the extrapolated plane vertices to perform the cross product operation. Depending on the order of the cross product operation the normal will face a different direction. See Figure 2.8 to see a surface with calculated normals.

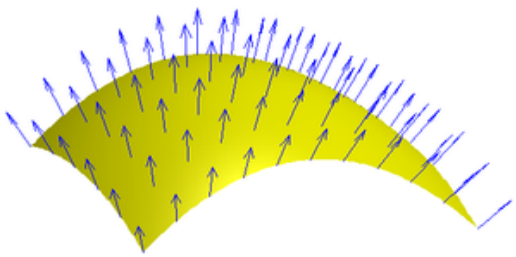


Figure 2.8: An example surface with calculated normals.

2.5.2 Surface Reconstruction

The process of constructing a surface (called a mesh) from a point cloud data set is called surface reconstruction and many techniques exist [41-43]. Meshing is a conversion step that uses an existing point cloud to create a mesh comprised of polygons. A mesh model is an essential requirement when used within a game engine environment in order to provide basic functions like collision detection between objects (discussed in section 2.9). Various methods for creating a point cloud mesh are available but only those that are applicable to this work are explained in some depth.

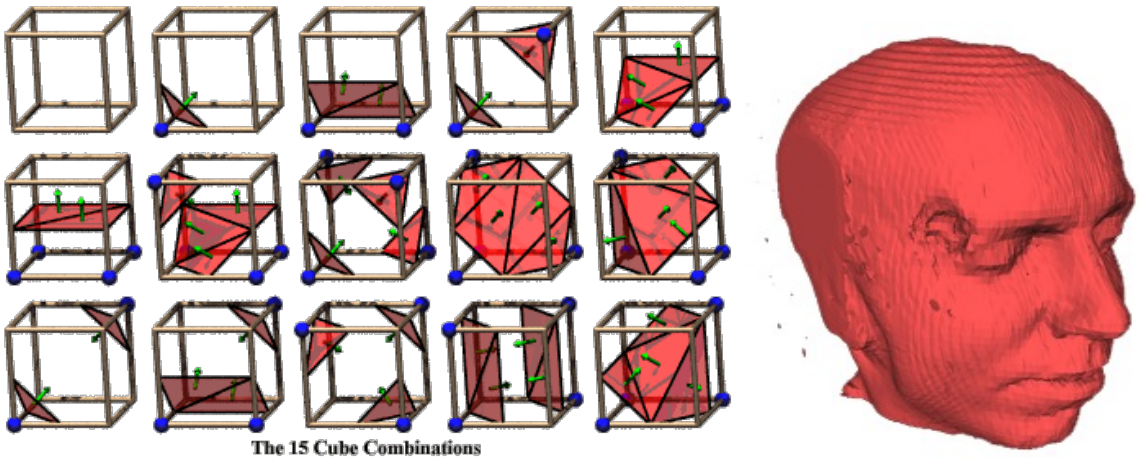


Figure 2.9: Marching Cubes Surface Reconstruction [42]. **Left:** The 15 possible cube combinations. **Right:** An example mesh using marching cubes.

A volumetric method developed by William E. Lorensen called Marching Cubes (MC) [42] divides space into a grid of cubes (voxels), only retaining the cubes that contain data. The 3D mesh is approximated using linear interpolation within a cube. Using a lookup table of 15 possible configurations (see Figure 2.9), vertices and polygons are created within the voxel to be used in a final mesh. Which configuration is chosen is based on each voxel's scalar values and whether they are higher or lower than the isosurface. The more points that are retained the better

approximation. In practice, marching cubes can be applied quickly for datasets of several million points. An example of a mesh produced by this method is shown in Figure 2.9.

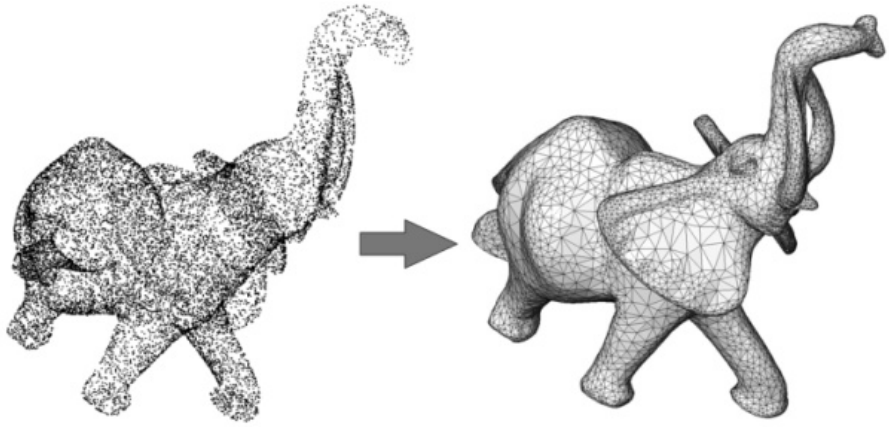


Figure 2.10: A point cloud set with little noise and the resulting Poisson reconstructed surface.

Poisson surface reconstruction [43] can be used to create a "watertight" model (exempt of holes or voids) that has a smooth surface. The points are broken down into voxels and then stored in adaptive octrees (trees where each internal node has exactly eight children). A user specifies the depth of the octree, and the higher the depth, the more memory and processing time will be required. Storing more points allows for better approximation and a more defined 3D model. Using each point's normal, an implicit function is computed of the inferred solid whose gradient best matches the input normals. The scalar function output, represented in the tree, is then iso-contoured (connecting similar trees) using marching cubes. Since the surface produced is watertight, voids in the data will be removed as they are filled. For the purposes of having a watertight and smooth surface, Poisson reconstruction is preferred. An example of a Poisson reconstructed point cloud is shown in Figure 2.10.

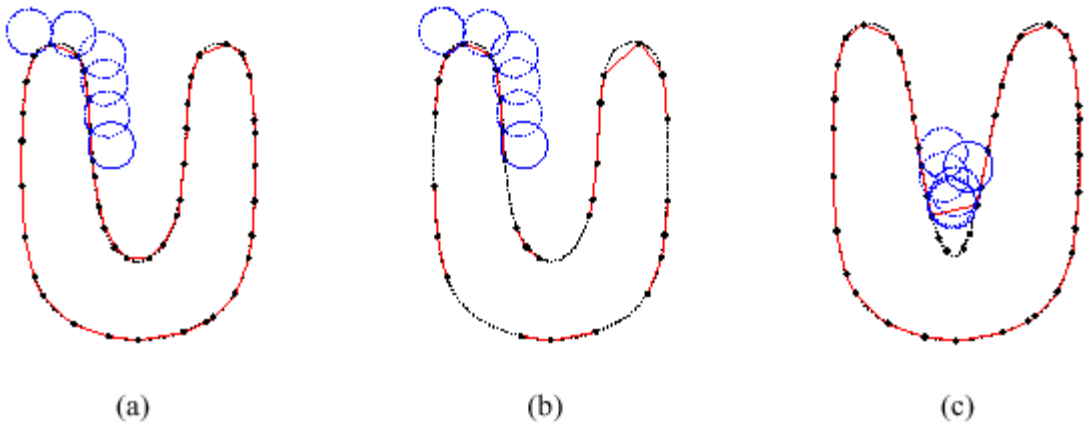


Figure 2.11: Ball Pivoting Algorithm for surface reconstruction applied to a 2D point data set. **A:** A ball pivoting along the vertices to create a surface. **B:** The ball is unable to reach a vertex because the radius is not large enough and a void is created on the surface. **C:** If the ball has a too large of a radius, vertices positioned sharply from each other may not be reached, in effect creating lower resolution meshes.

The Ball Pivoting Algorithm (BPA) is a simple surface reconstruction technique [41]. Initially, a starting vertex is selected and a ball with a defined radius ρ pivots around the vertex until another vertex intersects the ball. If there is an intersection, an edge is created between the vertices. Then, the intersected point becomes the next selected point and the previous steps are repeated.

Figure 2.11 provides examples of the BPA in use and certain limitations. If a point set is not uniformly distributed, using BPA may not be ideal. Vertices in the point cloud data that are farther than ρ from each other will not be reconstructed due to the ball never coming into contact with another vertex as shown in Figure 2.11 (b). Evenly distributed points in certain environments—especially those associated with disaster rubble point clouds—require a radius that is not too large to avoid over-simplification in the varying terrain. Using BPA is applicable in that voids are not unusual in disaster rubble point clouds (i.e., openings or holes in the rubble scene, etc.) since they are literally nothing and should not be reconstructed. Another challenge with BPA is if the radius is too large, sharply positioned points may never be reached as shown

in Figure 2.11 (c). Inversely, a radius too small can miss vertices by never coming into contact with the ball as shown in Figure 2.11 (c). Thus features can be missed altogether.

2.5.3 Mesh Decimation

The final mesh model processing step is to reduce the overall polygon count by generalizing a surface through decimation [47]. A reduction in the overall polygon count of the mesh model will allow better processing when rendered.

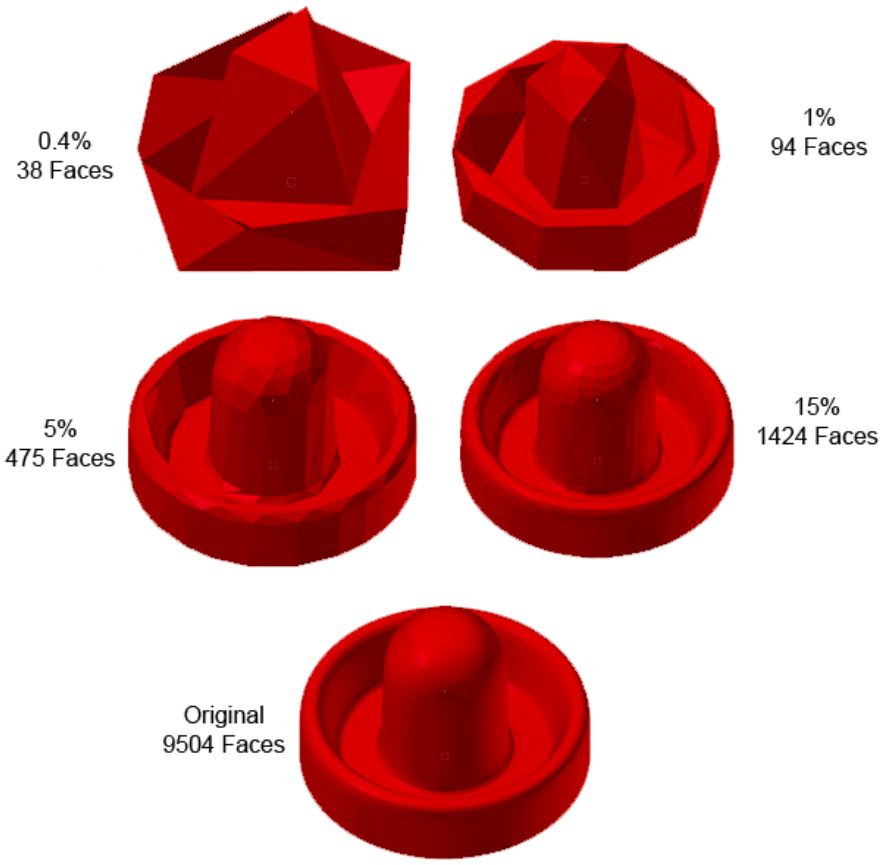


Figure 2.12: Example of a mesh being decimated [48].

Multiple passes are made over all the vertices in the mesh. Each pass chooses candidate vertices for removal based on a decimation criterion such as being within a defined distance to a

plane. If it meets the criteria, the vertex and connected polygons are deleted. The remaining void is then patched using the surrounding vertices. The termination criteria is a percent based reduction of the original polygon count. See Figure 2.12 for an example of a model and three percent reductions and respective models.

2.6 3D Environment Model Creation

Creating 3D models of real-world environments has gained popularity because of the availability of low-cost sensors like the Kinect. Using the point processing techniques previously described, real world environments can be modeled. Many examples exist using the Kinect sensor [36, 49]. Perhaps the most impressive is the Microsoft Research team's application called KinectFusion [32] which uses the sensor to dynamically create fully 3D meshed environments, in real-time, of a small area (four cubic meters). See Figure 2.13 for an example of KinectFusion.

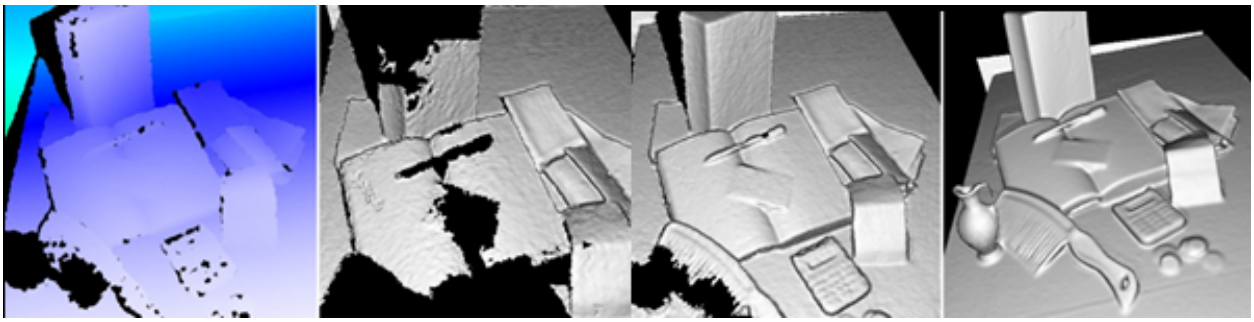


Figure 2.13: The KinectFusion 3D process from registration to model creation (left-to-right) [32].

An extension of this technique, called "Kintinuous" [35], extends the scannable area to a user defined size, but is limited to the amount of memory resources available when scanning. A hardware reliance of Nvidia's parallelizable graphic card technology—Compute Unified Device Architecture (CUDA) [50]—is needed to achieve high frame rates while constructing and smoothing the approximated mesh.

2.7 Computer Simulations

A computer simulation can be defined as representing real-world systems and their internal processes using a computer program. This can include real-world or hypothetical systems. A simulation has the benefit of allowing many test configurations to be run without substantial cost in time, materials, and effect. For example, testing nuclear bombs is a practical system to be simulated, avoiding all consequences and associated effort [51]. Simulating different types of building collapses require large amounts of resources, space, and time. Simulating disasters has been attempted in order to better understand their various characteristics. This is discussed in the following sections.

The use of computer technology has allowed researchers to create simulations of natural and man-made disasters. A simulation can be used to test large arrays of environment configurations by easily changing program variables. Disasters, in effect, can be replicated for further analysis and applied for various purposes including training [7, 52].

Natural disasters are not easily replicated due to their complexity and scale. The Hanshin-Awaji earthquake in Japan which claimed more than 6,400 lives [53] resulted in a simulation system being developed to better understand the implications of wooden houses collapsing [54]. Simulated wooden structures representing Japanese houses were subjected to artificial earthquake tremors which would affect those structures. Each structure was modeled with typical materials found in houses using their weight, dimensions, and load/stress characteristics of each component. The backend physics engine—Nvidia PhysX [55]—was used to determine the rigid body dynamics. At the time of publication, general-purpose computing on graphics processing units (GPGPU) was not available, and as such, it was reported that the simulation took about 10 minutes for one trial due to the computational resources required by the physics engine.

However, by using a simulation, the human cost of studying a full-scale building collapse is eliminated. Indeed, the simulation could be changed to measure any number of phenomena without effecting society.

Figure 2.14: Screen shot of USARSim during a mock disaster scenario [7].

The Unified System for Automation and Robot Simulation (USARSim) [7] employs the Unreal Development Kit (UDK) game engine [56] to drive a disaster and robot simulation. A screenshot is shown in Figure 2.14. This system has been used as a training platform in pre-created disaster scenarios and for rescue robot testing [7].

The National Institute of Science and Technology (NIST) USAR Arenas are mock-disaster environments that have been successfully simulated using USARSim. NIST's USAR Orange Arena was simulated to test robotics user-interfaces [57]. The system took less than three

months to develop and accurately reproduced the specifications found within the real NIST test environment.

Figure 2.15: Natural disaster simulation from ETC Simulation systems [52].

In the private sector, ETC Simulation [52] has created a simulation system to train individuals in mock scenarios found in fields such as policing, emergency response, and natural disasters as shown Figure 2.15. Using a large curved display and proprietary software, a trainee interacts within the simulation while taking on a specific role resulting in preparedness for an analogous real-world situation. Similar to the previous examples, virtual environments are created prior to each exercise.

A post-disaster environment modeling simulation and an earlier revision of this thesis system—Disaster Scene Reconstruction (DSR) —[10] uses another approach. Taking a point cloud as input as discussed in in Section 2.3.2, the Unity game engine [58] is used to devise an accurate model of the terrain recorded. The previous described simulations differ from this system in that they use fabricated data, or in other words data that is not applicable to an actual

disaster response. Initial experiments were conducted to determine the possibility of measuring human traversal on simulated rubble and how it compared with actual traversal of the real site. The original site and the created model are shown Figure 2.16. Human parameters were used and assigned to a game actor within the simulation. It was confirmed that a game engine can be used to successfully replicate traversal times using a 3D modeled replica of a disaster scene. It was found that after five time trials, the real-life recorded times and the simulation times differed slightly.

Figure 2.16: The mock disaster scene including a partially collapsed building in Bolton, Ontario and the resulting model.

2.8 Game Engines

Game engines are a form of development platform typically used to create virtual environments for interactive video games. “Serious games” are simulations made to inform a user of factual information or represent real-world events and solve a problem not for the purpose of entertainment. Among other applications, they can be used for training [7], education [59], and health [60].

Figure 2.17: Dental Implant Training Simulation [60].

An example serious game is shown in Figure 2.17. The program was developed for the Medical College of Georgia [60] to teach and train dental school students. Functionality includes patient assessment, diagnosis, and practicing dental procedures in a realistic virtual environment.

A game engine allows a developer to create simulations and add functionality not available in alternative visualization schemes. The realism that can be depicted, using simulated physics and graphics, have made these platforms plausible research tools especially when an

environment is particularly complex and the ability to recreate it prohibitively expensive by other means.

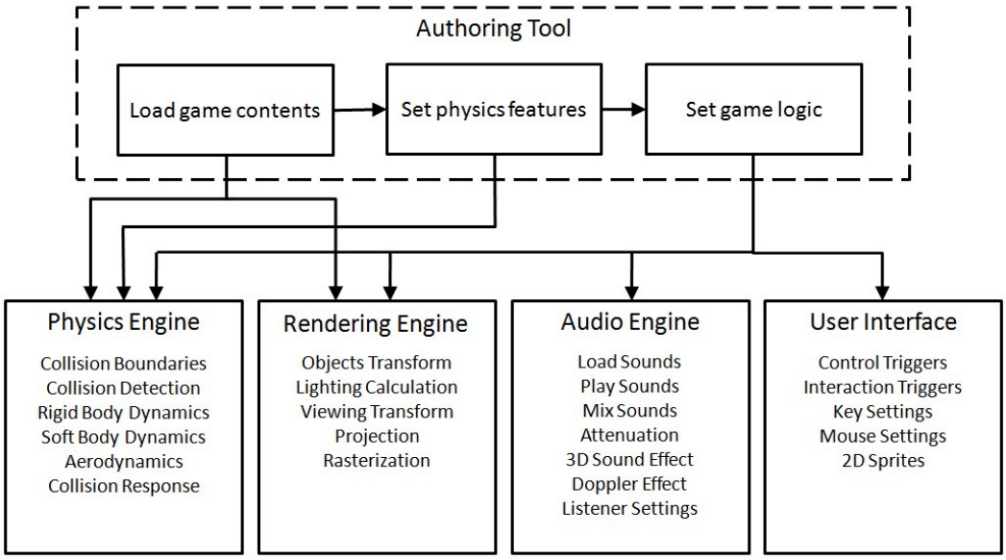


Figure 2.18: The architecture of a typical game engine.

A game engine allows the re-use of components, high-end graphics features, and most importantly, accurate simulated physics. The backend components provided to a programmer can differ between engines, however, common features typically include:

- physics
- rendering
- audio
- graphical user interface
- networking

This functionality is conveniently accessed through high-level API calls. Collections of prefabricated 3D models and prewritten code can be added to any "scene" (game environment) or "player" (game actor). This degree of modularity allows for re-use of resources and a

reduction in development time. The architecture of a common game engine is shown in Figure 2.18.

With the advancement of mobile hardware both graphically and computationally, mobile platforms such as tablets and smartphones are emerging as plausible target systems. Game engines have recently begun to provide mobile development APIs within their frameworks. Popular operating systems such as Google's Android [61] and Apple's iOS [62] are among other possible target mobile platforms.

2.8.1 Unity Game Engine

The Unity game engine is a cross-platform development system made by Unity Technologies [58]. The backend languages used for development are C#, JavaScript (JS), and Boo. Through the use of Microsoft's Common Language Infrastructure (CLI) [63], these languages can be packaged into a similar format allowing for internationalization. The platforms that Unity supports include: Windows, Linux, Mac, iOS, Android, Flash, Playstation 3, XBOX360, Nintendo Wii, and soon the Windows Phone 8 and Blackberry 10. The used backend graphics engine is dependent on the platform as listed: Direct3D (Windows), OpenGL (Mac, Windows, Linux), and OpenGL ES (Android, iOS). The backend physics engine is an industry standard—Nvidia PhsyX. A wide variety of 3D model file types are supported allowing the use of a range of modeling packages such, as 3DS Max [64] and Meshlab.

Since content developed in Unity is modular, developers can access pre-created content by importing it into any project. This allows a developer to avoid unnecessary development time by re-using already programmed scripts. These collections of scripts and content are referred to as "Assets". Many games have been developed using Unity such as [65, 66].

