Disaster Scene Reconstruction:
Modeling and Simulating Urban Building Collapse Rubble within a Game Engine

Alexander Ferworn, Scott Herman, Jimmy Tran, Alex Ufkes, Ryan Mcdonald
Department of Computer Science
Ryerson University
Toronto, Canada
{ aferworn, scott.herman, q2tran, aufkes, ryan.mcdonald }@ ryerson.ca

Keywords: USAR; Rubble; Reconstruction; Physics; Disaster

Abstract

Various natural and human-made events can occur in urban settings resulting in buildings collapsing and trapping victims. The task of a structural engineer is to survey the resulting rubble to assess its safety and arrange for structural stabilization, where necessary. Urban Search and Rescue (USAR) operations can then begin to locate and rescue people. Our previous work reported the use of an Unmanned Aerial Vehicle (UAV) equipped with a RGB-Depth sensor to build 3D point cloud models of disaster scenes. In this paper we extend this work by converting the point clouds into 3D models and importing them into a state-of-the-art game engine. We present a method to use these models to allow first responders to interact with the simulated rubble environment in real-time, without risk to human life. Experiments are conducted measuring traversal time both in the real world environment and using the simulation. We argue that this work will improve the safety of workers and allow a better understanding of extremely dangerous environments without unnecessary exposure during disaster response planning.

1. INTRODUCTION

Various natural and human-made events, such as the 2010 Haiti earthquake or the 2011 Fukushima disaster, can occur in urban settings resulting in buildings collapsing and trapping victims within the resulting rubble. Urban Search and Rescue (USAR) teams are deployed to locate, medically stabilize and rescue trapped "patients"[*]. 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 understanding—or situational awareness—of the rubble environment allows these people to better decide how to proceed with searching for and extricating patients. Our intent is to provide an accurate visualization tool that can be used at the scene of a disaster to enhance the situational understanding of first responders. By providing an accurate 3D model of the disaster scene, engineers and those involved in the search can view the scene from a variety of perspectives. This could potentially help to reduce the time it takes to find and rescue patients. 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).

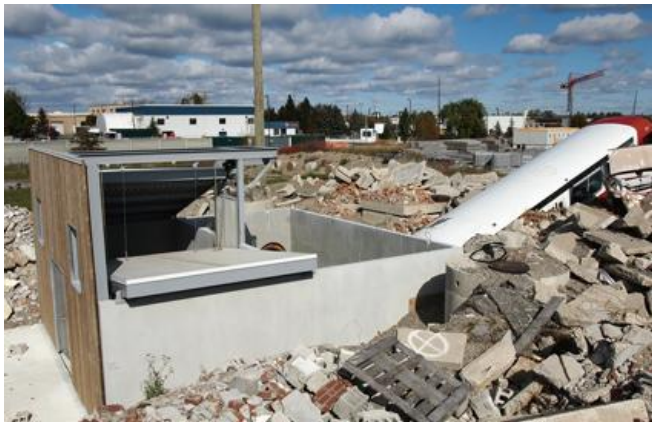


Figure 1. A mock disaster scene involving peripheral rubble and a partially collapsed building in Bolton, Ontario. The walls of the building obstruct the view to potentially trapped victims.

Our previous work has shown that using a UAV with an attached RGB-Depth sensor (such as the Microsoft Kinect) can create dense 3D point cloud models of rubble from a safe distance [1]. The Ontario Provincial Police (O.P.P.) USAR training facility at Bolton, Ontario shown in Fig. 1 depicts a mock disaster scene consisting of a partially collapsed building and surrounding rubble. The walls of the building obstruct the view of search teams who would need to search the building's interior for hidden victims. In this scenario, we use a UAV and video/depth sensors to create a 3D point cloud model. Flying over the terrain allows a large area to be surveyed quickly and safely, requiring no human entry. The resulting point cloud can be viewed within a 3D modeling program such as Meshlab [2]. Using Meshlab a user of this visualization can easily become disoriented as the point cloud itself obscures their ability to imagine the actual rubble view. In addition, functionality which might assist the planning for an additional ground search is missing—most notably the ability to interact with components of the mesh in ways similar to the actual material the mesh depicts.