2.9 Physics Engines

The component that provides simulated but realistic interactions within a game engine environment is its backend physics engine. The physics engine handles the mathematical functions related to physical interactions. Typically a physics simulation is broken down into two areas: collision detection and dynamics.

Collision detection determines which objects contact each other through their behavior. Dynamics is concerned with the dynamic motion of objects that exist independently or are affected by constraints, like joints (two objects connected at some point). The detailed processes of these calculations are abstracted to high-level API calls. In a game engine, these function calls are non-transparent and are of no concern to a developer when creating an application. The developer simply provides object parameters (i.e. mass, geometric dimensions, etc.) to be used within the engine. The accuracy of the physics layer within simulators is essential in providing realistic models of the outside world.

Reviews of available physics engines such as Open Dynamics Engine (ODE) and Newton have been conducted [67], but are, for the most part, outdated compared to the current industry standard engines such as Nvidia PhysX and Havok [68]. Typically these engines are compared in a controlled simulation setting to determine accuracy in a variety of physics related themes such as collision detection, hard (rigid) and soft body dynamics, particles, and load bearing.

CHAPTER 3 TECHNICAL APPROACH

In this chapter, the 3D model creation pipeline and simulation architecture created for this thesis are discussed. First, the real-world simulation disaster environment used to collect data is discussed in Section 3.1. How the point cloud data is collected and an overview of the 3D model creation pipeline is presented in Section 3.2. The individual pipeline steps and how they are best tuned for performance are detailed in Section 3.3. The Disaster Scene Reconstruction (DSR) simulation system and purpose-built USAR functionality is discussed in Section 3.4.

3.1 Simulated Disaster Environment

The OPP Reference Rubble Pile (RRP) is a purpose-built confined space and USAR training facility in Bolton, Ontario. It is the only such facility in Canada east of Manitoba. The environment is shown in Figure 3.1 and was used as the primary environment to collect data. Example areas employed in this thesis are highlighted and also labeled for future reference. The environment includes varied types of debris including a partially collapsed building, concrete and wooden rubble, several vehicles and sea containers, and a passenger bus partially buried in the rubble. The RRP provides an excellent aftermath simulation of structural collapses of several buildings, constructed of different materials, each of approximately six stories. The facility is available for research purposes to students associated with the NCART lab based on a Memorandum of Understanding concerning the research collaboration between the Ryerson NCART lab and the OPP UCRT.

Figure 3.1: The purpose-built RRP training facility in Bolton, Ontario owned by the Ontario Provincial Police. **A:** A collapsed structure simulating a floor collapsing onto another. **B:** A passenger bus partially buried in the rubble.

3.2 Attaining Data and the Model Creation Pipeline

For this work point cloud data is provided from a UAV and RGB-D sensor collection and registration technique shown in [5]. Disaster rubble is typically dangerous to traverse and often provides obstacles that are impassable to humans on the ground. Any inspection is difficult to accomplish and can be very time consuming on foot. Typically, time is of the essence during response operations to maximize the survival rates of people caught in the rubble. For example, a USAR TF may want to inspect an area across the debris, but it could take hours to reach due to hazards such as an unknown structural stability. Through the use of a UAV the challenges of ground mobility are avoided through rapid, remote sensing from above. A typical flight length ranges from under a minute to many minutes. A goal of this thesis is to use the point cloud data

collected from a RGB-D equipped UAV to investigate whether a simulation tool could be created to better inspect modeled rubble and interact with it in an accurate physics-based manner. For the point cloud data to be compatible with game and physics engines, the point cloud data must first be converted into a 3D mesh model.

Unfortunately, it is not possible to convert the point cloud data into a 3D mesh model by naively running the point cloud data through a surface reconstruction algorithm. In all likelihood, the resulting mesh will be severely disorganized due to the large amount of outlying and redundant (tightly positioned or overlapping) vertices in the resulting mesh. From a 3D modeling perspective, vertex redundancy must be minimized to reduce the number of polygons which supports faster rendering without sacrificing model resolution quality.

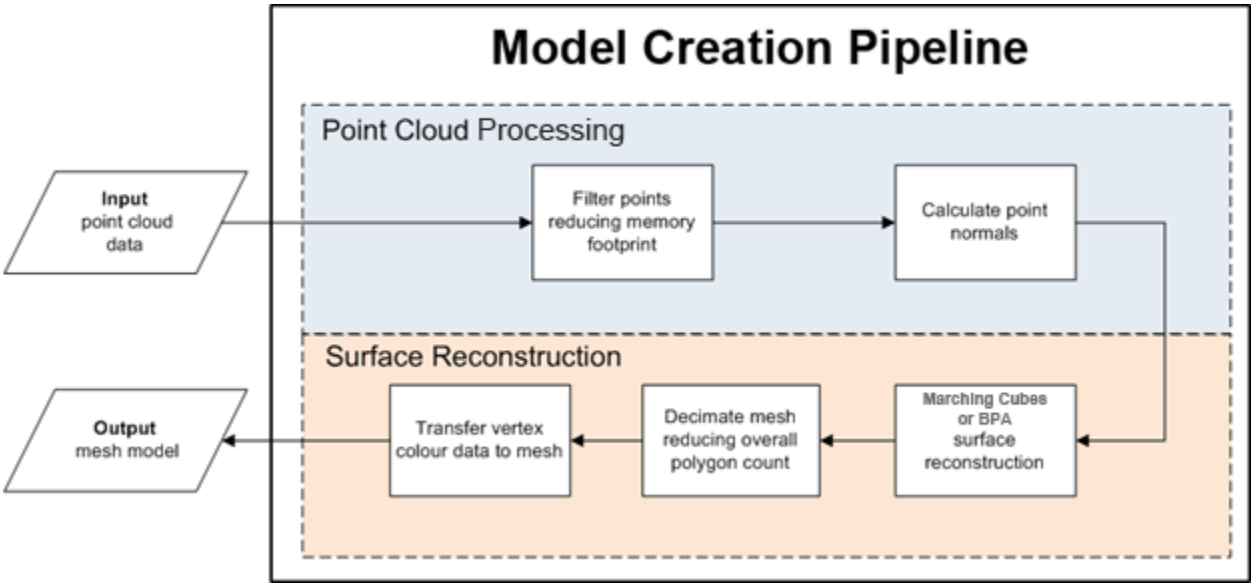


Figure 3.2: The mesh Model Creation Pipeline chart.

The 3D model creation pipeline is shown in Figure 3.2. The process has two phases consisting of point cloud processing and surface reconstruction. Before a point cloud can be used within a game engine environment the data must be converted to a compatible format. Various

filtering techniques can be used to reduce the overall memory size and increase the quality of the point cloud before a solid surface is created. The point cloud processing steps are described in Sections 3.3.1 and 3.3.2. After the point cloud is filtered, surface reconstruction is applied to the point cloud data in order to create a 3D mesh model. These steps are described further in Sections 3.3.3 to 3.3.6. We can produce models from input point cloud data under 5 hours which is a practical timeframe to provide the simulation to USAR personnel as it may take many hours to reach a disaster site.

3.2.1 Meshlab

The Meshlab modeling application [5] is used as the primary tool during the modeling creation pipeline. Meshlab provides a GUI comprised of a 3D model viewer and menus containing modeling algorithms. As algorithms are used, a new model is added to the application as a "layer" and can be seen within the viewer. A layer in Meshlab contains data independent of other layers and their visibility can be turned off and on. A useful feature is a user can turn off/on the rendering of each layer to compare and see changes. This allows for rapid inspection and can be used to compare parameter changes. Runtimes are displayed within the GUI for each step and are used to determine how long the modeling process takes.

3.3 3D Model Creation

The point cloud data attained from the RGB-D equipped UAV method is not compatible within a game engine. However, the point cloud data set contains vertices which are the basic component used in 3D modeling. Two vertices connected by a straight line become an edge. Three vertices, connected by three edges create a triangle—a simple polygon. Therefore, the point cloud data can be used to create polygons that are used to comprise a 3D model. The

following sections describe the various data conversion techniques that can be employed in the support of creating a model that can be used within a game engine environment.

3.3.1 Filtering

The data collection and registration technique [5] used for this thesis produces point cloud data from multiple overlapping scans that can result in a large number of redundant points. An unfiltered point cloud model may consist of tens of millions of points – too large for a game engine to process efficiently on a gaming computer with an i7 processor and GeForce GTX680M GPU. The first step in the model creation pipeline is to further filter the data in order to reduce the overall point count without losing data quality. This, in turn, allows for faster processing later in the model creation pipeline and improved efficiency when rendered in a game engine.

The Poisson-Disk Filter (Section 2.5.1.1) technique is used within Meshlab to spatially distribute the points evenly. The filtering algorithm suggests randomly selecting a point (called "dart throwing") from the set of available points and checking if any of the remaining points are within a defined spherical radial distance. If another vertex is found to be within that distance, it is removed from the point set. Then the remaining points in the set are exhaustively run through the same process.

In this work, filter distances of 10mm for point clouds less than a million points and 15mm for point clouds more than a million points were used. This removed most redundancy while maintaining the details of the environment. An example point cloud (before and after filtering is applied) is shown in Figure 3.3. The model was created from data obtained in area **A** as shown in Figure 3.1. The original set consisted of 15,101,504 points. After the filter was

applied the resulting set consisted of 1,851,614 points. This produced an 87.74% point reduction while maintaining the appearance and qualities of the original data set. Different radial distances were experimented with and are discussed in Section 4.2.

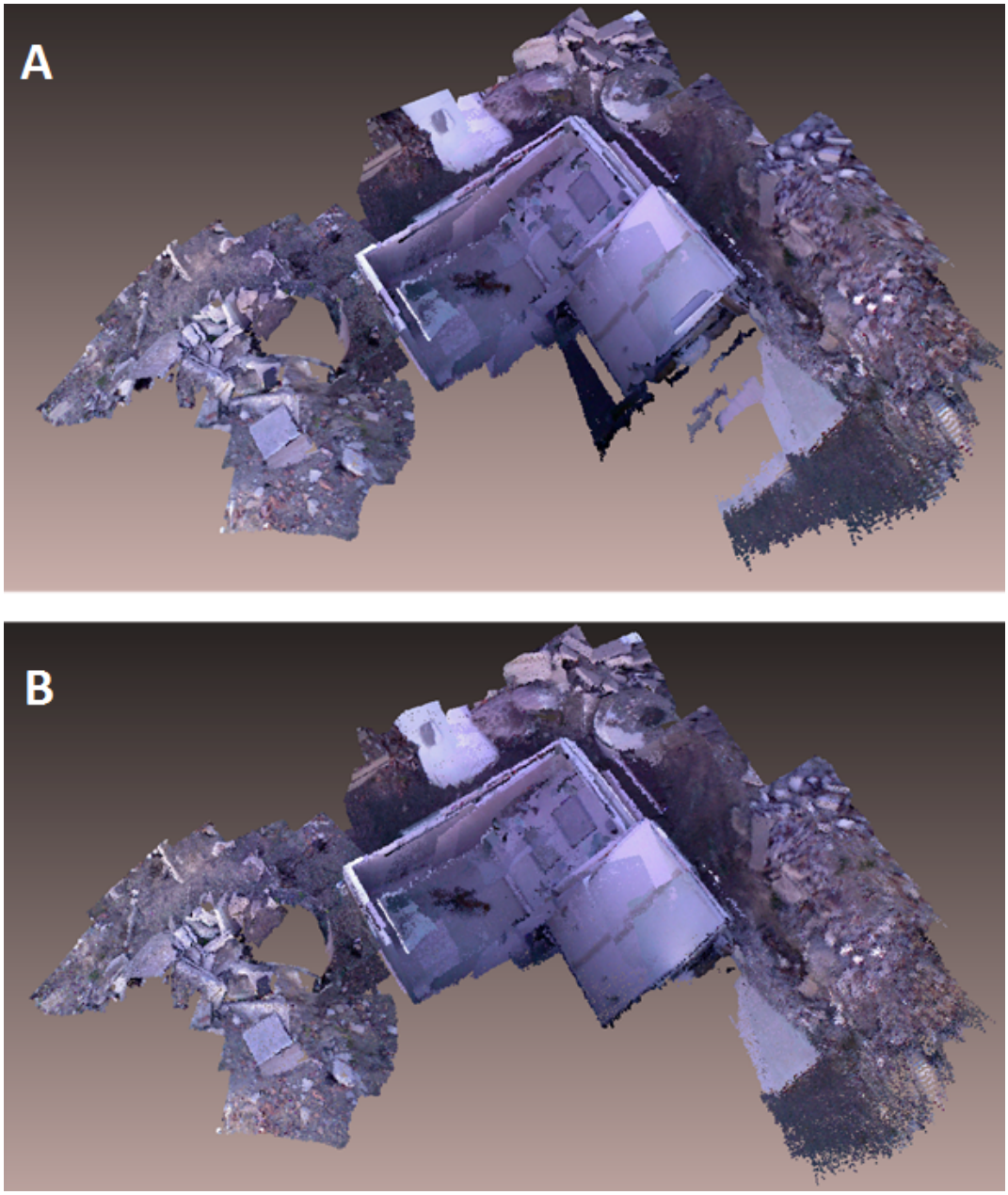


Figure 3.3: Filtered Bolton RRP. **A:** A raw point cloud consisting of 15,101,504 points. The model is so dense that it appears solid. **B:** The sampled model with manually deleted stray vertices and better distributed points. A total of 1,851,614 points remained resulting in an 87.74% reduction.

3.3.2 Vertex Normals

A vertex normal ensures proper viewing of a surface when rendered. The vertices from the filtered point cloud, with a user-defined "viewpoint", are used to calculate the normals within Meshlab using the "compute normals for point sets" function. A **viewpoint** is a point in space where the sensor was positioned during data collection. The viewpoint coordinates will be used to determine which order the cross product operation will be done using neighboring points.

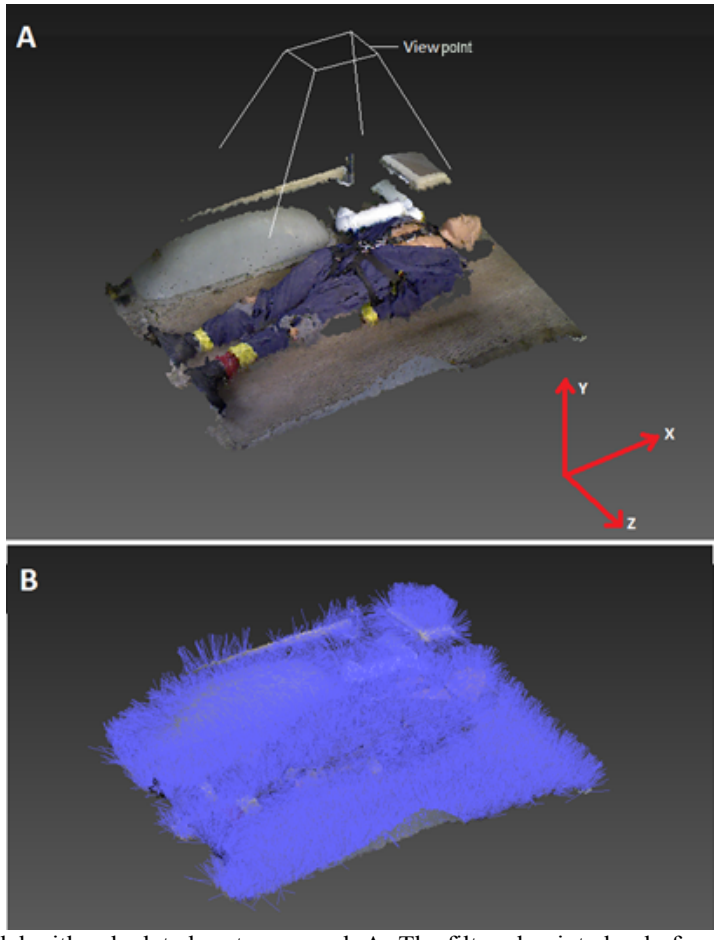


Figure 3.4: Body model with calculated vertex normal. **A:** The filtered point cloud of a mannequin body with a defined viewpoint above the data. **B:** The resulting point cloud with calculated normals facing the Y-axis (towards the viewpoint).

A sample point cloud is shown in Figure 3.4 with a defined viewpoint. If a normal is facing an incorrect direction, a constructed polygon may appear dark or unlit when rendered. The points can now be employed to efficiently reconstruct a solid surface.

3.3.3 Surface Reconstruction

After the point cloud processing phase in the model creation pipeline is completed the next phase is to create a 3D model. The surface reconstruction phase accomplishes this and is discussed below.

When constructing surfaces from a point cloud, the data should be as dense as possible, with little or (preferably) no “noise”. Irregularities found in the data, such as large distances between points, sharp angle changes of the surface, or lack of density, can produce undesirable results. The point cloud filtering step in Section 3.3.1 removes as much noise and redundancy as possible, but the data may still be chaotic because of artifacts introduced into the data because of the irregular surface features commonly found in collapsed building. To find the most suitable surface reconstruction algorithm for the simulation we performed ad hoc experiments using various available techniques resulting in the selection of the Ball Pivoting Algorithm (BPA) the most suitable [41]. BPA was selected because it is reasonably fast and typically produces a detailed model in several hours—conceivably within the time-frame needed to operationally deploy a TF to an incident. A comparison of the output from the algorithms is presented in Section 4.2.

The BPA rotates a sphere with a defined radius around each vertex, and if a collision occurs with another vertex, an edge is created between the vertices. The point cloud was filtered in Section 3.3.1 with the Poisson-Disk filter using a radial distance of 5cm. Therefore, we can assume that a slightly larger radius (+5mm) can be used by BPA without over-compensating for distances between vertices. An example point cloud before and after the BPA is shown in Figure 3.5.

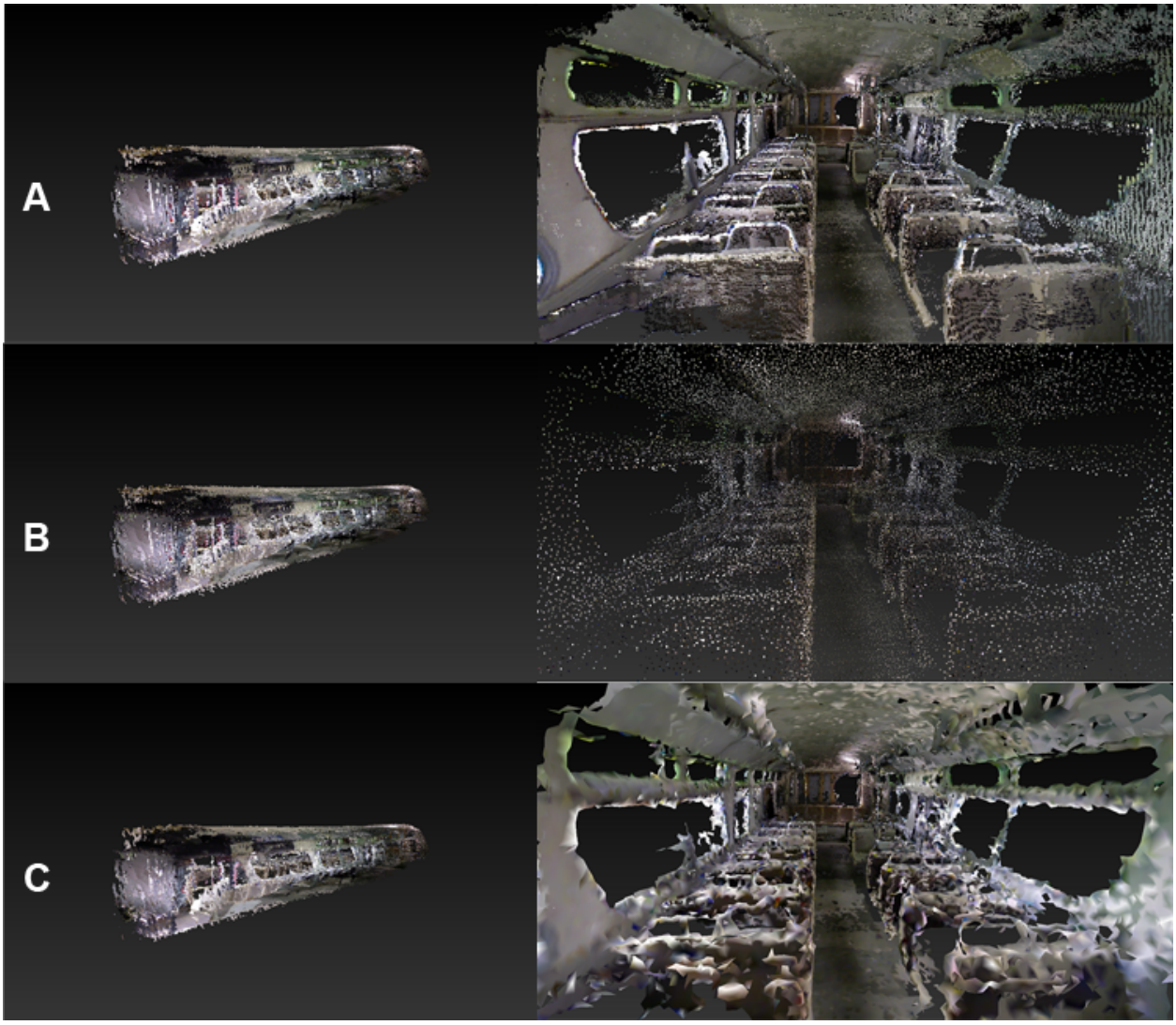


Figure 3.5: Two views of a passenger bus point cloud model being processed from the original data set to being surface reconstructed. **A:** The original passenger bus point cloud. **B:** The filtered point cloud. **C:** The 3D model produced from the BPA.

3.3.4 Decimation

To reduce the overall polygon count of the mesh, decimation is used within Meshlab. This allows for the removal of polygons in a mesh through automated classification and deletion

of vertices, preserving the original topology of the surface, and approximating a new surface from a resulting hole. As the reduction of polygons occurs, the geometric definition of the surface is reduced. Therefore, we must not decimate a high percentage of the original surface triangles. It was determined that a 50% reduction kept the definition that is desired and allowed the model to be inspected with high frame rates within the simulation. For example a decimated model is shown in Figure 3.6 with varying percentage decreases.

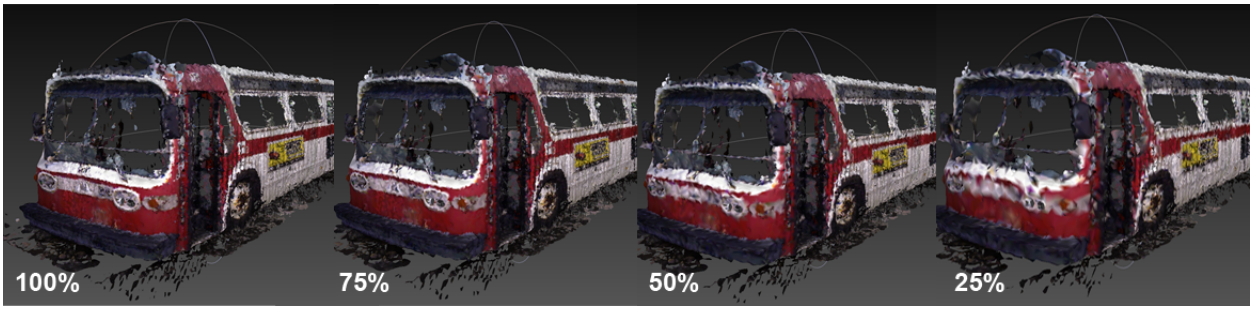


Figure 3.6: Decimation of a model at 100%, 75%, 50%, 25%.

3.3.5 Texture Creation

The last step in the model creation pipeline is to transfer the vertex colour data from the points to the mesh. Colour data brings a wealth of information to the mesh when being inspected. As seen in Figure 3.7, a texture can provide additional data to an observer of the model. Meshlab allows us to transfer this data by transferring the colours from the original point cloud data to the newly created 3D model. For each vertex of the 3D mesh, the closest vertex in relation to the original point cloud data is determined and the interpolated colour attributes copied into the 3D mesh vertex. The resulting texture is a JPEG with a user-defined resolution.

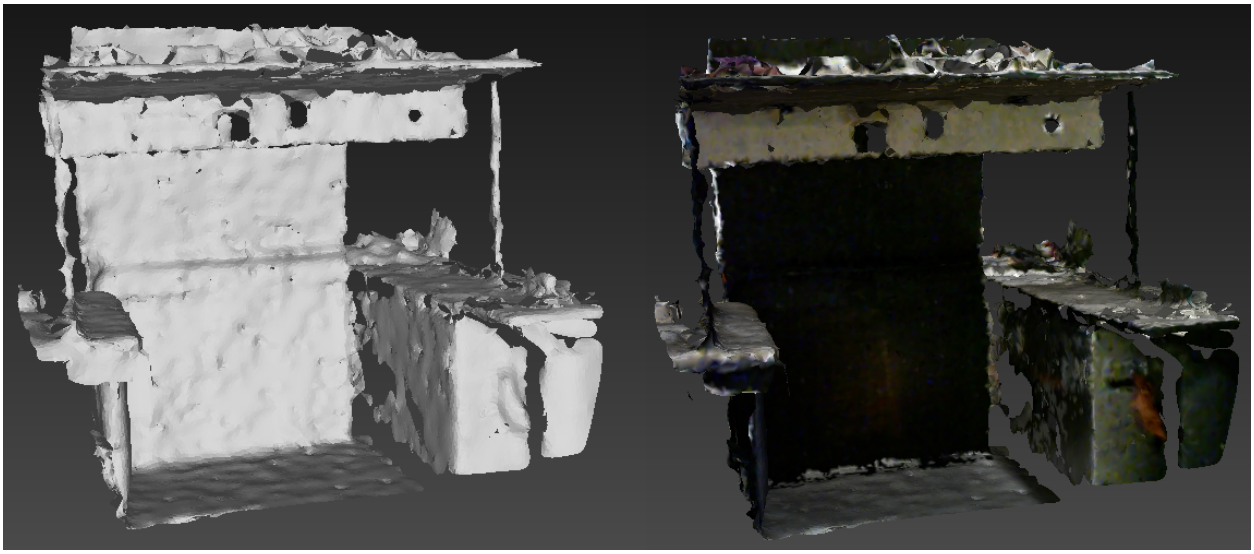


Figure 3.7: Before and after of model and model with texture

3.3.6 Model Output

The final model is exported as an OBJ (.obj file format). The format represents the geometry of a 3D model including the position of each vertex, each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices and texture vertices.

Sample screenshots of three point cloud models at each step of the model creation pipeline are shown in Figure 3.8.

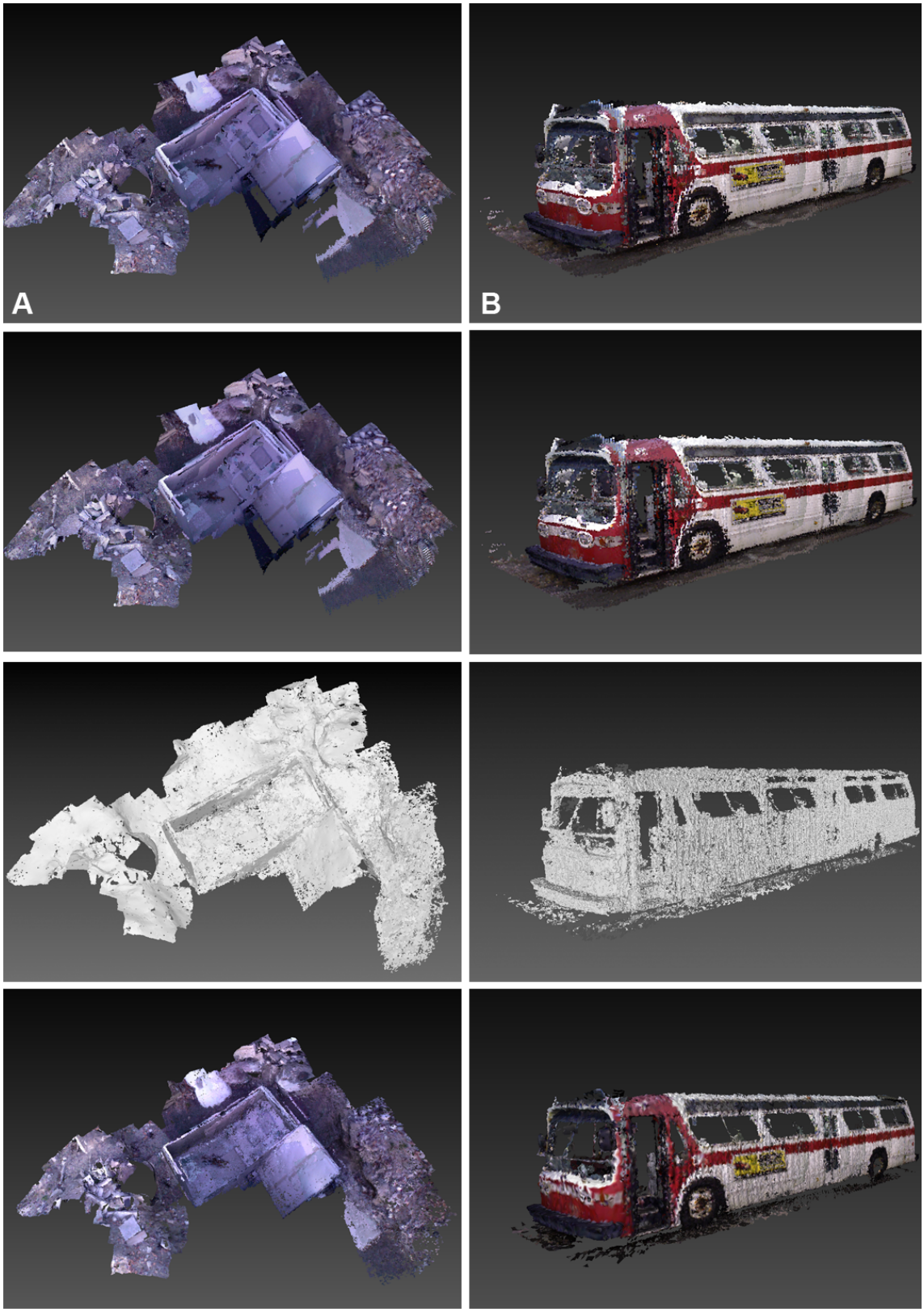


Figure 3.8: From top to bottom: Original point cloud data, filtered point cloud data, surface reconstructed data, decimated and textured 3D model ready for use. **A:** A collapsed building. **B:** A passenger bus.

3.4 Disaster Scene Reconstruction (DSR) Simulation

The Disaster Scene Reconstruction (DSR) simulation is created by using the models produced by the model creation pipeline and the Unity game engine. The goal of the system is to allow interaction with the modeled environments, in a realistic manner, in order to plan and safely perform tasks in the simulation before attempting to perform the task in a more dangerous real environment.

In this chapter, the Unity game engine is introduced as the chosen simulation platform. The architecture of the DSR system is discussed. In addition, the functionality of the DSR simulation pertaining to the support of USAR rescue operations is further explored.

3.4.1 Unity Development Engine

The Unity game engine is the development platform chosen to create the DSR simulation. The models from the model creation pipeline are compatible with, and can be imported, into the engine as an "asset". An **asset** is any component that is used within a "scene" such as 3D models, GUI graphics, scripts, particle systems, etc. A **scene** is an area within the engine that a developer can add assets into. The primary scripting languages used for the DSR simulation are JavaScript and C#. A sample scene with a created model asset is shown in Figure 3.9.

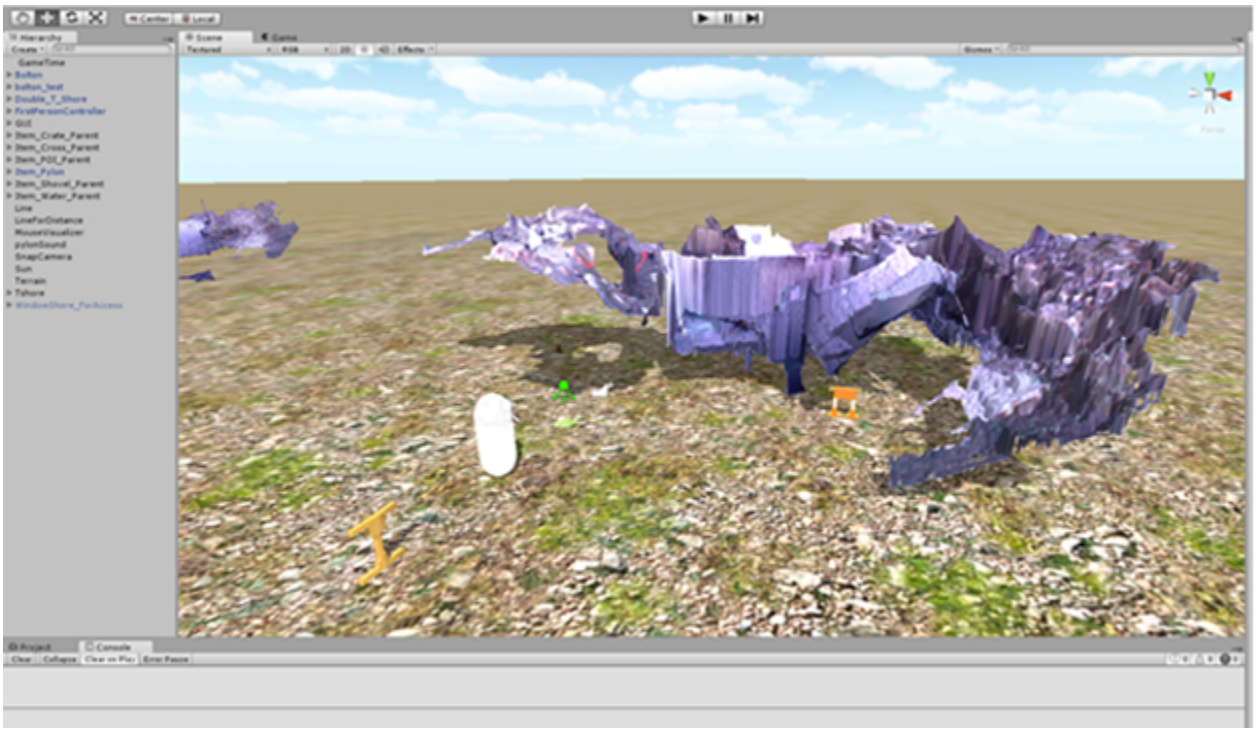


Figure 3.9: The unity game engine development environment including a created model (asset).

3.4.2 DSR Architecture

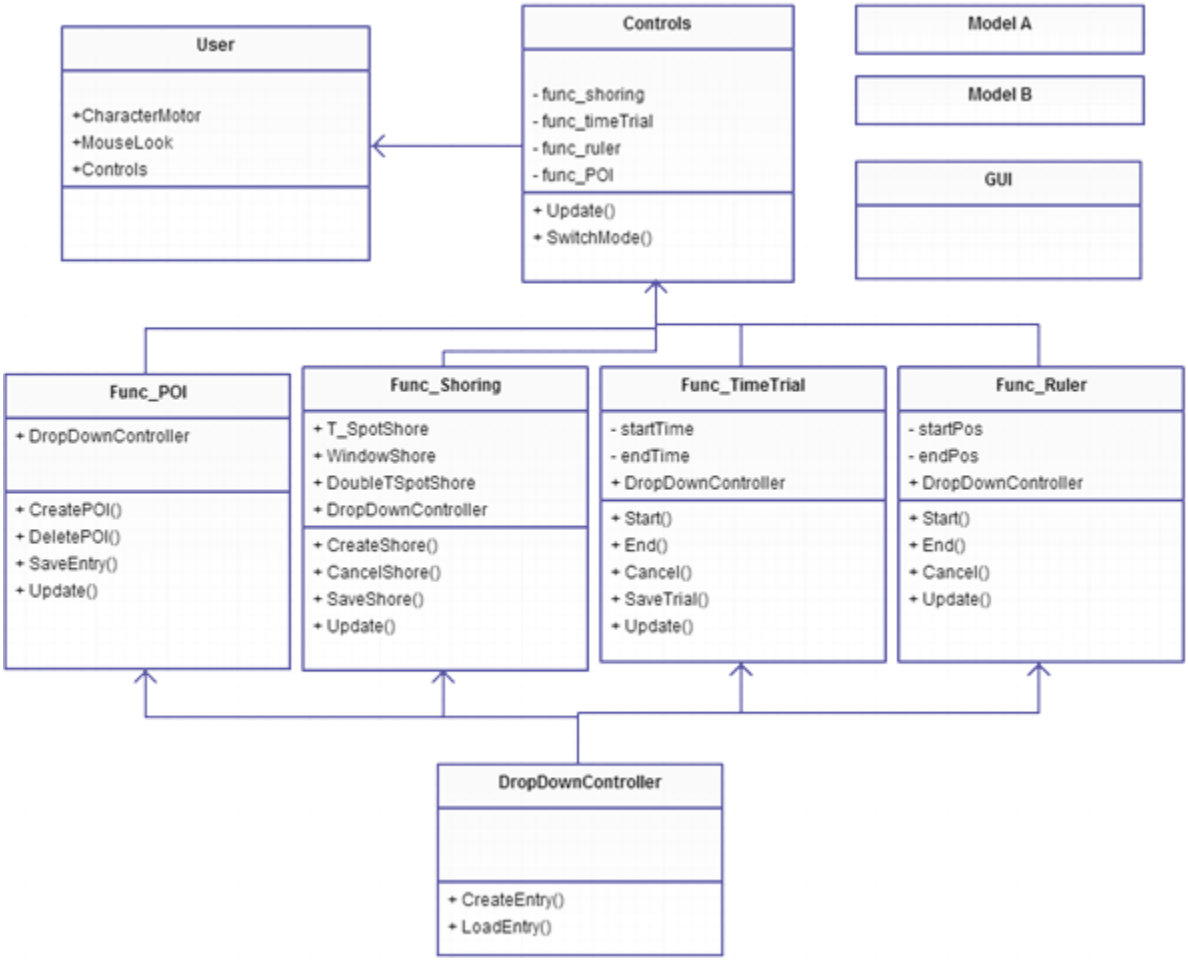


Figure 3.10: The DSR simulation application architecture.

The architecture of the DSR simulation system is shown in Figure 3.10. Each class contains functionality that can be added to a scene and toggled on and off. The Controls class is used to trigger functionality through key strokes and mouse use such as mode switching. The User class provides user movement controls. The CharacterMotor in the User class contains variables that are used to specify parameters associated with the physical characteristics and abilities of the simulated human. The parameters will affect the user in a physically-realistic manner during the simulation as the physics engine come into play. The user is a 3D model

controlled using the mouse and keyboard. Parameters provided by a user include the weight, height, and average walk and run speeds of the user.

Other classes contain functionality that is tailored to environment inspection and interaction in a USAR context. The class names represent the type of functionality they provide. These are: USAR_Shoring, USAR_Ruler, USAR_TimeTrial, USAR_POI.

The DSR_GUI class controls changes in the GUI during mode changes and to provide information.

3.4.3 DSR Functionality

The functionality provided in the DSR simulation system was based on the contents of the FEMA USAR field operations guide [20] and from discussions with USAR professionals. This guide provides a substantial listing of common USAR operations and techniques with descriptions. While the guide is not universally applicable to international USAR operations, it substantially covers operations in Canada and The USA. The focus during development was to provide a GUI to assist in using the system and add functionality that a trained USAR first responder could use to inspect, interact with and learn from an environment as they became familiar with the environment at the same time. We spoke to USAR professionals and based what modes were included in the DSR system on their input.

3.4.3.1 Interface

When the DSR system is first launched a user will see an open scene with 3D models. An overlaid set of icons are used to assist the user in using the system. What a user views when using the simulation is based on the camera's viewport and the overlaid GUI.

The interface is intentionally left uncluttered, avoiding users becoming overwhelmed with too many options. Mode specific controls are located at the top-right. The current mode of the system is displayed at the top. The modes are shown by icons that pertain to the included functionality at the bottom. Additional information pertaining to each mode is displayed at the bottom-left. When a user changes between modes, the GUI is changed to provide information pertaining only to a mode that is triggered, thus reducing unnecessary clutter. A sample of what a user would expect to see once the DSR system is run is shown in Figure 3.11. A skybox and ground texture was added purely added for aesthetic.

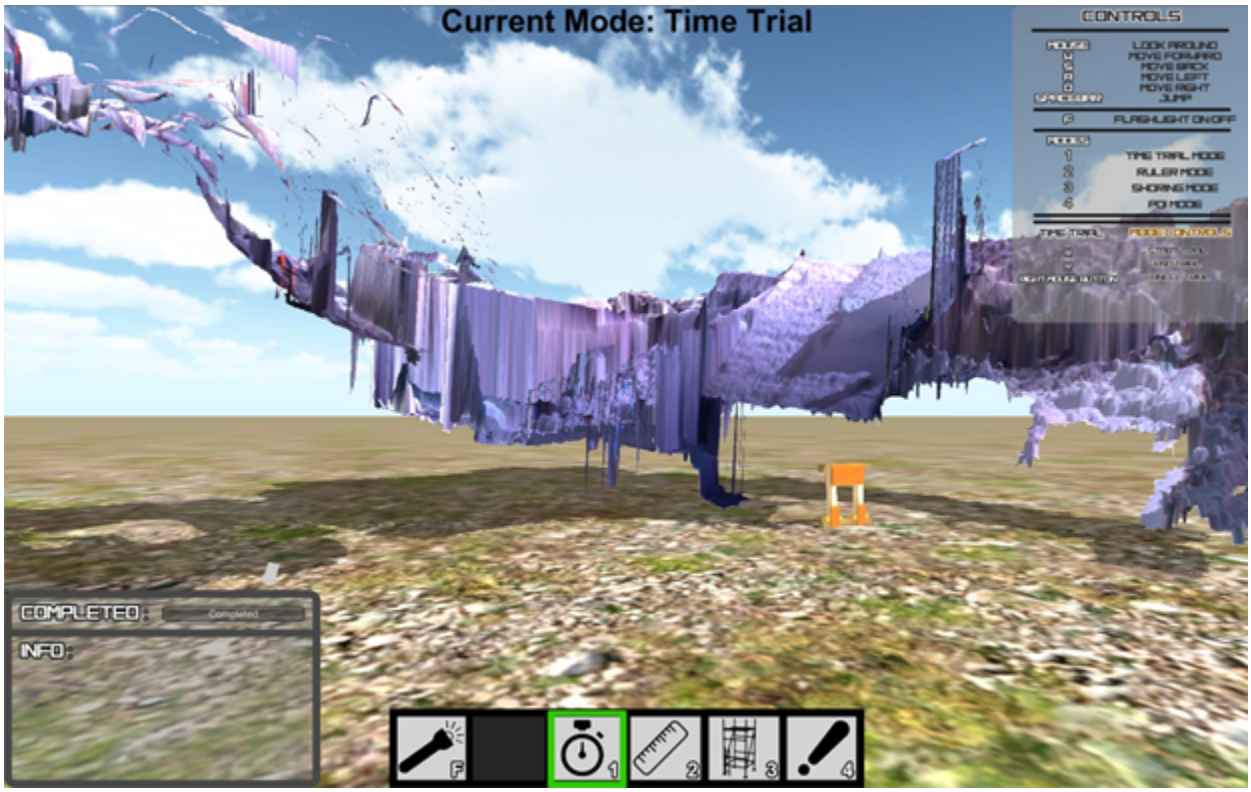


Figure 3.11: The initial DSR system GUI.