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

The use of game engines for serious simulation is becoming more common [3-5]. Their modular design allows their functionality to be applied to other domains, allowing the rapid creation of many diverse scenarios. 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 specialized rescue teams to arrive at a disaster site could potentially be hours—normally wasted. With accurate models, this time could support concurrent planning for the actual operation. This paper describes the approach to creating these models.

1.1. Contributions

To the best of our knowledge, we are the first to develop a system that allows for rapid creation and examination of real world disaster models in simulation. Our system provides functionality that allows a structural engineer/first responder to virtually examine real world disaster rubble and plan a rescue. The functionality provided is customized for the needs of a first responder to examine the terrain and find points of interest. The simulation is created in a practical timeframe from the point of scanning, creating a 3D model, and using the model within the engine.

We conducted a series of experiments to accurately predict the task of traversing rubble within a simulation and validate our results against actual rubble traversal. In a complementary experiment, we used the model to locate a mock victim that was not directly visible inside the building structure.

2. RELATED WORK

Using software to simulate real world environments is common practice. Common examples are the simulation of weather patterns [6] and flight [7]. These simulations are used to better understand environments and train personnel safely, effectively, and less expensively. Creating these simulations typically require specialists and a considerable amount of development time.

Various simulations have been created to better understand urban disasters. Wooden houses collapsing due to the Hansin-Awaji Earthquake resulted in many casualties. This led to a simulation of a wooden house building collapse, but required extensive 3D modeling time and the simulation itself proved computationally expensive [8]. The Unified System for Automation and Robot Simulation (USARsim) project [3] is used as the basis for the RoboCup Rescue Virtual Robot Competition [4], but focuses primarily on robot simulation. The NIST USAR Orange Arena has been simulated using UDK requiring two months to recreate the environment [5].

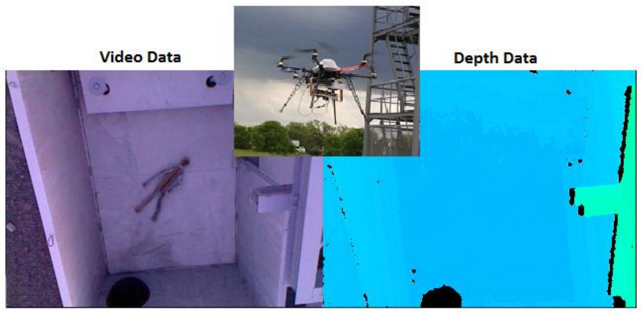


Figure 2. The UAV with attached RGB-Depth sensor with supplemental video and depth data produced from flying over the terrain. In the depth data, green represents objects near, and blue represents objects far from the sensor.

USARsim is an open-source project built on the commercially available game engine Unreal Development Kit (UDK) [9] to simulate robots and disaster scenarios. Large scale simulations have been constructed to simulate urban areas [4]. However, the disaster simulations are "artificial" in the sense that they model a disaster in an existing place but this disaster has not actually occurred and may not match the reality of a real incident at the same location.

In order to use a game engine as a simulation platform, 3D models must be provided in order to construct a scene. Using hardware such as a UAV and RGB-D sensor shown in Fig. 2 allows for terrain surveying from safe distances. The UAV and RGB-D sensor combination can create 3D point cloud representations of terrain quickly and at a low cost [1, 10, 11]. The resulting 3D point cloud is used in creating the models needed within a simulation.

RGB-D sensors have been used to create 3D models from point cloud representations in dynamic scenes, but only in small areas. This has been demonstrated by the Kinect Fusion project [12]. An extension of Kinect Fusion, called Kintinious [13], allows the scene reconstruction to be undertaken in much larger areas, but is currently unavailable as an Application Programming Interface (API).