3.4.3.2 Controls

The main controls that are used to interact with the simulation and toggle modes are shown in Figure 3.12. The movement and jump controls are based on the common WASD and spacebar configuration for QWERTY keyboards.

The number keys activate “modes”. Modes typically contain a subset of commands and are further explained in their respective sections. The mouse is used to rotate the viewing perspective and sometimes used in other modes. Toggled modes provide additional controls and are only available when a mode is enabled.

MOVEMENT	
KEY	**ACTION**
Mouse	look around
W	move forward
A	move left
S	move backwards
D	move right
Spacebar	jump

UTILITY	
KEY	**ACTION**
F	flashlight ON/OFF

MODES	
KEY	**ACTION**
1	Time Trial Mode
2	Distance Measure Mode
3	Shoring Mode
4	Point of Interest Mode

Figure 3.12: The main control set for the DSR system.

Changing between the available modes is accomplished by adopting a common inventory access system found in many video games—using the number keys on a keyboard. When a key is pressed the selected icon in the toolbar is highlighted to inform a user what mode has been selected as shown in Figure 3.13. Mode specific controls are shown at the top-right of the screen. Information pertaining to the selected mode is shown at the bottom-left of the screen.

Figure 3.13: The GUI element used for mode switching within the DSR system. The Time Trial mode is selected.

A flashlight is a common tool used in operations to illuminate particular aspects of structures found in rubble. A virtual flashlight can be turned on and off by pressing the 'F' key. The flashlight icon at the bottom of the screen is highlighted if the flashlight is ON and is shown being used in Figure 3.14.

Figure 3.14: An example area within a scene before and after using the flashlight.

3.4.3.3 Time Trial Mode

Time Trial Mode		
Press "1" to Engage Mode		
Step	**Key**	**Action**
1	R	Start trial
2	R	End trial
-	Right Mouse Button	Cancel trial

Figure 3.15: Time trial mode controls.

As the data collected at the disaster scene is accurate, the models produced are also physically accurate providing the ability to measure physical space. Moving through space takes time and can be measured in the simulation through "Time Trial" mode. The controls available in Time Trial mode are shown in the Figure 3.15.

The Time Trial mode is enabled by first pressing '1' on a keyboard. Once enabled, a user can measure a time trial by pressing 'R' to start and once again to end the trial. If at any time the user wishes to cancel a trial being run, the right mouse button can be pressed. The path of the time trail is shown with a coloured line (using the Unity Line Renderer component) allowing the path taken to be reviewed later.

Determining traversal times on the rubble terrain can be used to assist in estimating the time to complete real-world tasks. The advantage of determining these estimates within a simulation allow avoidance of the real-world hazards. The information provided to a user once completing a trial are the distance traveled (in meters) and time taken. The path taken during the trail is displayed by a line and can be used to display hypothetical routes on the terrain.

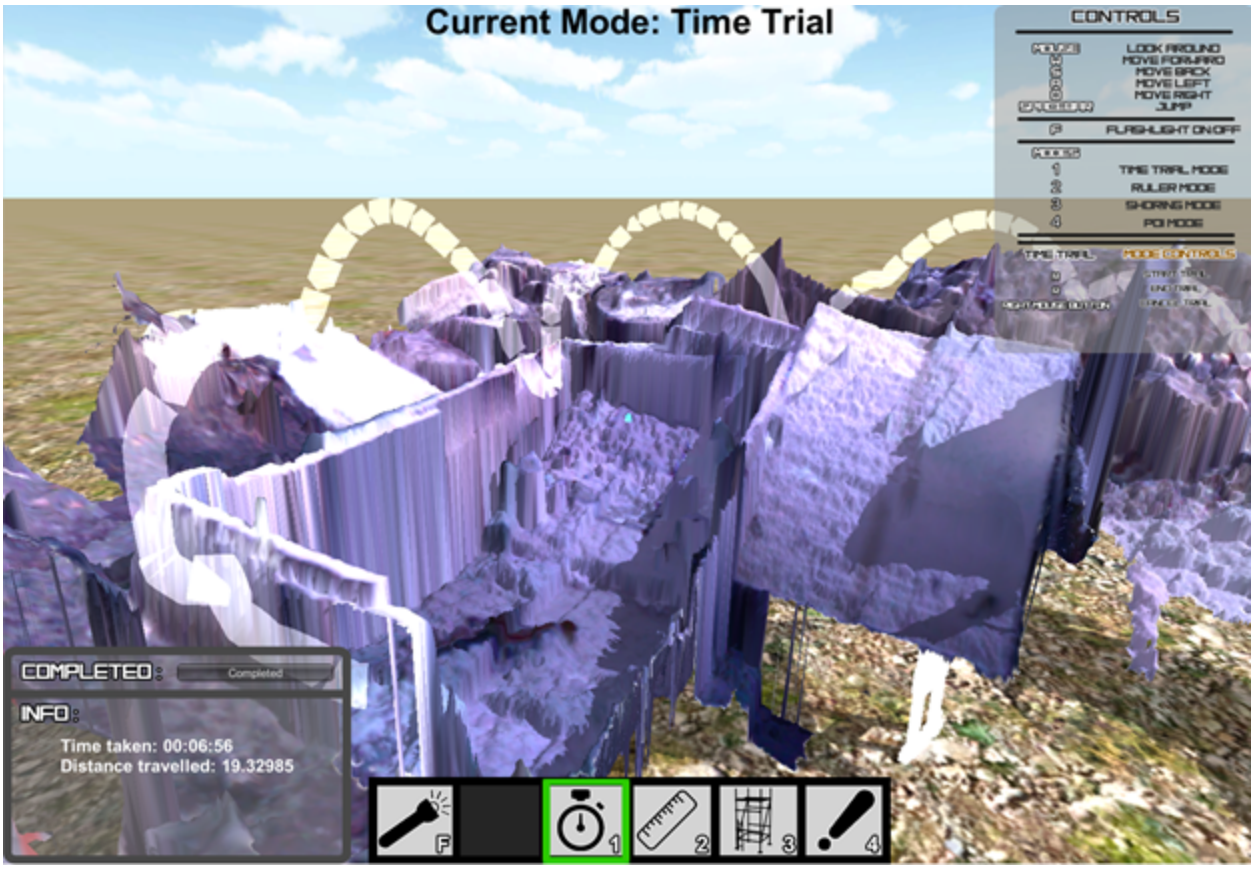


Figure 3.16: The path taken during a trial, the time taken to complete it, and the distance travelled during the simulation.

After a trial is completed the time taken and distance travelled are shown on the bottom-left of the screen as shown in Figure 3.16. The white line contains arcs from jumping during the trial. Past trials can be accessed by using a dropdown listing. A selected past trial will provide information about the trial and display the path taken.

3.4.3.4 Ruler Mode

Ruler Mode		
Press “2” to Engage Mode		
Step	**Key**	**Action**
1	R	Begin measurement
2	Left Mouse Button	Start measurement
3	Left Mouse Button	End measurement
-	Right Mouse Button	Cancel measurement

Figure 3.17: Ruler mode controls**.**

Rulers allow us to measure distance. As all distances between points in the simulation can be calculated, distances are commonly needed within operations to determine how long individual elements must be in order to construct structural supports like “shores”. The controls during ruler mode are shown in Figure 3.17. The ruler mode is enabled by first pressing ‘2’ on a keyboard. Once enabled, a user can begin a Euclidean distance measurement by pressing ‘R’. When the user presses the left mouse button a “ray” is shot in a straight line from the user’s perspective and if the ray intersects an object in 3D space the point is used as either the start or end coordinate for the measurement. A measurement can be cancelled at any time by pressing the right mouse button.

Measuring distances within the simulation allows learning about disaster terrain without posing risk to the inspector and can be used to determine whether an area is worth pursuing for entry or may requiring structural support in order to allow safe operation. Potential entry points can be measured to determine if they are fit for entry by dogs, robots, or rescuers before inspecting the area further. Other areas impractical to measure such as extreme undulations, dangerous terrain, and large distance can be measured.

Figure 3.18: The ruler being used to measure a modelled person.

Using the Ruler Mode on the rubble terrain is shown in Figure 3.18. A line is used to represent the measurement. The distance is reported at the bottom-left of the screen. Past measurements can be accessed from a dropdown list at the bottom-left of the screen. The selected measurement line is highlighted with pertinent information. In the future we wish to provide the ability to measure non-linear areas.

3.4.3.5 Shoring Mode

Shoring Mode		
Press "3" to Engage Mode		
Step	**Key**	**Action**
-	R	Cycle Through Shoring Configs.
1	Left Mouse Button	Place Shore
2	Mouse Scroll Wheel	Rotate Shore
3	Left Mouse Button	Initiate Growing Step
-	Right Mouse Button	Cancel Shoring

Figure 3.19: Shoring mode controls.

A building collapse may produce many areas needing structural support to make them safe for first responders to search them. USAR TFs will construct support mechanisms called "shores" to temporarily support areas so that operations can safely continue. While there are many mechanisms from which shores can be constructed, for the most part permanent shores are constructed from lumber and held together with nails. To a certain extent, creating shores is a resource allocation problem as lumber is usually only available in limited supply at a disaster site. To efficiently allocate resources for each shore being built, a shoring "calculator" function was added to the simulation to estimate the resources needed for different configuration of shore. Another potential benefit of such a calculator, coupled with an accurate model, is the ability to allocated resources for shore construction before a TF actually arrives at the incident—thus saving time as the rough measurements for a shore may already have occurred while the TF is inbound allowing construction to occur shortly after they arrive.

The controls during Shoring mode are displayed in Figure 3.19. Shoring mode is enabled by pressing ‘3’ on a keyboard. The ‘R’ keystroke is used to cycle through the three available shoring configurations (T-Spot, Double T-Spot, and Window/Door). While there are many different types of shores, three were selected to demonstrate the feasibility of creating virtual shores. Other shores are beyond the scope of this work.

The chosen shore will be instantiated and its position will follow where the user moves the cursor. When the user presses the left mouse button, a “ray” is projected in a straight line from the user’s perspective and, if the ray intersects an object in 3D space, the shore is placed at those coordinates. This action initiates a series of steps to create a shore.

The first construction step uses the mouse scroll-wheel to rotate the shore’s Y-axis. After the shore is oriented as desired, the left mouse button is pressed to initiate the “growing phase”. During the growing phase the shore’s height increases within the simulation and halts when a section of the shore model intersects another model. The intersection determines the distance between the ground plane and the unstable area being supported. The distance measurement is then used to calculate the required materials to build a shore. The lumber types and measurements are shown at the bottom-left of the screen. If a shore surpasses a max height, the model will be colored red to indicate an ineligible shore—meaning that this shore could not be constructed in reality. At any time during a shore’s construction the right mouse button can be pressed to cancel the process.

The shores are assumed to be constructed out of wood and include the following types of lumber selection: 6” x 6”, 4” x 4”, 2” x 4”, and ¾” plywood. The lumber is simulated within

DSR as shown in Figure 3.20 and is used to assemble the “T” Spot Shore, Double “T” Shore, and Window/Door Shore.

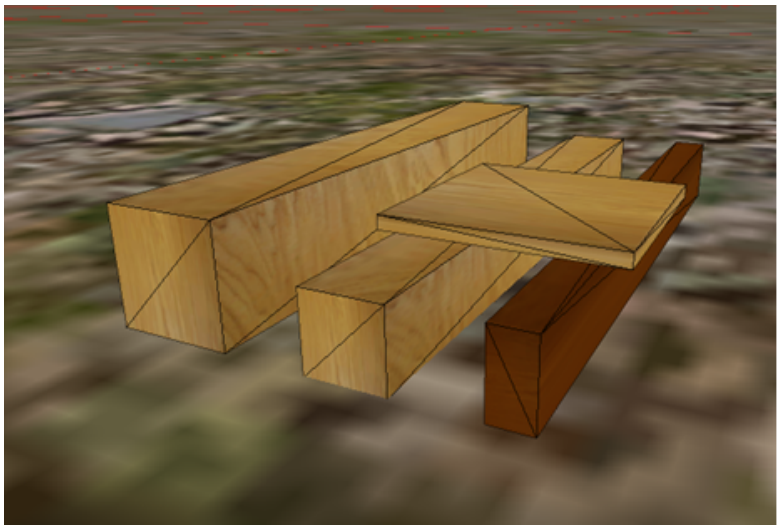


Figure 3.20: The three four lumber types used within DSR. From left to right, a 6” x 6”, 4” x 4”, 2” x 4”, and a 12” x 12” piece of ¾” plywood on top.

Each shore is comprised of common pieces. The **header** supports the weight at the top of the shore. The **sole plate** distributes the weight at the bottom of the shore. A **post** supports and funnels the weight from the header to the sole plate. **Wedges** (shims) are used to tighten the fit of the post between the header and sole plate. **Plywood Gusset Plates** are used to hold multiple pieces in place with nails. Lastly, a **cleat** is used to act as a support or prevent slippage.

The available shores constructed from simulated lumber are shown in Figure 3.21 to Figure 3.23. Each shore’s respective specifications can be found within the appropriate FEMA Field Operations Guide [20] and are shown beside each simulated shore.

The “T” Spot Shore is a vertical support as shown in Figure 3.21. It is used as a temporary shore to support weights of up to 4,000 pounds. A single post is used to distribute the

weight between a header and sole plate based on the Double-Funnel Principle (see Figure 2.3). Two plywood gusset plates are used to stabilize the header and post. A cleat is used to stabilize the sole plate, post and wedge.

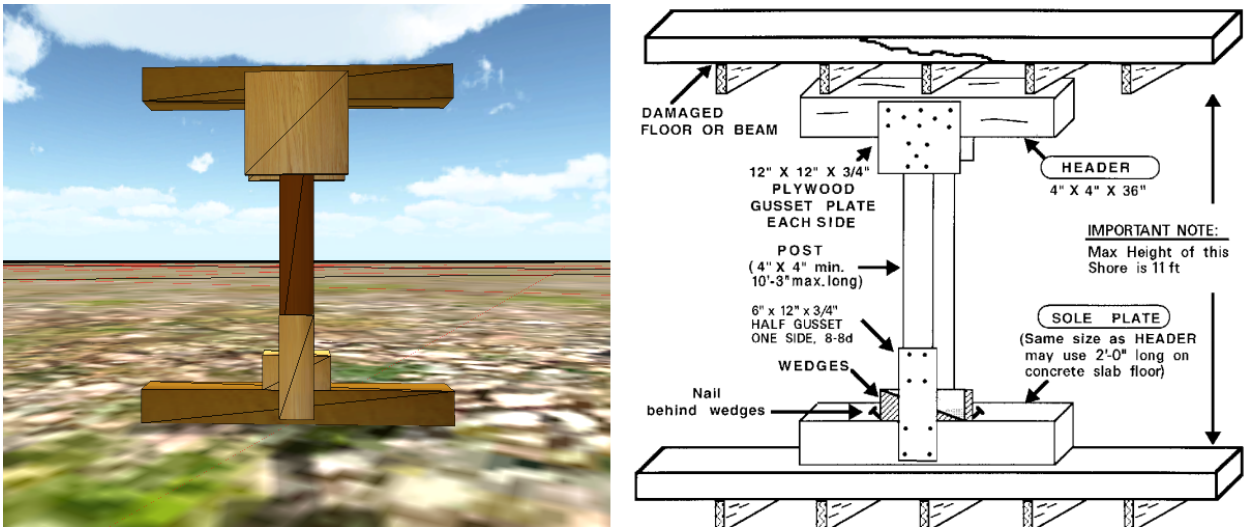


Figure 3.21: On the left, the simulated "T" Spot Shore within the DSR simulation. On the right, the specifications of the shore found within the FEMA Field Operations Guide.

The Double "T" Spot Shore is a vertical support as shown in Figure 3.22. More stable than a "T" Spot Shore, it is used to support weights of up to 16,000 pounds. Two posts are used to distribute the weight between the header and sole plate. Two plywood gusset plates are used to stabilize the sole and posts. A cleat is used to stabilize each post to the sole plate. If the area needing support is larger than six feet in height, an additional two plywood gusset plates are used for stabilization in the middle of the posts. When the shore is in its growing phase, the model will change to accommodate the extra gusset plates for heights greater than six feet.

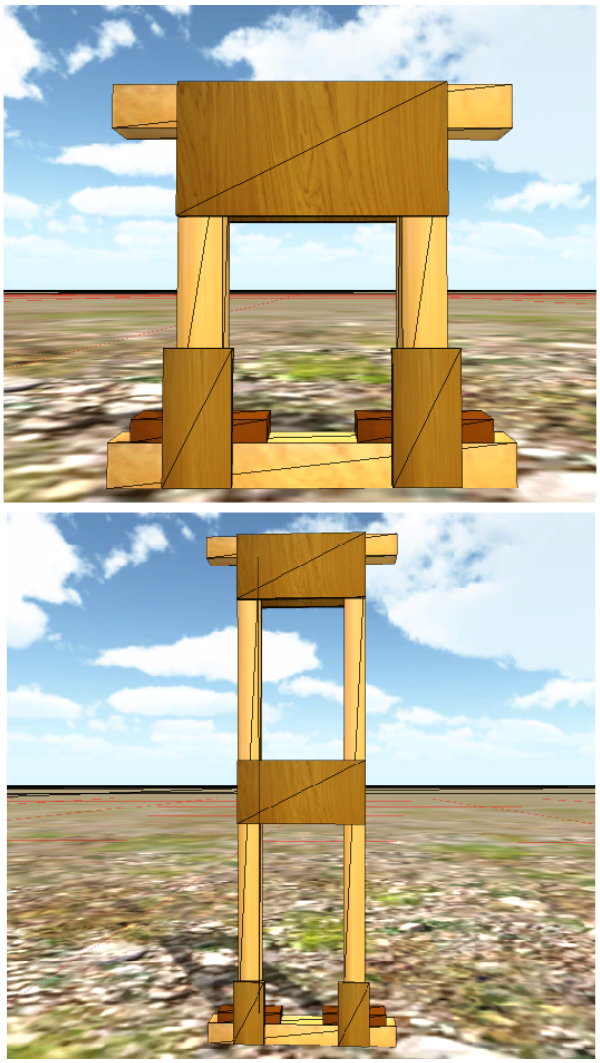


Figure 3.22: Top-left: The simulated Double "T" Spot shore configuration below 6 feet. **Bottom-left:** The simulated shore with a height greater than 6 feet. **Right:** The specifications of the shore found within the FEMA Field Operations Guide.

The Window and Door Shore is a vertical support as shown in Figure 3.23. It is used to either support a compromised door or window opening and can be used to allow entry into an opening safely. Two posts are used to distribute the weight between the header and sole plate. The header piece requires 1" of thickness for every foot of horizontal opening (example: 3' opening = min. 4" x 4" header). Wedges are used to tighten the fit between the edge of the door/window frame and between the posts, sole plate and header. Cleats are used for stabilization

at the base of the shore. The diagonal braces shown in Figure 3.23 are omitted as the shore created within the simulation is intended for entry. When the shore is in its growing phase and exceeds a horizontal length of 4', the header will be changed to a 6" x 6" lumber piece. If the vertical maximum height is exceeded, only the posts will be highlighted in red. If the horizontal maximum width is exceed, only the sole plate and header will be highlighted in red.

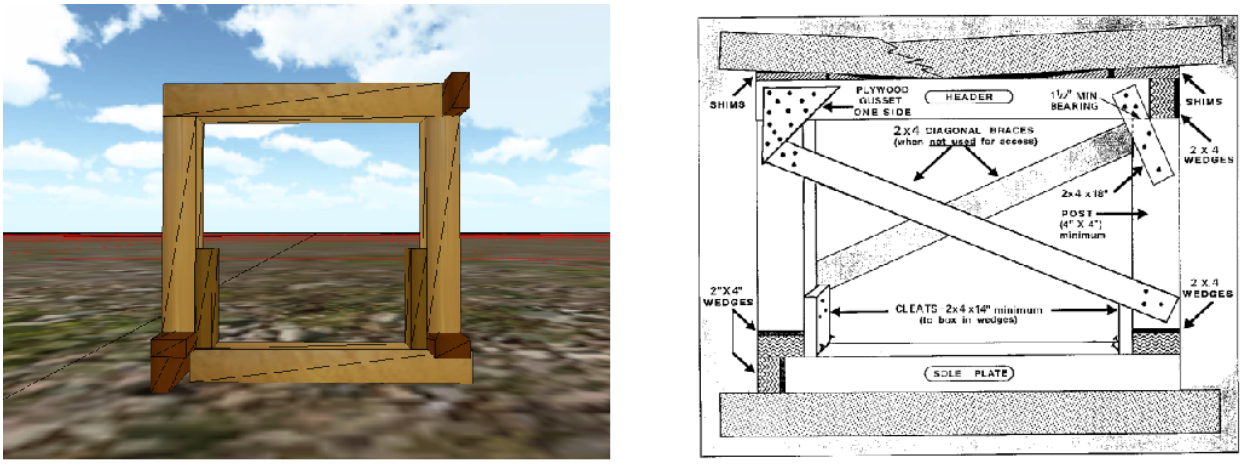


Figure 3.23: Left: The simulated Window/Door shore. **Right:** The specifications of the shore found within the FEMA Field Operations Guide.

A Double “T” Spot shore exceeding the max height of 12’ is shown in Figure 3.24. The shore changes into a red color indicating an ineligible support distance.

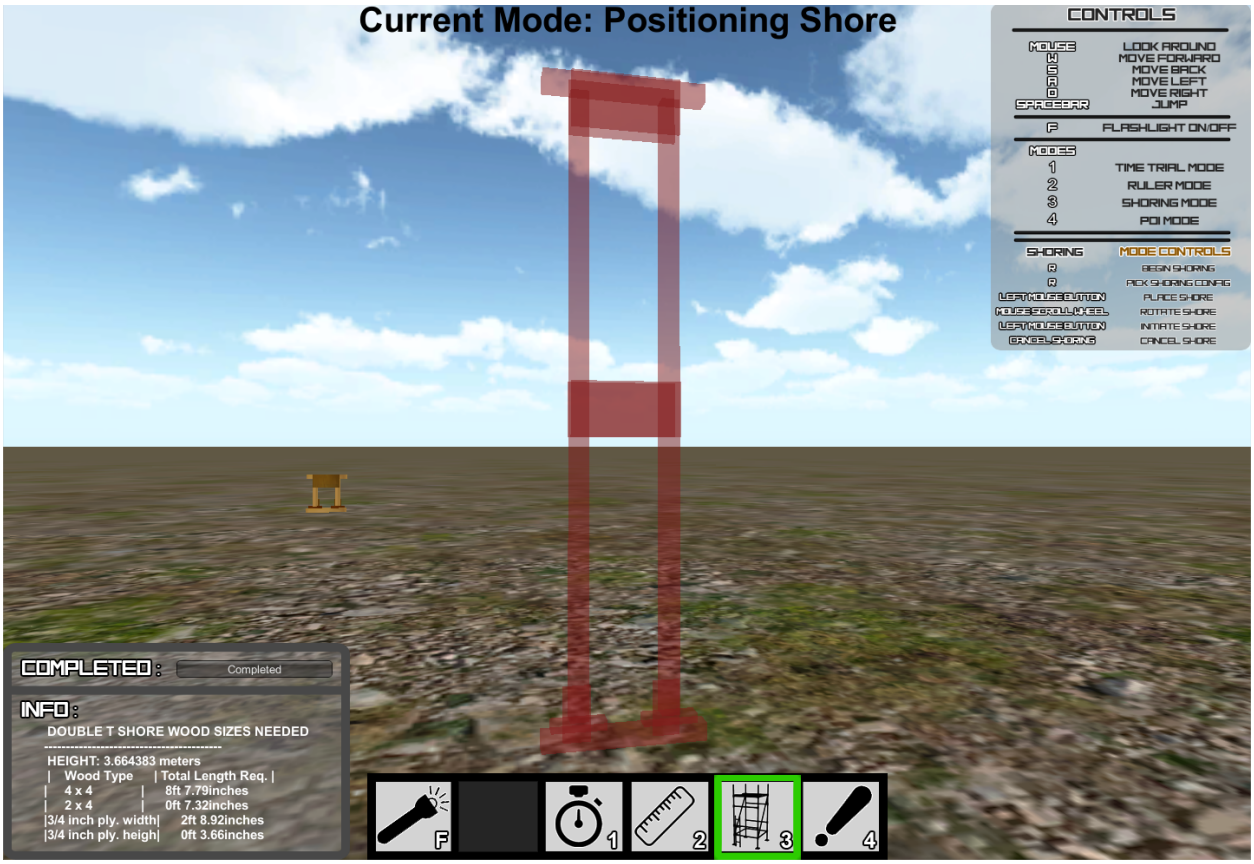


Figure 3.24: A Double “T” Spot shore created within the DSR system. The height is 3.6643 meters and exceeds 12’ (the max height). The model has turned red to indicate an ineligible shore.

An example “T” Spot shore within the simulation is shown in Figure 3.25. The distances and materials are reported at the bottom-left of the screen. Past shore calculations can be accessed from a dropdown list at the bottom-left of the screen.

Figure 3.25: A “T” Spot shore created within the DSR system to support a structure. The height is 1.264401 meters and the wood needed to construct it are shown at the bottom-left of the screen.

3.4.3.6 Point of Interest Mode

Point of Interest (POI) Mode		
Press "4" to Engage Mode		
Step	**Key**	**Action**
-	R	Cycle through POIs
1	Left Mouse Button	Create POI
-	Right Mouse Button	Delete POI

Figure 3.26: Shoring mode controls.

As an additional feature for representing points of interest in the simulated environment, six 3D models were created to be used during Point of Interest (POI) mode. Various 3D models are used to visually represent areas where activities are taking place. The premise is to allow information sharing between concurrent users within the simulation. This functionality is present within the current system but will be applicable in future work when networking is added to the simulation.

Different user types such as medical personnel, USAR workers, Structural engineers (SS), Excavation, etc., use POI models within the DSR simulation to represent a type of activity and share the event with other concurrent users. For example, a Medical user would place a POI within the simulation for any medical event happening in the real-world environment. A Structural Engineer user would place a POI for any area inspected and that was determined to need shoring. This information sharing between users allows for situational awareness of the modeled disaster areas with people en-route to or working elsewhere at a disaster site. En-route users would see events occurring within the simulation and become aware of how their planned work will fit into the entire operation.

The POI mode is enabled by pressing ‘4’ on a keyboard. Once enabled, a user can cycle through the available models by pressing ‘R’. When the user presses the left mouse button a “ray” is projected in a straight line from the user’s perspective and if the ray intersects an object in 3D space, a POI is placed at the coordinate. The right mouse button can be pressed to delete POI models with the mouse cursor as long as the user is within a close enough distance (2 meters.)

The current models available are shown in Figure 3.27. These are;

- Pylon: used to represent a number of things including perimeters
- Red Cross: medical attention is required or is being conducted
- Exclamation point: an important aspect of the model was noticed
- Water faucet: a water hazard is in the area
- Box: support is needed
- Shovel: excavation is required or is being conducted

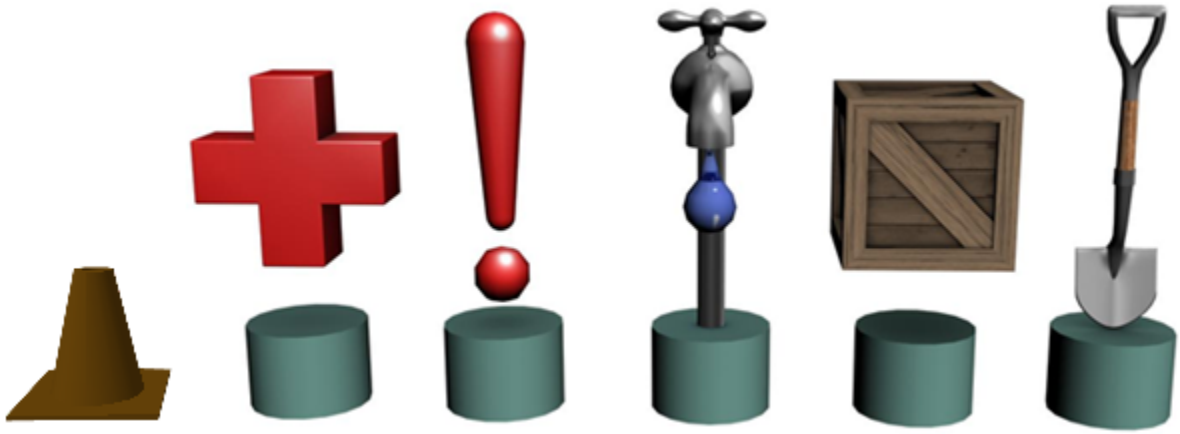


Figure 3.27: The 6 3D models a user can place onto the model to signify points of interest.

An example Red Cross POI and pylons used within the simulation to signify medical attention is required is shown in Figure 3.28. The model being inspected shows a person lying on the ground.

Figure 3.28: A Red Cross model and pylons placed beside a body.

CHAPTER 4 EXPERIMENTAL RESULTS

In this chapter, we present the results of experiments that compare different 3D modeling techniques using disaster terrain. In addition, we demonstrate the performance of the DSR system in accurately representing certain physical phenomena modeled from the real world.

The test environments used to collect point cloud data are presented in Section 4.1. The different techniques used during the 3D model creation process are compared in Section 4.2. A series of trials measuring time-for-traversal (TFT) accuracy within DSR are presented in Section 4.3. Finally, example shores are created within the simulated environment and compared to real-world measurements in Section 4.4. These experiments are intended to provide evidence that the DSR system can be used to support real-world decision based on the simulated environments it can represent. We are not making the claim that DSR can be used in all circumstances however, our analysis seeks to support the assertion that DSR can support additional functionality in the future based on our observations concerning the performance of its existing tool set.

4.1 Testing Environments and Data

Acquiring disaster terrain data is a unique opportunity since accessing a site where an incident has occurred is understandably problematic. Point cloud data concerning physically-accurate and simulated urban disaster environments was attained from the OPP Reference Rubble Pile (RRP) in Bolton, Ontario. Several expanses of the rubble pile were sensed and recorded using the technique described in [5] with a RGB-Depth sensor employed by a UAV and by hand scanning[3]. A typical scan can range from a couple seconds to many minutes. The RGB-D sensor is a Microsoft Kinect that records at a resolution of 640 x 480 at 30 frames-per-second.

[3] Manually holding the sensor and scanning an area where flight was either precluded or as an expedient.

The Infrared (IR) depth sensor has a detection range of 0.7-6 meters (2.3-19.7 feet). The hardware used for the experiments is a PC with an Intel i7 quad-core CPU, a GeForce GTX 680M graphics card and16 GB RAM. It is assumed that similar hardware would be available to first responders in the field.

4.1.1 OPP RRP Environment

Figure 4.1: The Ontario Provincial Police (OPP) Reference Rubble Pile training facility at Bolton, Ontario. A building with floors collapsing onto each other (a "pancake" collapse) and a bus contained within the rubble are shown in this partial view of an expanse that extends about an acre.

Figure 4.1 reveals features within the rubble that would be of interest to decision makers within a USAR TF. The partially collapsed building and passenger bus have a higher probability of containing live victims than other areas and would have a higher search priority than other areas where finding victims would be less likely. The structure to the left is a simulated "pancake" collapse where one floor has collapsed onto another. A bus is present to the right that is contained within the rubble at a steep angle. These areas are surrounded by potentially

dangerous terrain comprising of hazards such as sharp glass, rebar, sudden drops, and an unknown structural load capacity. To circumvent these dangers we flew a UAV above the terrain while capturing 3D image data with the RGB-D sensor.

4.1.2 N-CART Lab Environment

Figure 4.2: The N-CART research lab at Ryerson University.

The N-CART research lab (Figure 4.2) was used to create artifacts that would typically require some form of structural support intervention to support the safety of search teams working in an actual structural collapse. These micro-environments allowed us to gather data for the shoring experiments in Section 4.4. Figure 4.3 displays a few desks assembled to represent an area that might be selected to receive a vertical shore and a doorway that may be assessed to require a window/door shore to allow searchers to enter a an enclosed space. Both areas were scanned by hand.

Figure 4.3: Two areas used for modeling within the N-CART research lab. **Left:** Three desks used to simulate a roof of a building requiring a vertical shore. **Right:** The door of the N-CART lab that will be used for a window/door shore for entry.

4.1.3 Point Cloud Data Sets

The point clouds from the test environments are displayed in the following images. These point clouds were used as input into the 3D Model Creation Pipeline and later inspected within the DSR simulation system. Each image contains the real-world environment, the amount of points produced in the point cloud, and two perspectives of data.

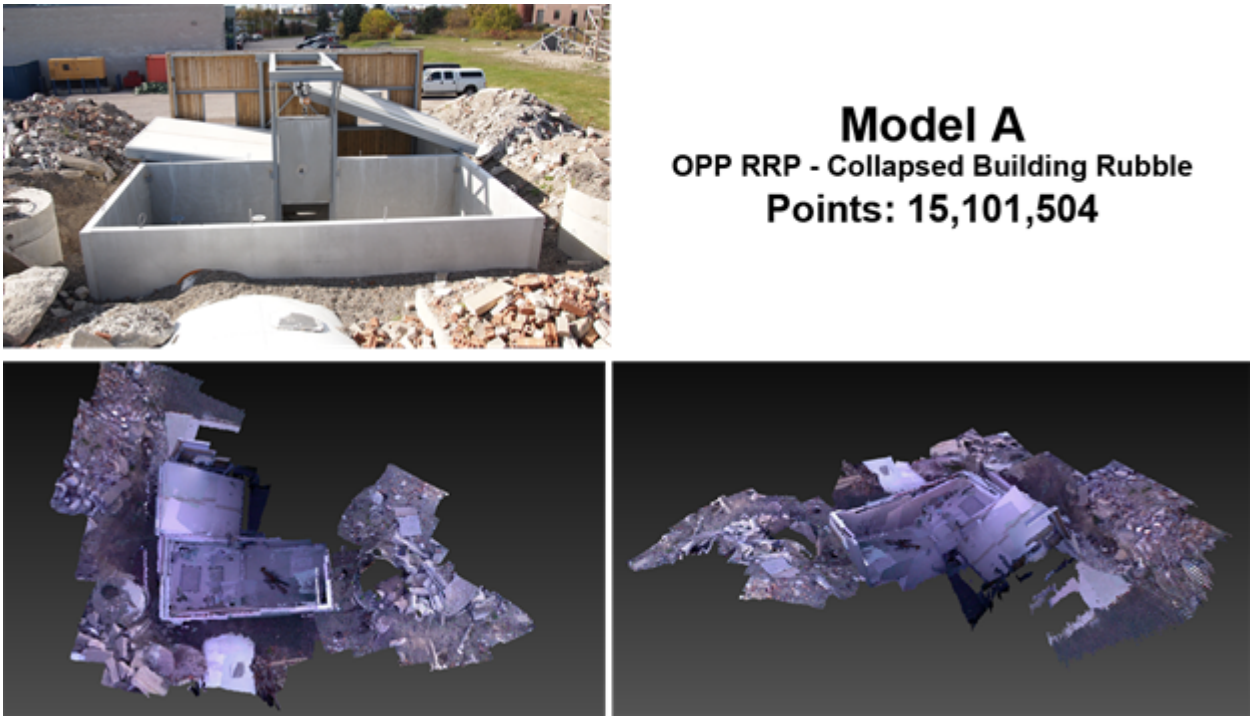


Figure 4.4: The OPP RRP collapsed building environment and point cloud model output. The environment was scanned using the UAV and consisted of a sensor-equipped UAV and the data was gathered in a 4 minute flight.

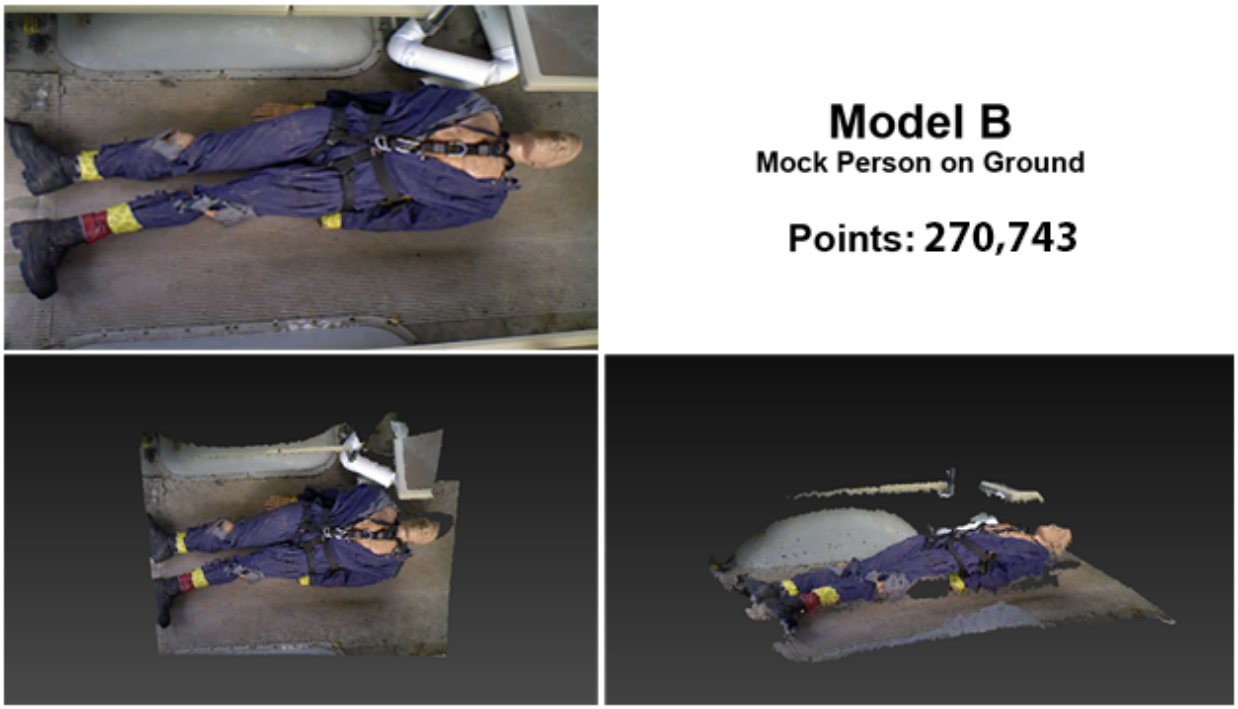


Figure 4.5: A simulated victim scanned by hand.

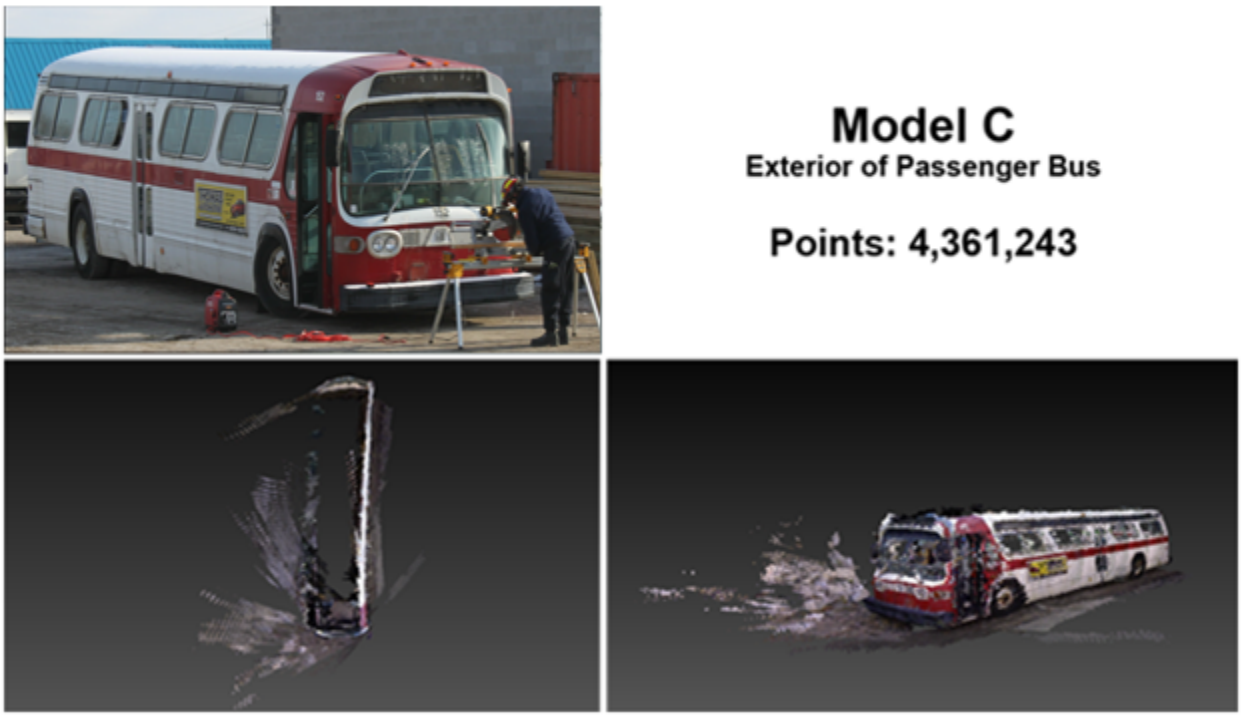


Figure 4.6: The exterior of a passenger bus scanned by hand.

Figure 4.7: The interior of a passenger bus which was scanned by hand.

Figure 4.8: A doorway at the N-CART research lab. Scanned by hand.