The accuracy of the physics layer within simulators is essential in providing realistic models of the real world, especially how the simulators support interaction with the model. Popular physics engines such as Havok [14] and Nvidia PhysX [15] provide developers with physics-based APIs that can be used in simulating physical systems such as rigid or soft body dynamics. A comparison of different physics engines has been conducted in [16, 17]. The results concluded that each engine excels in simulating different aspects of physics such as friction or collision detection.

3. TECHNICAL APPROACH

From our previous work [1], we gather RGB-D data to reconstruct 3D point cloud models of disaster sites. The point cloud data cannot be imported directly into a game engine. It must be converted into a 3D mesh model.

3.1. 3D Model Creation

The Mesh Model Creation is done in two steps: point cloud data reduction; and surface reconstruction. The point cloud data is created from multiple overlapping scans that result in redundant points. An unfiltered point cloud model may consist of tens of millions of points – too large for a game engine to handle due to computation and memory constraints. Filtering for this step was done using the Poisson-Disk sampling filter [18] from Meshlab. The voxel size of 5cm was chosen to be consistent with the depth resolution of the RGB-D sensor. This typically removes most redundancy while maintaining details of the environment.

Next, the filtered point cloud is meshed in two steps: first by estimating the point normals for each vertex; then surface reconstruction is performed. The Meshlab implementation of Moving Least Squares [19] was used to smooth the data and estimate the vertex normals.

Finally, mesh surface reconstruction is performed using either Marching Cubes [20] or Poisson surface reconstruction [21]. Using marching cubes is ideal when a quick proxy model is adequate, but the resulting mesh is rigid and less accurate. If time permits, Poisson surface reconstruction is used producing a smoother mesh surface. Both techniques are implemented from Meshlab. The model is then UV mapped resulting in a JPEG that is used to provide texture on the model within the engine. Notice in Fig. 3 that the detail from the original point cloud data is not lost on the final mesh model. Fig. 3 A shows the original point cloud with 2,337,820 points. Fig.3 B shows the meshed point cloud with 425,253 points, an 81.8% decrease. Fig. 3 C shows the mesh from Fig. 3 B containing 826,510 polygons along with added texture.

This process of collecting data and creating the model is accomplished within 1 to 2 hours; considerably faster than creating the model by hand.

Viewing this resulting model within a game engine provides close up views of the terrain with the ability to change perspective easily using an intuitive control set.

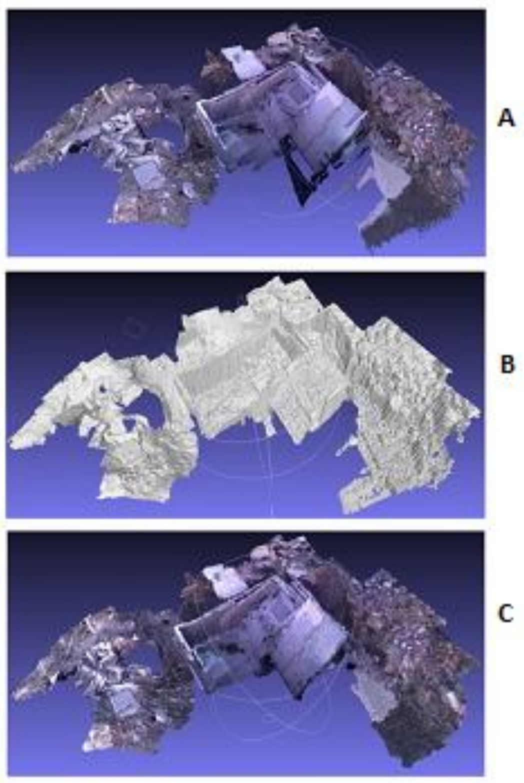


Figure 3. A: The dense 3D point cloud created consisting of the area shown in Fig 1. **B**: The point cloud filtered to create a mesh usable within a game engine environment. **C:** The textured 3D model ready to be used within a simulation.