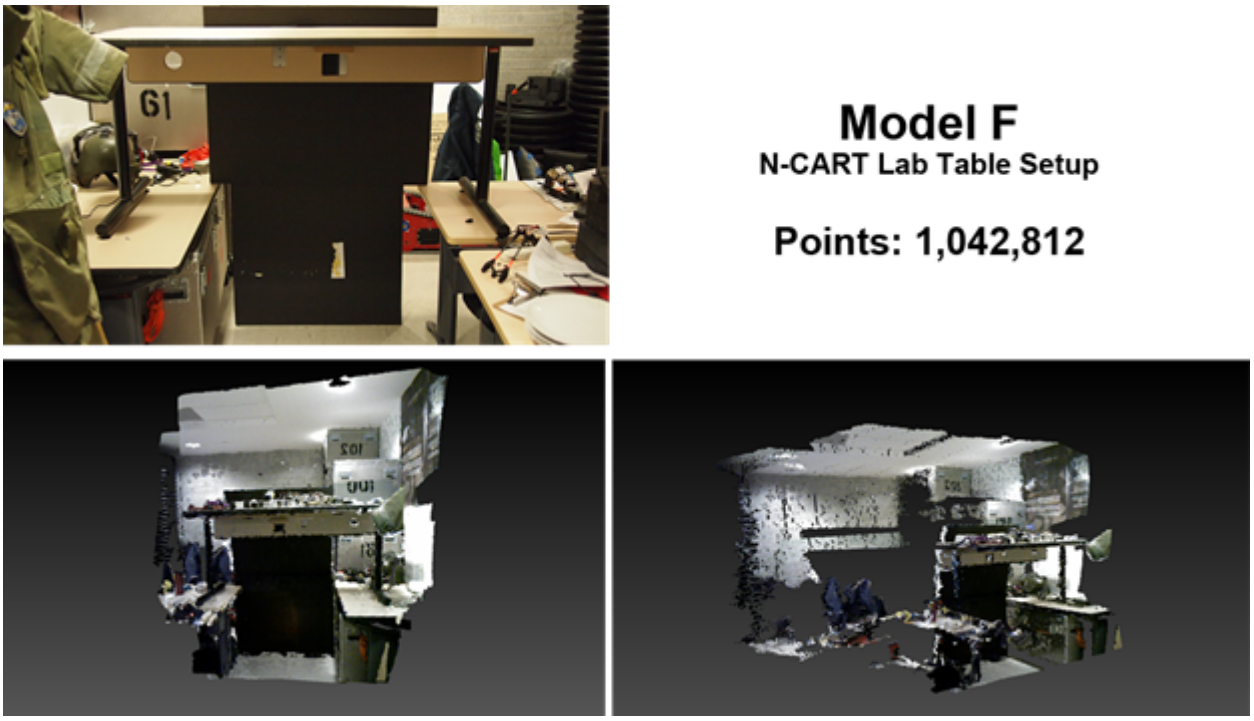


Figure 4.9: Three tables stacked at the N-CART research lab. Scanned by hand.

4.2 3D Modeling Experiments

The point clouds are provided as input into the Model Creation Pipeline. The data contains details of the environment such as voids and sharp surface changes—features typically required during the inspection of an urban disaster before searching for survivors can begin.

Depending on the length of time taken to conduct the scan the actual data could contain millions of vertices. It was unclear as to which techniques would be most effective at creating a 3D model from this type of terrain. The rendering time of each model within the DSR simulation is directly related to the vertex and polygon counts and can be a significant source of computational delay. Therefore these counts should be reduced as much as possible.

We experimented with different parameters during the first step of point cloud processing phase—filtering—to determine what distance parameter is most effective at reducing the vertex count without oversimplifying the implied surface. We experimented with 3 surface reconstruction techniques to determine which approach was best at utilizing the points within the set and creating a surface that preserves the geometric properties of the terrain. The experiments were conducted in the phase where polygons are added to the data set—the first step of the surface reconstruction phase.

After each point cloud is produced from the registration technique [5] it is manually inspected and "cleaned" by selecting and deleting stray points within Meshlab to ensure that no unknown artifacts or inaccuracies have been introduced. After a model is scrubbed, it is run through the pipeline. Occasionally, during the data collection, problems may be detected that require additional scans of the environment to correct. It was observed that quick movements would sometimes produce an inaccurate point cloud due to the motion blur in RGB video from

the camera sensor. In other cases the infrared (IR) light from the sun would wash the readings from the infrared sensor resulting in undesirable voids in the point cloud model. It was later discovered that a steady movement of the sensor during dawn or dusk—to reduce IR interference—produced the best point clouds. It is assumed that using a better sensor would improve the data collected. However, this is simply conjecture and beyond the scope of this work.

4.2.1 Filtering

This experiment has been included as a mechanism for determining a filter distance parameter that is effective in reducing the overall vertex count for data collected in an urban disaster environment. A large filter distance could remove too many vertices. In some cases a point cloud model may contain clusters of densely positioned points. These vertices are redundant if they are close to one another or overlapping[4]. They do not add any detail to the topology of the model but, cumulatively, add to processing time. Therefore these redundant points should be removed.

We experimented with a series of parameters using the Poisson-Disk Filter [45] to remove redundant vertices and evenly distribute the data. The filter uses an absolute distance parameter and randomly selects points within the data set—deleting adjacent points within a defined spherical radius. There is an implied provision that the radius must not be too great lest it oversimplify the geometric details of the data.

Each point model was run through the Poisson-Disk filter with varying absolute distance parameter values (in millimeters) consisting of 2.5, 5, 7.5, 10, 15, and 20 mm. Two sample point

[4] Caused by multiple overlapping scans with the collection sensor.

cloud models (before and after each distance parameter) were applied and are shown in Figures 4.11 and 4.12.

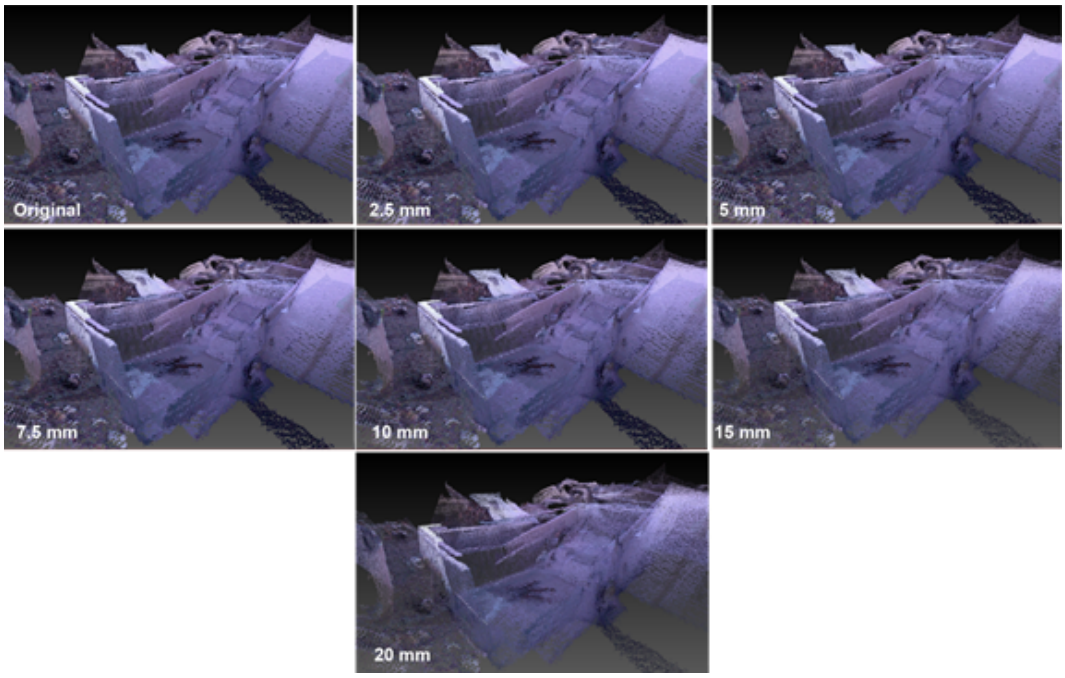


Figure 4.10: Bolton rubble model after each distance parameter is applied.

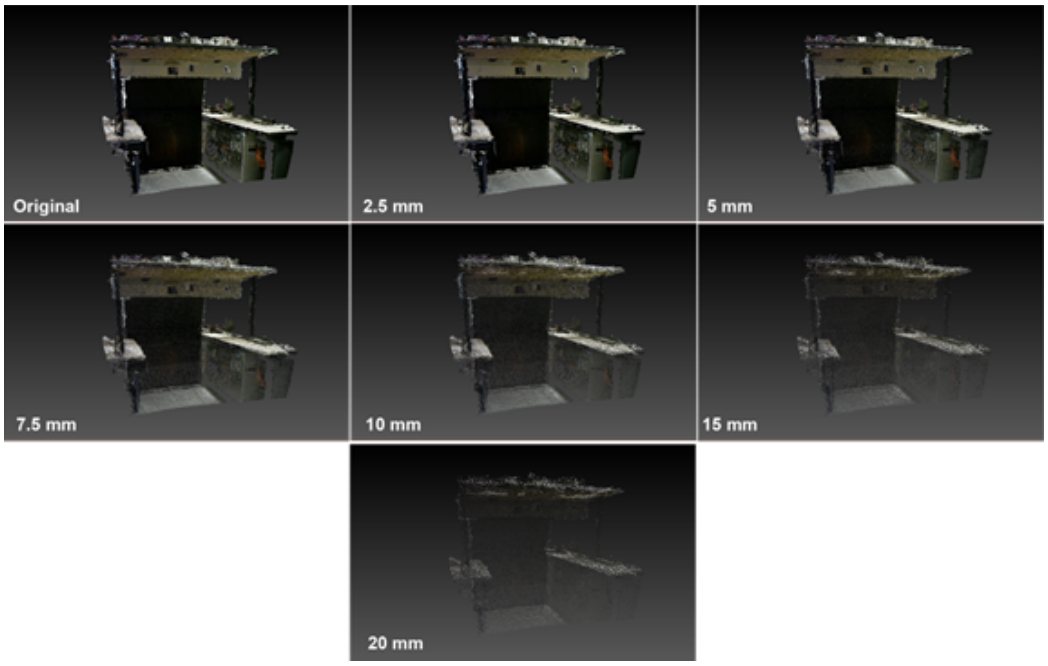


Figure 4.11: Table model after each distance parameter is applied.

Table 1 shows the amount of points remaining after the Poisson-Disk filter is applied using the different distance parameters.

Table 1: The Poisson-Disk filter results after 6 distance parameters are applied to each point cloud model.

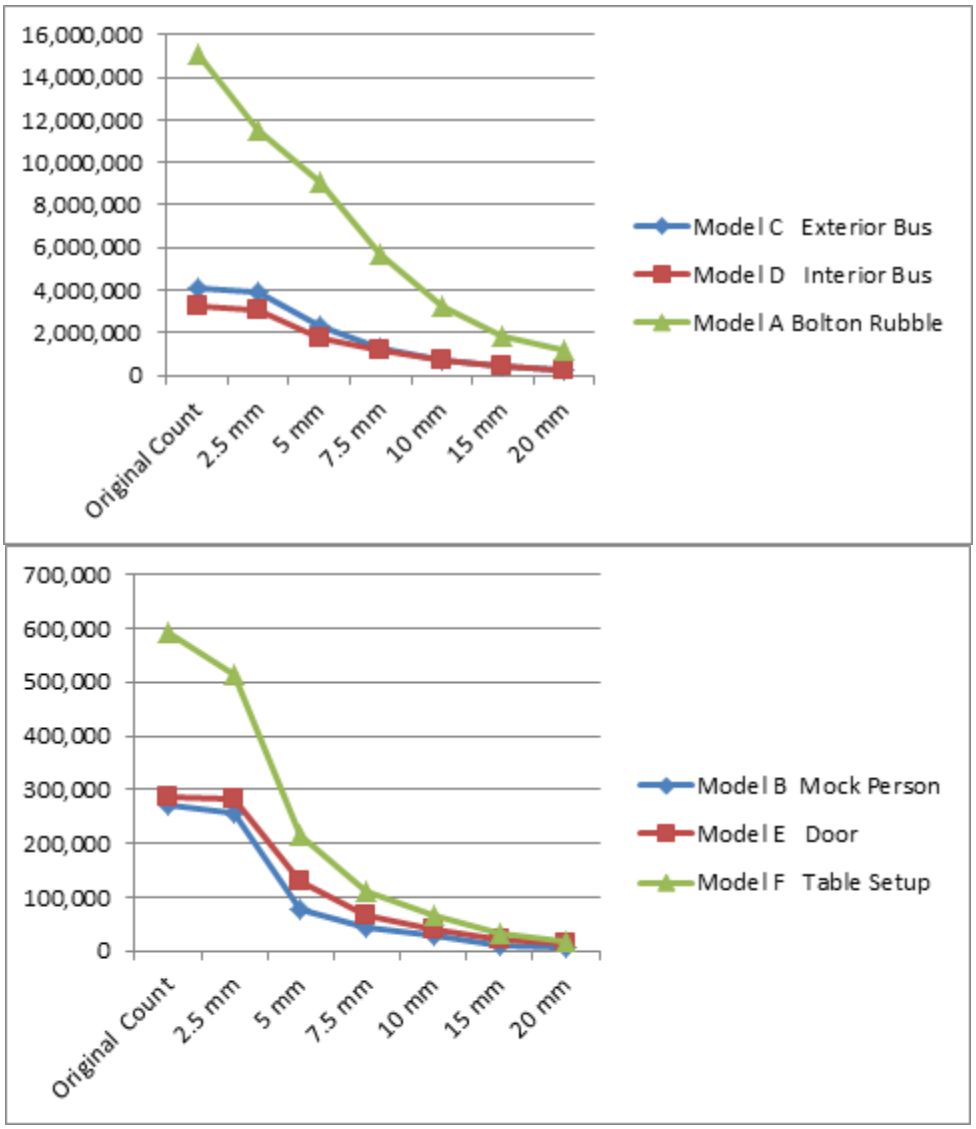
		Filter Distance					
Model	**Original Vertex Count**	**2.5 mm**	**5 mm**	**7.5 mm**	**10 mm**	**15 mm**	**20 mm**
Model A Bolton Rubble	15,101,504	11,515,824	9,032,658	5,732,641	3,215,899	1,851,614	1,173,829
% Difference		23.74	40.19	62.04	78.70	87.74	92.23
Model B Mock Person	270,743	255,187	76,982	41,809	27,054	11552	6521
% Difference		5.75	71.57	84.56	90.01	95.73	97.59
Model C Exterior Bus	4,093,292	3,958,940	2,291,315	1,262,434	758,800	395,129	239,956
% Difference		3.28	44.02	69.16	81.46	90.35	94.14
Model D Interior Bus	3,234,271	3,095,648	1,768,503	1,182,213	757,545	419,040	268,253
% Difference		4.29	45.32	63.45	76.58	87.04	91.71
Model E Door	284,932	281,926	130,747	66,207	37,801	21,068	12,350
% Difference		1.05	54.11	76.76	86.73	92.61	95.67
Model F Table Setup	593,117	514,513	214,826	110,387	64,294	31,152	15,873
% Difference		13.25	63.78	81.39	89.16	94.75	97.32

It was discovered that a filter distance of 2.5 mm did not change the point count substantially. Models B, C, D and E were reduced by an average of 3.59%. Models A and F were reduced by 23.74% and 13.25% respectively, much larger amounts compared to the other models. This was possibly due to the overlapping scanned areas which produced a higher point density, and therefore redundancy. Filter distances larger than 2.5 mm, perhaps not surprisingly, resulted in larger vertex reductions. Filter distances of 5, 7.5, and 10 mm reduced the point count

by an average of 53.16%, 72.89%, and 83.77% respectively. The filter distances 15 and 20 mm produced average point reductions of 91.37%, and 94.77% respectively.

The point reduction curves for each filter distance are shown in Table 2. The top chart displays results for point clouds larger than a million points. The bottom chart displays point clouds with less than a million points.

Table 2: The point reduction curves after each distance was applied.



A filter distance of up to 10 mm displayed a large reduction of points for models B, E and F. Models with higher vertex counts displayed large reductions up to 15mm. Therefore, during step 1 of the Model Creation Pipeline a filter distance of 10 mm is used for point clouds containing less than a million points and for more than a million points a filter distance of 15 mm is used.

These experimentally derived filter distances should be taken as a heuristic that applies to data sets which are, broadly defined, "similar" to those likely to be found in urban disasters involving large amounts of building debris. The use of these filter distances on data collected in other environments are subject to experimental verification and are beyond the scope of this work. Other sensors will have different levels of resolution and the filter distances can be adjusted based on the sensor specifications. In our case we used the Microsoft Kinect.

4.2.2 Surface Reconstruction Technique Comparison

This experiment applied three surface reconstruction algorithms using the filtered point clouds to determine which is most effective at creating 3D surface detail. The techniques chosen were based on what was available within Meshlab. Each technique must avoid generalizing surface details—thus removing potential details from the real world—and must not add artifacts to the model that could be construed to represent details in the real world that are simply not present. Keeping in mind that time is of the essence during a real-world disaster response, the processing time when running the algorithm must be as low as possible. Resulting point and polygon counts should be as low as possible in order to allow efficient rendering within the DSR system.

The chosen technique will be used to add surface detail to the point clouds during the Model Creation Pipeline and ensure that a model can be provided in time-critical scenarios. Each point set is provided as input to each of the three surface reconstruction algorithms: Poisson, Marching Cubes, and the Ball-Pivoting Algorithm (BPA). The resulting vertices, polygons, and processing times are recorded and compared. A visual inspection of the model is conducted to determine which technique produced a model that best represents the scanned terrain in a detailed comparison of the real-world terrain. The hardware used for the experiment is a PC with an Intel i7 quad-core CPU, a GeForce GTX680M GPU and 16 GB RAM. It is assumed that similar hardware would be available to first responders in the field.

Table 3: The surface reconstruction data results.

	Surface Reconstruction Method								
	Poisson			Marching Cubes			Ball Pivoting		
Model	Vertices	Faces	Time (sec)	Vertices	Faces	Time (sec)	Vertices	Faces	Time (sec)
Bolton Rubble	453,541	906,980	144.928	4,723,386	9,018,062	5054.24	1,868,618	1,509,858	5.71 hrs
Mock Person	28,431	56,548	6.941	36,557	69,669	0.34	25,950	41,778	3.548
Exterior Bus	297,400	594,766	45.446	507,666	898,810	95.825	419,838	575,698	2245.45
Interior Bus	248,539	496,852	44.118	388,999	685,686	127.375	395,129	630,755	3018.86
Door	40,411	80,514	8.283	51,225	97,092	2.174	37,801	63,987	9.623
Table Setup	42,860	82,201	9.539	110,687	209,372	4.94	64,406	113,278	33.462

The Poisson surface reconstruction technique proved ineffective at creating an accurate model surface. The input parameters used were an octree depth of 12 and solver divide of 8. The high octree depth ensured a higher level of detail, while a solver divide of 8 allowed faster processing time. Sample output can be seen in Figure 4.12.

Urban disaster environments—specifically building rubble—tend to contain different types of "voids" or holes. Poisson produces water-tight models by adding polygons to the surface where voids were present. These “filled voids” are undesirable due to the fact that holes are

actually quite important as they provide potential points of access to potential survivors below the surface. Those holes which are not inspected within the simulation could make the difference as to whether holes are examined by searchers. Therefore, Poisson was eliminated as a potential technique for the Model Creation Pipeline based on this particular application.

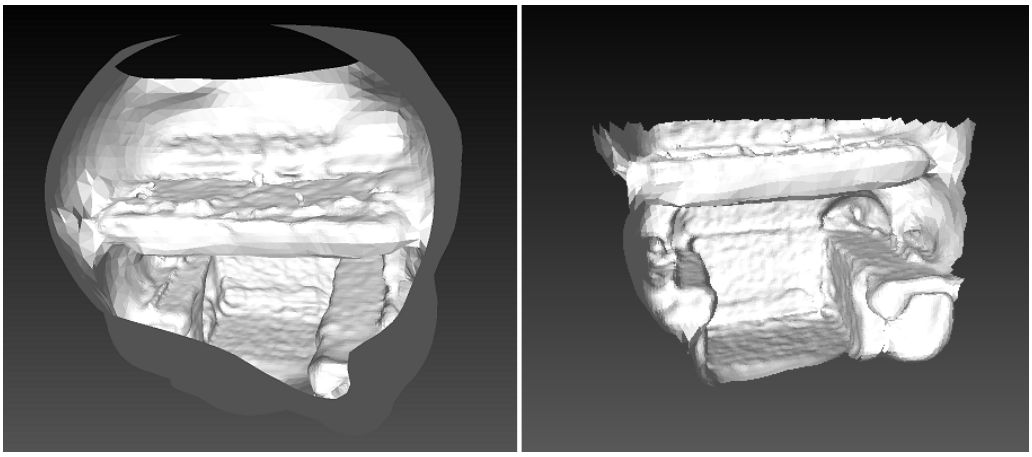


Figure 4.12: The Table model with Poisson applied. Voids are filled in and unwanted artifacts are created. The right model is after some of the added faces are manually deleted (note the holes that were removed by the algorithm).

The Marching Cubes (MC) and the Ball-Pivoting Algorithm (BPA) techniques produced similar results as seen in Figure 4.13. Unlike the Poisson technique, voids within the data remained intact and no artifacts were added.

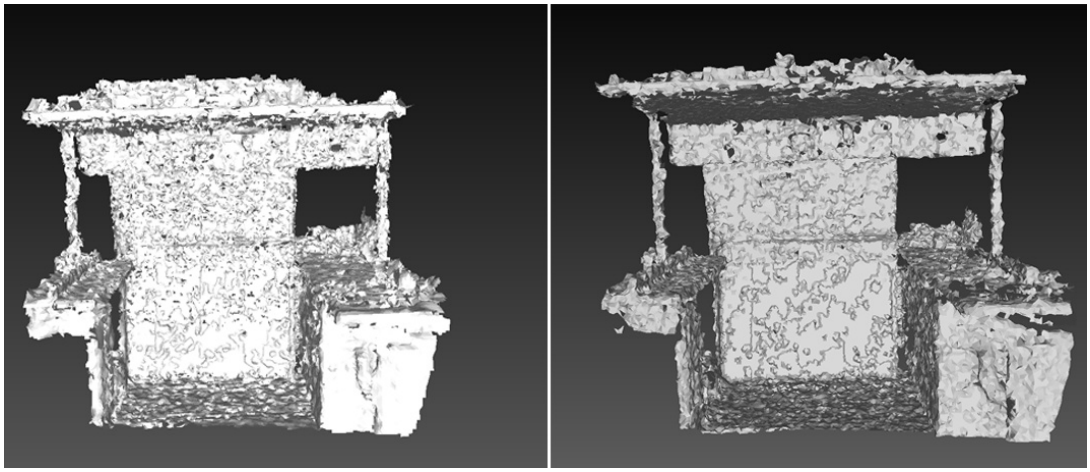


Figure 4.13: Marching Cubes and Ball Pivoting Algorithm output example. **Left:** A model after Marching Cubes is applied. **Right:** A model after Ball Pivoting Algorithm is applied.

A voxel grid resolution of 500 was used during MC and required less processing time than BPA. This was due to the reduced search space using voxels. When applied, many points can be included in one voxel, and voxels without points are not interpreted, thus, optimizing processing time. Although MC was the best performing from a computational point of view, each created model contained the most vertices and faces compared to Poisson and BPA. A MC model is not ideal when used within the DSR system since the simulation frame rate may suffer, but MC models are still a viable option.

For BPA, a radial distance parameter of 5mm more than the applied filtering distance was used (Section 4.2.1); i.e. 10mm filter distance + 5mm = 15mm radius. This distance was found to use the majority of the point cloud vertices based on the applied filter distance. BPA produced the least amount of faces, but required the most amount of processing time. This was due to a larger search space than MC. While using BPA, each point is interpreted requiring a considerable amount of time.

Considering the speed of MC, it can be used to create a quick proxy model for initial inspection. When time permits, BPA is the technique of choice because of the resulting low vertex and polygon counts. We found the output from the BPA technique produced a better looking model aesthetically that MC. As additional models are added to the scene, the cumulative polygon counts will begin to affect the frame rate of the DSR system. How many polygons before the performance will begin to unacceptably degrade is beyond the scope of this thesis.

The final models used in the DSR system are shown in Figure 4.14. Each model has been run through the Model Creation Pipeline using BPA and is shown with an applied texture.

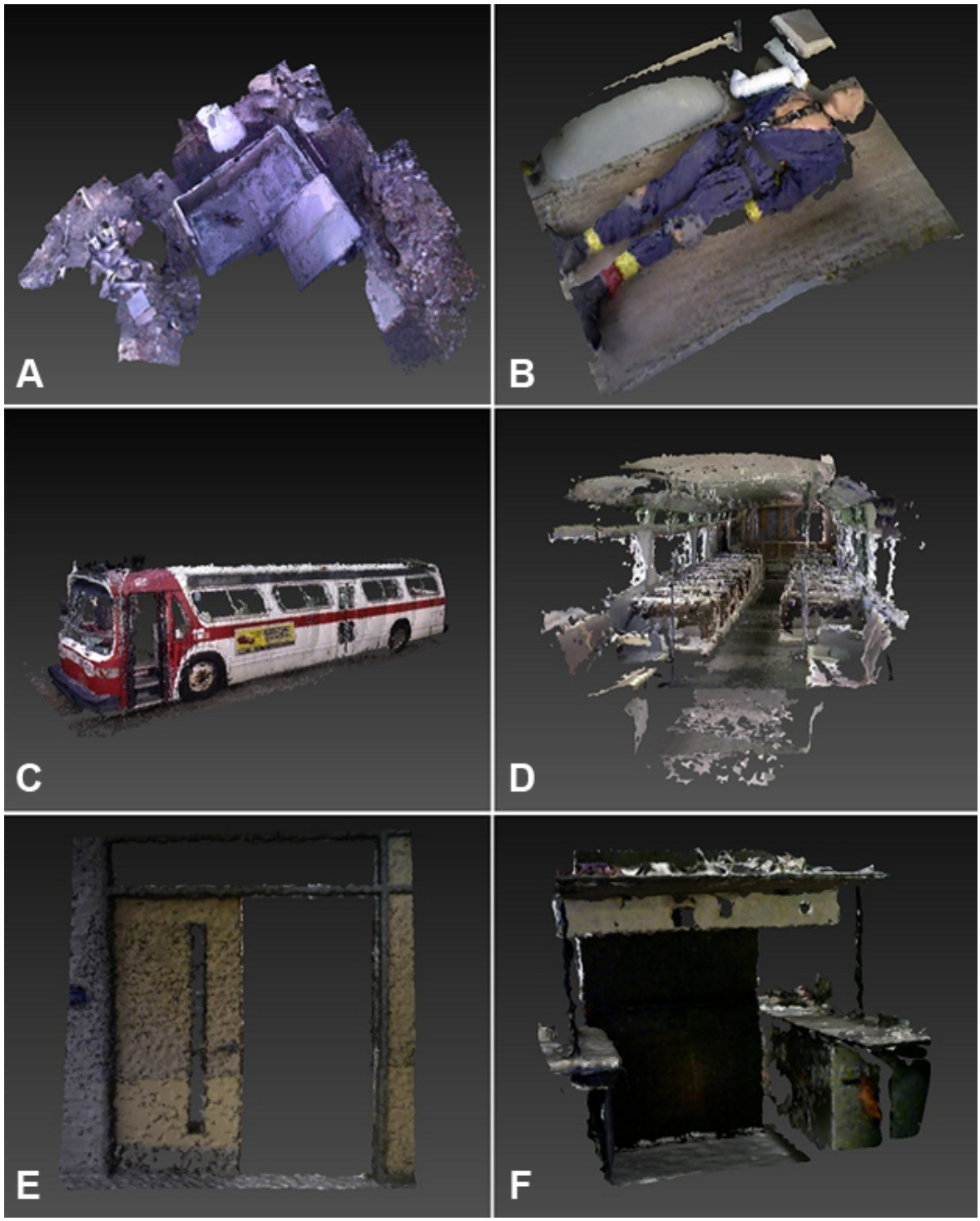


Figure 4.14: A: Bolton Rubble. **B:** Mock Person. **C:** Exterior Bus. **D:** Interior Bus. **E:** Door. **F:** Table Setup.

4.3 Time Trials

When using a simulation to depict reality in safety-critical situations, the accuracy of the simulated world is pertinent to an immersive experience. Inaccuracy in the model could lead to poor planning by the first responder using the simulation. With this in mind, we have chosen to demonstrate the feasibility of accurately measuring the time required for an individual to move on rubble. The movements of the actor on the simulated rubble should accurately depict how USAR personnel would actually move in the real environment. Movement in rubble is often difficult but we believed it can be modeled in some scenarios. An earlier publication of the DSR system [10] conducted a series of time-for-trials (TFT) on two models from the OPP RRP. This study was an initial experiment to evaluate whether the backend physics engine of the game engine was accurate enough to estimate TFT across terrain.

To verify that the timing estimates for traversal of a simulated rubble pile by actors on foot matches the measured time required to traverse the actual rubble environment, two areas were selected from the OPP RRP test environment—the rubble surrounding a collapsed building and the interior of a passenger bus positioned at an angle.

Table 4: The traits of a 24-year-old male that are used as parameters for the game actor.

Height	5’ 11”
Weight	170 lbs
Walk Speed	2.68 meters/s
Jump Height	19.0inches

Two sets of TFTs traversing the real-world and simulated rubble were run. Data concerning the height, weight, walking speed and jumping height of a male aged 24—the author of this work—was entered into the Unity environment as game actor parameters; see Table 4.

These traits are used by the physics engine when calculating collision detection and rigid body dynamics.

4.3.2 Test Results

The first set of trials was recorded on a section of terrain surrounding a partially collapsed building. The distance of the actual path to be traversed is 23.02 meters. The terrain contained hills and minor obstructions, formed from loose rubble consisting mostly of bricks and concrete which reduced the walking speed of the actor as seen in Figure 4.15. The results of the trial are shown in Table 5.

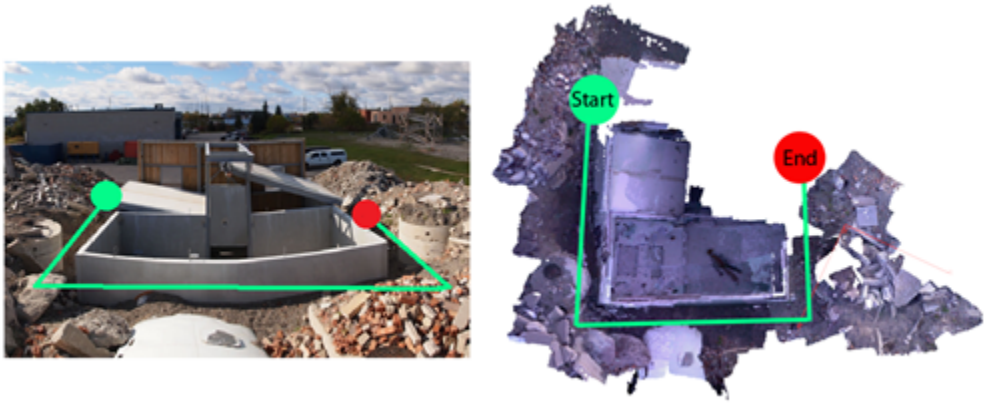


Figure 4.15: The real-world and simulated building rubble. Overlaid, the paths taken are shown.

Table 5: The recorded TFTs for the RRP rubble pile within the simulation and the real-world.

Trail Number	Real Life Time (in seconds)	Simulated Time (in seconds)
1	21	20.1
2	19.1	20.02
3	19.7	22.08
4	20.4	18.07
5	20	21.03
Average:	20.04	20.26
Meters/Second:	1.15	1.14

The second set of trials was recorded within the interior of a passenger bus. The distance of the path is 8.84 meters. The interior was, for the most part, intact besides some debris and a

few broken windows. While running the real-world time trials the person would have to walk up a slope of 45° while paying attention to where they were stepping to make progress as seen in Figure 4.16. The results of the trial are shown in Table 6.

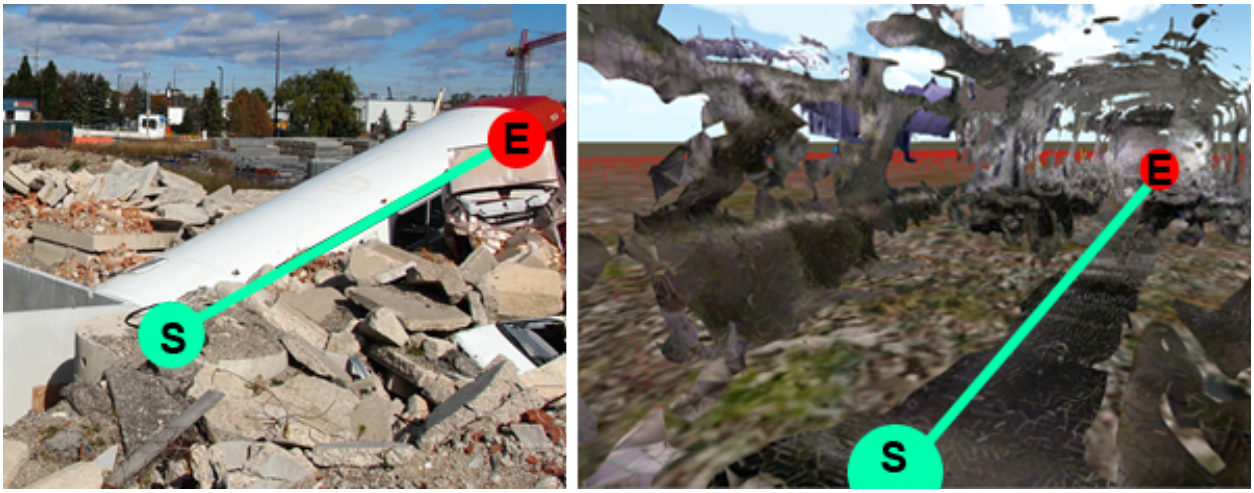


Figure 4.16: The real-world and simulated passenger bus. Overlaid, the paths taken are shown.

Table 6: The recorded TFTs within the simulation and the real-world for the bus.

Trail Number	Real Life Time (in seconds)	Simulated Time (in seconds)
1	28.5	24.36
2	32.8	29.68
3	29.9	37.09
4	28.6	23.54
5	32.6	27.51
Average:	30.48	28.44
Meters/Second:	0.29	0.31

The trials within the simulated environment closely matched the times measured in real world trials. If travelling at a constant speed with the walking speed used in Table 4, the time to traverse the distance of the path on the rubble and bus should have taken 8.59 and 3.29 seconds respectively. The third simulation trial within the passenger bus was around 10 seconds longer than the other trials. This was because of a model collision with the game actor and the actor becoming stuck. In this scenario a user would have to change position by moving away from the

collision. Regardless of this anomaly requiring more time, the average times recorded of traversal at the training facility and within the simulation differed slightly.

We conclude from these results that accurate walking times can be estimated within a game simulation using real-world parameters in at least some circumstances. Rescue personnel could traverse the rubble virtually and estimate times to cross the terrain accurately and safely.

These results conclude accurate walking times can be depicted within a game simulation using real-world parameters. Rescue personnel could traverse the rubble virtually and estimate times to cross the terrain accurately and safely. Additionally, features of the pile can be analyzed visually to determine needed support in areas of the terrain.

4.4 Shoring Calculator

This experiment addresses the accuracy of the DSR Shoring Mode. Recall, that shores are constructed to provide structural support for areas of a collapsed building that needs to be made safe for search and rescue operations to proceed. To construct appropriate shores, a TF crew would be sent into the rubble to measure an area requiring shoring manually. One of our goals in dealing with shoring within DSR is to allow the preparation of shoring measurements (cut tables) without ever needing to endanger a human when entering a partially collapsed structure. If the experiment is successful, the DSR system can then be used to safely measure areas and efficiently allocate lumber resources at a disaster site.

The T-Spot shore and Window/Door shore configurations were chosen to be simulated and then built. These particular shores were selected because they are used relatively frequently in actual operations, require few carpentry skills to construct and could actually be constructed within a lab with the resources available.

Two areas were modeled and virtually shored within the DSR system. The calculated shore measurements were then used to cut lumber and create a support which was placed within the real-world environment. If the shore supported the area firmly then the estimated measurements were considered accurate, providing weight of evidence that DSR can provide this facility.

4.4.1 Shoring Test Environments

The N-CART research lab was used to create analog environments that mimic areas requiring structural support by a shore. The RRP at Bolton, Ontario was not used for this experiment because of lack of access at the time of the experiment. Three desks were stacked to simulate an area needing a vertical shore (T-Spot) to support weight from above. The lab entrance was used to simulate an area needing both lateral and vertical shoring (Window/Door) to support an unstable frame. Both areas are shown in the following image along with overlaid measurements. Note, we are not arguing that these props represent actual rubble configurations that might be found in a disaster. Instead, we argue that if we can remotely measure, create models and deduce measurements that are substantively similar to actual measurements, we will have demonstrated evidence of the efficacy of DSR in similar tasks involving actual disaster debris and the resulting structures needing support within it.

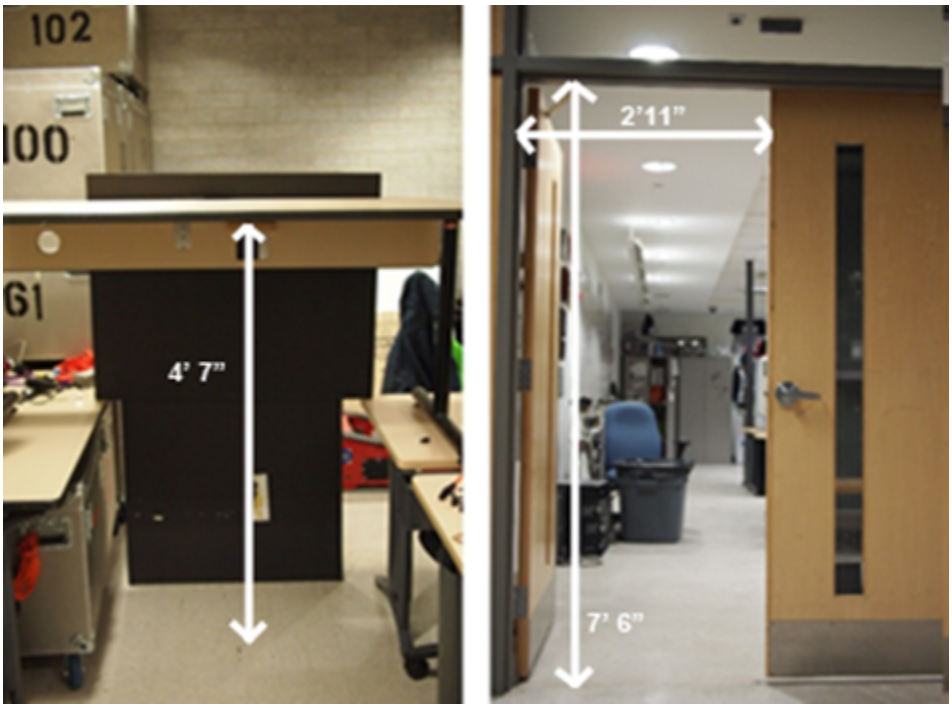


Figure 4.17: Both areas used for shoring experiments with overlaid dimensions.

4.4.3 Simulated Shores and Real-World Construction

The areas were sensed, modeled and then shored "virtually" using the Shoring Mode of DSR. A breakdown of the lumber pieces needed and their lengths to create the shore are provided to a user at the bottom-left of the simulation. The two areas shored within DSR and the output materials list are shown in Figures 4.19.

We took the DSR shoring measurements and attempted to construct the actual shores. Different types of lumber including 4x4, 2x4 and ¾" plywood were used to create each shore. The lumber was cut to the specified lengths as produced by the DSR Shoring Mode and assembled based on the instructions from the FEMA Field Operations Guide [20]. The shore was then inserted into the space to determine if the shores would fit.

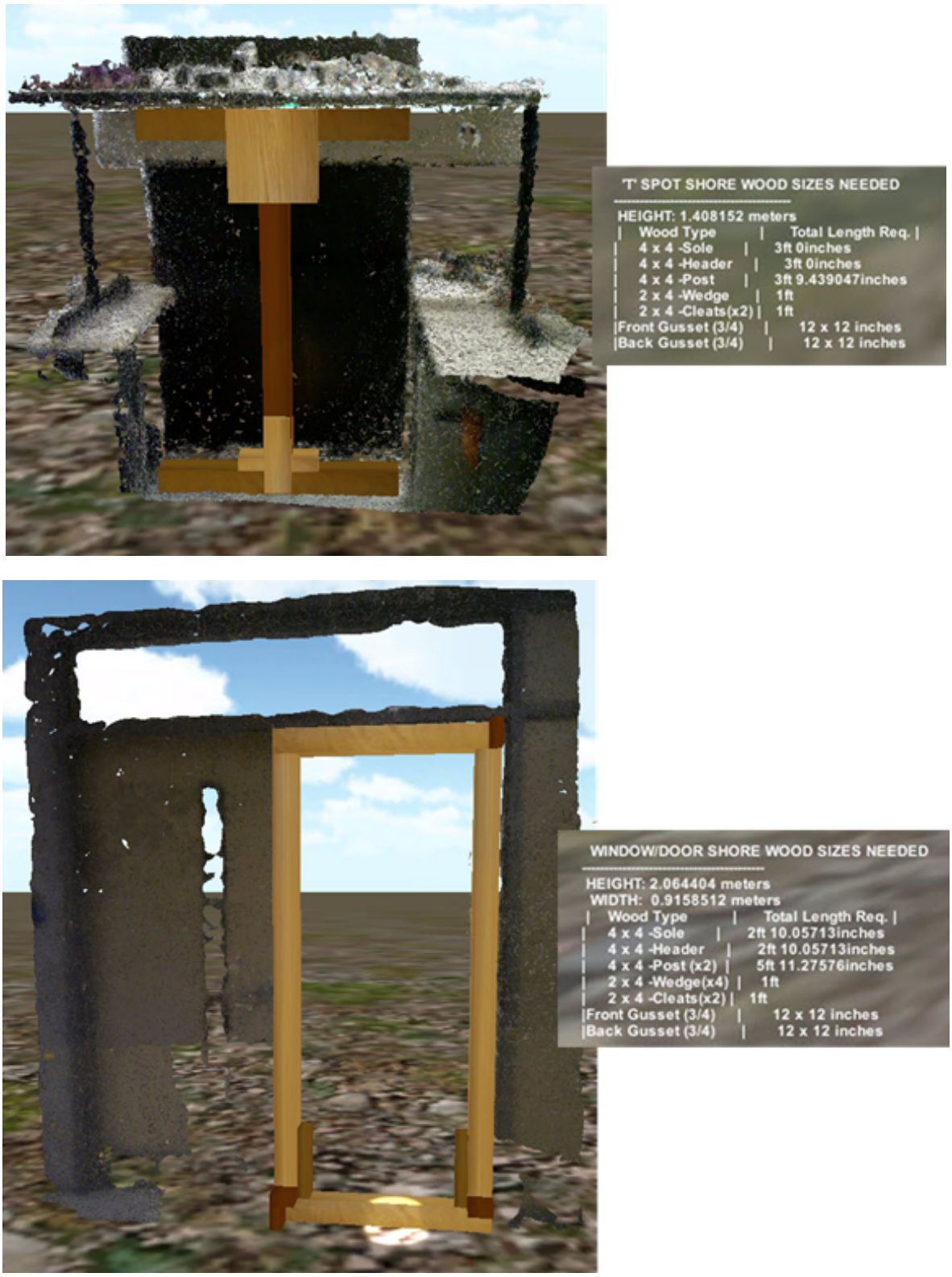


Figure 4.18: The T-Spot and Window/Door shores constructed within DSR. The output materials are shown to the right.