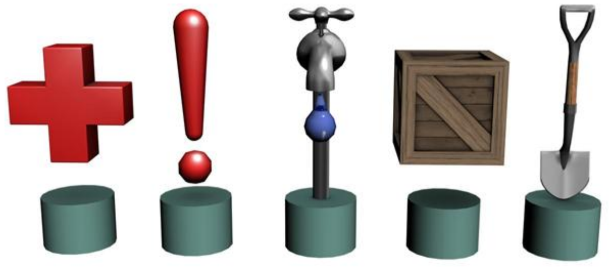


Figure 4. Various models that a user can place on the model to signify points of interest.

3.2. Game Engine

Using a game engine allows us to easily create simulations with added functionality not available in other 3D model viewers. Collections of prefabricated 3D models and prewritten code can be easily 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. Additional custom markers that are usable within the simulation to represent points of interest are shown in Fig. 4. These models are an example of the custom functionality that can be easily transferred from one project to the next.

The Unity game engine [22] was selected as the target development environment for the creation of our simulations. Unity has the advantage of being compatible with the file types produced from the model creation process (.OBJ), while UDK uses proprietary file types exported from commercial products such as Autodesk 3ds Max [23]. Unity provides the common features of all game engines (environment creation, programming modules, networking etc.). An additional advantage is that the Unity engine allows for deployment of the rubble model on a variety of platforms including personal computers, smartphones, tablets, and internet browsers. This is practical since the potential users of this application could use a variety of different platforms. An example of the rendered 3D model within the game engine is shown in Fig. 5 providing different perspective views.

4. EXPERIMENTS

When using a simulation to depict reality, 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, 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 believe it can be modeled.

4.1. Time Trials

To depict realistic traversal of the rubble pile within the simulation, a set of five time trials were conducted at the O.P.P. USAR training facility in Bolton, Ontario with a live actor traversing the real world rubble pile. The trials were recorded on a section of the rubble that surrounded the staged collapsed building. The distance of the path is 23.02 meters and surrounded the partially collapsed building. The terrain contained hills and minor obstructions, formed from loose rubble consisting mostly of bricks and concrete which negatively affected the walking speed of the actor. The path at the site in the real world and within the simulation is shown in Fig 6.

Data concerning the height, weight, running speed, walking speed and jumping height of a male aged 24 was also collected and is shown in Table 1. These values were used within the Unity environment as game actor parameters. Using this data, another set of five time trails took place along the same path in the simulated rubble using a game actor. By analyzing these results, we were able to adjust the game actor to better reflect the movement of the live actor.

Table 1. The traits of a 24 year old male that are used as parameters for the game actor.

Height	1.80 meters
Weight	77.11 kg
Run Speed	5.36 meters/s
Walk Speed	2.68 meters/s
Jump Height	0.48 meters

4.2. Model Acuity

Intentionally placing a mock victim within the building, we wished to determine whether the victim could be seen within the simulation. The dummy victim was obstructed from view when viewing the building from the perimeter as seen in Fig. 7 parts A and B. Model completeness and colour accuracy are two other areas which would affect an interaction with a model but were not examined in this study