4.4.5 Shoring Evaluation

The constructed shores are shown in Figure 4.19. “Shims” or wedges are common components in any lumber construction project—including those involving USAR. It was found that shims were required to ensure a tight fit. Shimming is expected during construction as cuts may be based on rough measurements. The T-Spot shore produced a tight fit after shimming between the ground and load. The Window/Door shore required an extra two inches of wood to ensure a tight fit between the posts and sole piece. A 2x4 was placed between the sole and posts to make up for the discrepancy. The posts were not cut to the proper length because the virtual shore header intersected with a polygon from the model slightly lower than the top of the frame during the growing phase. To prevent this in the future, models could be manually cleaned further after having its surface reconstructed or provide the ability to manually adjust the virtual shore.

Figure 4.19: The built shores using lumber cut to the specifications from the DSR Shoring Mode.

Given this slight discrepancy in results, we have shown that it is feasible to construct actual shores from simulated data using DSR.

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Summary of Findings and Contributions

The motivation for our work has been to investigate the creation of a methodology and tool set for the capturing, accurately reproducing and interacting with models of urban disaster environments. This was done to provide USAR professionals with another way of safely viewing, inspecting and planning in the chaos before, during an urban disaster and a way of electronically archiving a disaster for future virtual interaction.

When a disaster strikes an urban area, the massive and numerous structural collapses of buildings is a common occurrence. Emergency First Responders engage in tasks such as inspecting the rubble structures, searching for, medically stabilizing and rescuing people in an environment which is inherently dangerous due to the building collapse debris and the hazards contained within and around it.

The goal of this thesis has been to gather and present evidence that it is possible to circumvent many of the dangers associated with direct physical inspection of disaster environments through remote sensing and accurate modeling. The Disaster Scene Reconstruction simulation system is designed to be used by Emergency First Responder for the purposes of planning in advance of actually committing scarce resources in reality. We envision that DSR can be employed in many scenarios including those where a Task Force may be many hours away from an incident. The disaster scene could be modeled by local personnel through DSR and the resulting model transmitted to the in-bound Task Force for planning. As DSR inherently supports incremental growth, components can be added to existing models as more data becomes available—supporting improved situational awareness of the incident. Finally, the DSR models

can be used as a powerful training tool as past incidents are captured and can be "replayed" in different scenarios.

Currently there are three methods for finding hidden and trapped people within building rubble. The first is for USAR personnel to manually inspect the debris using various sensors inserted through available or created access holes. However, the debris could be inaccessible and is normally quite dangerous, posing risks to the first responders. Also, in a large-scale disaster, it may be problematic to decide where to look within the rubble. The second method involves using USAR dogs and their keen sense of smell to locate trapped victims. Dogs may be able to negotiate terrain better than humans and access confined spaces, but the dangers to dogs are the same as those faced by humans. In addition, contaminants may preclude the use of air scenting by dogs as evident from the on-going Fukushima nuclear disaster. The last method focuses on remote environment inspection using specialized ground-based robots. Their limitations were outlined in Chapter 2, but the main concerns are their general inability to negotiate rubble terrain, maintaining contact using a wireless connection, and the risk of losing an expensive robot within the debris. A person cannot be found if the robot cannot get close enough to sense them or communication is lost. More recently, UAVs have been used to inspect large areas by flying over them. As UAV prices continue to decline, they are increasingly viewed as an alternative search tool. We extended our previous work in [5] the use a UAV and RGB-D sensor to fly over rubble and collect data. In the original work, the data collected could only be viewed within specialized point cloud viewers. We felt the data could be further processed to create 3D models for use within a game engine where we could take advantage of the human interface and physics components. Thus the risks posed to humans, dogs, and robots could be mitigated.

It is our claim that employing our Model Creation Pipeline (MCP)—as described in Section 3.2—with input point cloud data we can generate a geometrically-accurate 3D models as output. In essence, we have demonstrated techniques that are successful when modeling disaster environment point cloud data.

In our work, we discovered and demonstrated in Section 4.2.1 that a filter distance of 10mm for point clouds of less than a million points and 15mm for point clouds of greater than a million points were effective in reducing point density for the data sets we collected. In most cases these filter distances resulted in a point reduction of 90%. This reduction allowed for faster processing time in later steps of the MCP. In the future we will apply our chosen filter distances with other types of modeled terrain to determine whether our chosen parameters are applicable.

Based on our experiments from Section 4.2.2 two methods were found to be viable candidates to create3D surfaces from filtered points with trade-offs between the two. The Marching Cubes technique required the least processing time and could be used to create quick proxy models. However, this technique resulted in the highest polygon count out of the tested methods and could eventually impact simulation performance. If time-permitted, the Ball-Pivoting Algorithm is the preferred method to create models with the lowest face counts, but required the most processing time. The tested data is a small subset as to what could be encountered during a disaster response. We feel this approach is a viable method to modeling complex terrain and could be further verified using different types of test data.

We further claim that the Disaster Scene Reconstruction simulation system, described in Chapter 3.4, can be used to remotely inspect disaster environments and provide useful data to complete real-world tasks. Not only did we want to allow inspection of the modeled terrain, but

we wanted to create a toolset for simulating physical tasks that could provide pertinent information while planning a rescue initiative. To our knowledge, this is the first simulation system designed to utilize real-world data collected at a disaster scene to model and plan work actually to be conducted at the scene. Four user modes were developed based on the information found within standardized USAR training manuals [18, 20]. The modes are: Time Trial (Section 3.4.3.3), Ruler (Section 3.4.3.4), Shoring (Section 3.4.3.5), and Point of Interest (Section 3.4.3.6). These modes allow a user to perform dangerous tasks remotely and safely.

In Sections 4.3 and 4.4, we confirmed that two of the DSR modes can be used to accurately estimate physical actions. Disaster models created using the MCP and were imported within the DSR system to verify their functionality in the real world. In Section 4.3, we concluded that traversal times across terrain can be estimated using Time Trial mode with real-world human parameters. Disaster terrain is dangerous and typically requires more time to negotiate than simply walking in open ground. Regardless of this added time, the resulting average walking speeds in both the real-world and simulation terrain differed only by hundredths of a second.

In Section 4.4, we verified that Shoring mode can be used to create virtual shores and estimate material lengths. Normally this task would require a human to manually measure an area—placing the person in possible danger. Two areas were modeled and shored in simulation. The material lengths were gathered from the simulation and used to construct a real-world shore to be placed within the actual terrain. After placement, the real shore was found to firmly support the weight load and confirmed that using the output measurements from the Shoring mode is feasible.

The terrain modeling contributions and Time Trial Mode presented in this thesis were accepted as a conference paper at the Summer Computer Simulation Conference (SCSC 2013).

The purpose of this research was to determine if a game engine could be used to accurately estimate physical tasks using modeled disaster terrain. The results of the experiments showed that the goal of providing a simulation system to perform USAR-related tasks safely and remotely has been achieved in some cases. The modularity of the DSR system architecture allows other developers to extend the system and create other USAR-related functionality. One day (after additional development and testing) we hope this system could be used as a technological adjunct during an actual USAR operation to assist in saving lives.

5.1.1 Limitations and Restrictions

Despite having achieved the creation of a USAR simulation system to perform tasks using real-world data, there are limitations associated with collecting data and using the DSR simulation.

We used the Microsoft Kinect sensor [69]—an affordable consumer gaming product—to collect data when flying over terrain. It was found that the IR sensor used to detect depth levels can be negatively affected by the sun when used during daylight. As noted in Section 4.2, data collection is ideal during dusk or dawn to avoid IR interference. This limitation prevents scanning during the day or night when—arguably—most operations would take place. While it is possible to use other sensors to mitigate this problem, it is beyond the scope of this work. A UAV was used to fly over the dangerous terrain and collect the data remotely.

Currently, UAVs are being enhanced to fly in all sorts of weather conditions and environments. At the time of this thesis it was only possible to fly above ground that was clear of

overhead overhangs and other air-space obstacles. Indeed, flight could only take place during calm conditions (little or no wind and zero precipitation). Flying indoors, especially in the confined spaces found in most rubble environments was not possible. Presumably, as UAV technological improves, it will become easier to obtain data.

Depending on the length of the data collection, point cloud models could contain millions of points. Running the point clouds through the Model Creation Pipeline may require hours of processing time. Ideally, the data would be provided in real-time, but currently this is not possible. As hardware processing speeds increase, the time to run data through the MCP will decrease, specifically during the filtering and surface reconstruction steps however, our technique will always provide some delay in making models available. Another possibility is using GPU processing for the filtering and surface reconstruction steps as shown in [32]. The modularity of the MCP allows different techniques to be switched or added and will be an area of investigation in future experiments.

The amount of rendered polygons before performance would begin to suffer in the DSR simulation was not investigated. It is assumed that eventually the frame rate would begin to falter. Considering models could contain hundreds of thousands of polygons, eventually many modeled environments in one scene could lead to poor frame rates. To avoid this decrease in performance, separate simulation "scenes" could be created and switched between to avoid this potential problem. This functionality is not present within the current DSR system. Ideally, this would not be required and in the future may be avoided utilizing occlusion culling or GPGPUs.

5.2 Future work

A simulation, like all other computer programs, is dependent both on the data used and functionality provided. The DSR system is no different in this regard. A game engine is meant to allow a developer to create functionality for multiple platforms more rapidly than other development environments. DSR is a framework that can be built upon. Future work will involve implementing added functionality based on the information found within the various FEMA standards. This could include a "Spray Paint Mode" for terrain marking or adding additional shoring configurations to Shoring Mode. Some of the more ambitious goals for the system are outlined below.

The Unity game engine supports all major mobile platforms (iOS, Android, Windows). It can be assumed that all responders would have some sort of mobile smartphone available. A logical next step would be to integrate mobile controls with the current functionality. This would allow interaction with the simulation using touch-based controls. The difficulty with porting an application to mobile hardware is ensuring performance. The graphical capabilities of smartphones are not on par with PC GPUs. As mobile GPU hardware improves, this will likely be resolved.

Testing in varying types of environments would provide a stronger basis for verification for both the Model Creation Pipeline and DSR functionality. We modeled terrain at the OPP RRP and N-CART research lab. These environments are a subset of what could potentially be encountered during a disaster. Additional models should be tested in the future to further verify the accuracy of the system. In the unfortunate circumstance that an actual disaster occurs, we would also like to verify that this system could be used successfully at a resulting disaster scene.

Currently our system does not support more than one user. Our vision of this system involves having concurrent user instances to allow information sharing. This would involve implementing a client-server and database backend system. The premise would be to share real-time information found at the disaster site to in bound personnel and to allow them to interact in virtual tasks within the simulation. A user would become aware of the environment and events occurring within the simulation prior to their actual arrival—reducing the time to get up to speed. To our knowledge, no system currently uses 3D modeled environments during an active USAR operation in this manner. An extension of having concurrent users would be to have "types" of users. For example, a structural specialist user would be concerned with terrain inspection and assigning areas to be shored. Another example would be to have a medical user concerned with notifying where victims are located or are being attended to.

For this thesis we used the Meshlab point cloud processing tool because we could apply techniques to the data and see the results visually. The techniques we chose in the MCP are found within Meshlab, but as GPU processing techniques become available in the software they can be swapped or included into the pipeline. None of the techniques used in the pipeline currently use GPU processing and it is suspected that by using GPU enabled surface reconstructions techniques that the overall time it takes to create 3D models would be greatly decreased.

BIBLIOGRAPHY

[1] Wikipedia. (March 24, 2014). *Algo Centre Mall*. Available: http://en.wikipedia.org/wiki/Algo_Centre_Mall

[2] D. J. Van Hoving, W. P. Smith, E. B. Kramer, S. d. Vries, F. Docrat, and L. A. Wallis, "Haiti: The South African perspective," *SAMJ: South African Medical Journal,* vol. 100, pp. 513-515, 2010.

[3] I. Takewaki, S. Murakami, K. Fujita, S. Yoshitomi, and M. Tsuji, "The 2011 off the Pacific coast of Tohoku earthquake and response of high-rise buildings under long-period ground motions," *Soil Dynamics and Earthquake Engineering,* vol. 31, pp. 1511-1528, 2011.

[4] K. Webb. (2010, April, 27, 2014). *September 11, 2001: Remembering 9/11*. Available: http://www.kevinwebb22.com/september-11/september-11-2001-remembering-911

[5] A. Ferworn, J. Tran, A. Ufkes, and A. D'Souza, "Initial experiments on 3D modeling of complex disaster environments using unmanned aerial vehicles," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* , 2011, pp. 167-171.

[6] (April 27, 2014). *Meshlab*. Available: http://sourceforge.net/projects/meshlab/

[7] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007, pp. 1400-1405.

[8] R. Rathnam, M. Pfingsthorn, and A. Birk, "Incorporating large scale SSRR scenarios into the high fidelity simulator USARSim," in *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, 2009, pp. 1-6.

[9] J. Wang, M. Lewis, and J. Gennari, "A game engine based simulation of the NIST urban search and rescue arenas," in *Simulation Conference*, 2003, pp. 1039-1045.

[10] A. Ferworn, S. Herman, J. Tran, A. Ufkes, and R. Mcdonald, "Disaster scene reconstruction: modeling and simulating urban building collapse rubble within a game engine," presented at the Proceedings of the 2013 Summer Computer Simulation Conference, July 7-10 2013, Toronto, Ontario, Canada, 2013.

[11] (March 20, 2014). *Merriam-Webster Dictionary*. Available: http://www.merriam-webster.com/dictionary/disaster

[12] "Damage Situation and Police Countermeasures associated with 2011Tohoku district - off the Pacific Ocean Earthquake," National Police Agency of Japan March 11, 2014.

[13] J. Casper and R. R. Murphy, "Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 33, pp. 367-385, 2003.

[14] J. Yardley, "Report on Deadly Factory Collapse in Bangladesh Finds Widespread Blame," *Nytimes. com,* 2013.

[15] E. Canada. Emergency Management Basics. Available: http://www.ec.gc.ca/ouragans-hurricanes/default.asp?lang=En&n=31DADDF5-1

[16] E. M. Australia. (2004, Urban Search and Rescue: Capability Guidelines for Structural Collapse Response. Available: http://www.em.gov.au/Documents/Manual16-USARCapabilityGuidelinesforStructuralCollapseResponse.pdf

[17] I. F. S. T. Association, *Technical Rescue for Structural Collapse* vol. First Edition, 2003.

[18] P. S. Canada, "Canadian Urban Search and Rescue (USAR) classification guide," 2014.

[19] I. F. S. T. Association, *Urban Search and Rescue in Collapsed Structures*, 2005.

[20] FEMA, "National Urban Search & Rescue (USAR) Response System: Rescue Field Operations Guide," *US Department of Homeland Security,* 2006.

[21] J. L. Casper, M. Micire, and R. R. Murphy, "Issues in intelligent robots for search and rescue," 2000.

[22] S. Costo and R. Molfino, "A new robotic unit for onboard airplanes bomb disposal," in *35th International symposium on robotics ISR*, 2004, pp. 23-26.

[23] R. R. Murphy, "Trial by fire [rescue robots]," *Robotics & Automation Magazine, IEEE,* vol. 11, pp. 50-61, 2004.

[24] A. Ferworn, J. Tran, A. Ufkes, S. Herman, and C. Kong, "Establishing network connectivity under rubble using a hybrid wired and wireless approach," in *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, 2012, pp. 1-2.

[25] R. L. Finn and D. Wright, "Unmanned aircraft systems: Surveillance, ethics and privacy in civil applications," *Computer Law & Security Review,* vol. 28, pp. 184-194, 2012.

[26] (May 10, 2013). *Canadian Mounties Claim First Person's Life Saved by a Police Drone*. Available: http://www.theverge.com/2013/5/10/4318770/canada-draganflyer-drone-claims-first-life-saved-search-rescue
[27] Y.-H. Shyy, "Laser range finder," ed: Google Patents, 1993.
[28] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Robotics and Automation. Proceedings. IEEE International Conference on*, 1985, pp. 116-121.
[29] C. Ribeiro, A. Ferworn, and J. Tran, "WIRELESS MESH NETWORK PERFORMANCE FOR URBAN SEARCH AND RESCUE MISSIONS," *International Journal of Computer Networks & Communications,* vol. 2, 2010.
[30] Microsoft. (April 27, 2014). *Develop with Kinect for Windows*. Available: http://www.microsoft.com/en-us/kinectforwindows/develop/
[31] J. Smisek, M. Jancosek, and T. Pajdla, "3D with Kinect," in *Consumer Depth Cameras for Computer Vision*, ed: Springer, 2013, pp. 3-25.
[32] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton*, et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), IEEE International Symposium on*, 2011, pp. 127-136.
[33] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3d object modeling," *The International Journal of Robotics Research,* vol. 30, pp. 1311-1327, 2011.
[34] V. Frati and D. Prattichizzo, "Using Kinect for hand tracking and rendering in wearable haptics," in *World Haptics Conference (WHC), 2011 IEEE*, 2011, pp. 317-321.
[35] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended kinectfusion," 2012.
[36] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors,* vol. 12, pp. 1437-1454, 2012.
[37] P. C. Library. (2011, April 27, 2014). *PCL and Toyota Code Sprint 2011*. Available: http://pointclouds.org/news/2011/09/19/pcl-tocs-2011/
[38] (2013, April 27, 2014). *Leica Geosystems Partnership*. Available: http://pointclouds.org/news/2013/01/08/leica-geosystems-partnership/
[39] R. B. Rusu and S. Cousins, "3D is Here: Point Cloud Library (pcl)," presented at the Robotics and Automation (ICRA), IEEE International Conference on, 2011.
[40] (March 24, 2014). *The Visualization and Computer Graphics Library (VCG)*. Available: http://vcg.isti.cnr.it/~cignoni/newvcglib/html/
[41] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *Visualization and Computer Graphics, IEEE Transactions on,* vol. 5, pp. 349-359, 1999.
[42] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *ACM Siggraph Computer Graphics*, 1987.
[43] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," in *Proceedings of the fourth Eurographics Symposium on Geometry Processing*, 2006.
[44] P. C. Library. (April 27, 2014). *Point Cloud Filters*. Available: http://docs.pointclouds.org/trunk/a02945.html
[45] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Transactions on Graphics (TOG),* vol. 5, pp. 51-72, 1986.
[46] N. J. Mitra, A. Nguyen, and L. Guibas, "Estimating surface normals in noisy point cloud data," *International Journal of Computational Geometry & Applications,* vol. 14, pp. 261-276, 2004.
[47] L. Kobbelt, S. Campagna, and H.-P. Seidel, "A general framework for mesh decimation," in *Graphics interface*, 1998, pp. 43-50.
[48] I. Langworth. (2012, April 26, 2014). *Reducing Polygon Counts and File Sizes*. Available: http://blog.artillery.com/2012/05/reducing-polygon-count-and-file-sizes.html
[49] J. Tran, A. Ufkes, M. Fiala, and A. Ferworn, "Low-cost 3D scene reconstruction for response robots in real-time," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, 2011, pp. 161-166.
[50] C. Nvidia, "Compute unified device architecture programming guide," 2007.
[51] G. Fái and J. Randrup, "Statistical simulation of complete events in energetic nuclear collisions," *Nuclear Physics A,* vol. 404, pp. 551-577, 1983.

[52] E. S. T. Systems. (2014, March 25, 2014). *Advanced Disaster Management Simulator*. Available: http://www.trainingfordisastermanagement.com/
[53] H. Tanaka, J. Oda, A. Iwai, Y. Kuwagata, T. Matsuoka, M. Takaoka*, et al.*, "Morbidity and mortality of hospitalized patients after the 1995 Hanshin-Awaji earthquake," *The American journal of emergency medicine,* vol. 17, pp. 186-191, 1999.
[54] M. Onosato, S. Yamamoto, M. Kawajiri, and F. Tanaka, "Digital gareki archives: An approach to know more about collapsed houses for supporting search and rescue activities," in *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, 2012, pp. 1-6.
[55] Nvidia. (March 24, 2014). *PhysX info*. Available: http://physxinfo.com/
[56] E. Games, "Unreal Development Kit (UDK)," *URL: http://www.udk. com/[1.9. 2010],* 2009.
[57] J. Wang, M. Lewis, and J. Gennari, "A game engine based simulation of the NIST urban search and rescue arenas," in *Simulation Conference, 2003. Proceedings of the 2003 Winter*, 2003, pp. 1039-1045.
[58] U. G. Engine, "Unity Game Engine-Official Site," ed: Online][Cited: October 9, 2008.] http://unity3d. com.
[59] D. R. Michael and S. L. Chen, *Serious games: Games that educate, train, and inform*: Muska & Lipman/Premier-Trade, 2005.
[60] B. G. ltd. (March 24, 2014). *Dental Implant Training Simulation*. Available: http://www.breakawaygames.com/serious-games/solutions/healthcare/
[61] G. Mobile, "Google Apps for Android," ed: Google.
[62] Apple. (April 27, 2014). *Apple iOS7*. Available: http://www.apple.com/ca/ios/
[63] J. S. Miller and S. Ragsdale, *The common language infrastructure annotated standard*: Addison-Wesley Professional, 2004.
[64] K. L. Murdock, *3DS MAX 9 BIBLE (With CD)*: John Wiley & Sons, 2007.
[65] E. Campbell. (March 24, 2014). *Over 50 Unity Games in Development for Wii U*. Available: http://ca.ign.com/articles/2014/03/24/over-50-unity-games-in-development-for-wii-u
[66] Unity3D. (March 25, 2015). *Unity Engine Game Showcase Gallery*.
[67] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, 2007, pp. 281-288.
[68] A. Bond, "Havok FX: GPU-accelerated physics for PC games," in *Proceedings of Game Developers Conference 2006*, 2006.
[69] (March 23, 2014). *Microsoft Kinect*. Available: http://www.microsoft.com/en-us/kinectforwindows/