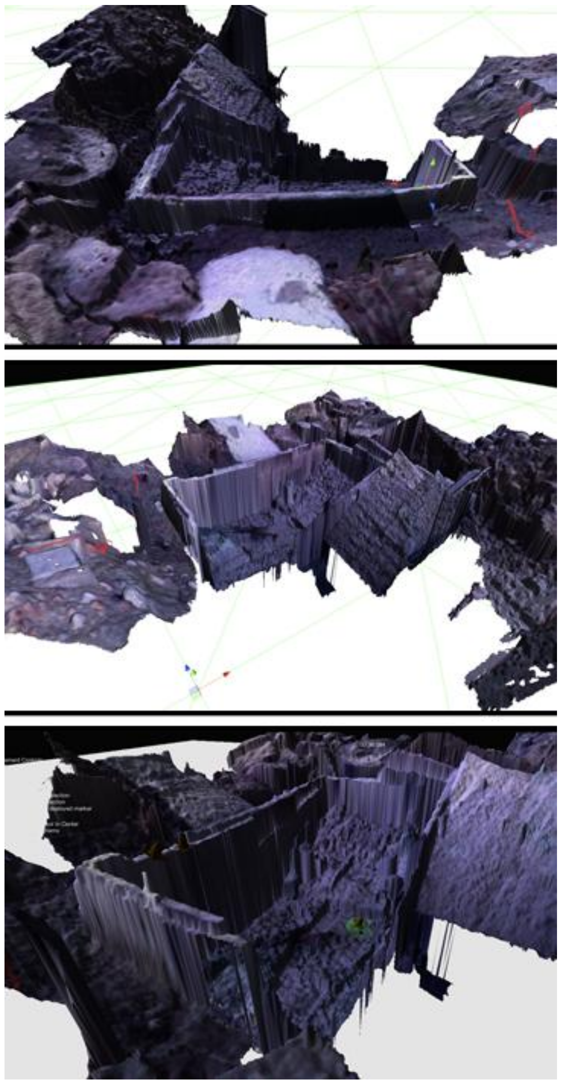
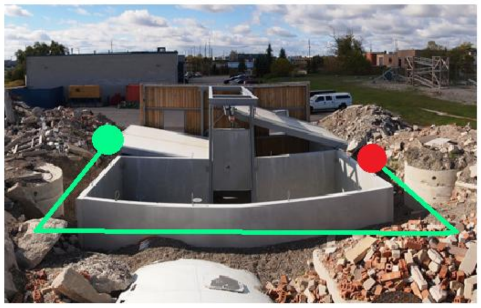


Figure 5. Top and Middle: Two perspectives of the 3D model used in the game engine environment. Bottom: The game actor's perspective within the Unity game engine while running a time trial.

5. RESULTS

The UAV allowed surveying terrain not otherwise accessible, but attaining a complete model proved difficult. Not knowing what terrain had been flown over proved to create holes within the 3D model and led to an incomplete model in some areas. Also, due to windy conditions the UAV swayed and was difficult to control effectively, thus affecting model accuracy.

The simulation ran at an average of 48 frames per second using a laptop with an Intel i-7 2.60GHz processor, 16GB of RAM, and an Nvidia GTX 680M graphics card. Five time trails were performed traversing the real world rubble terrain and within the game engine environment. Both sets of time trials with average times are shown in Table 2.



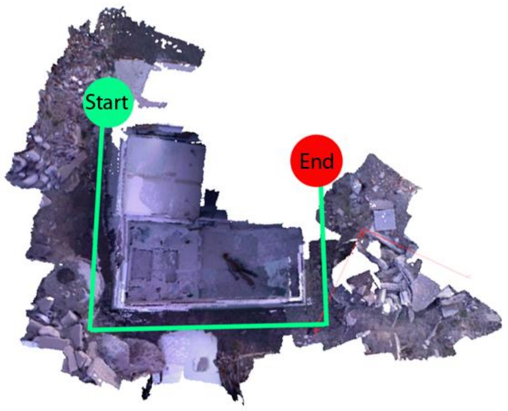


Figure 6. Top: The path taken shown at the training facility grounds. Bottom: The path shown in the simulated environment from a top-down perspective of the scene.

The trials within the simulated environment closely matched the real world trials. The time to traverse the distance of the path should have taken 8.59 seconds if travelling at a constant speed with the walking speed used in Table 1. Regardless of requiring more than double the amount of time, the average speeds of traversal at the training facility and within the simulation only differed by one hundredth of a second. 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.

Table 2. The recorded times for the simulated and real world time trails.

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

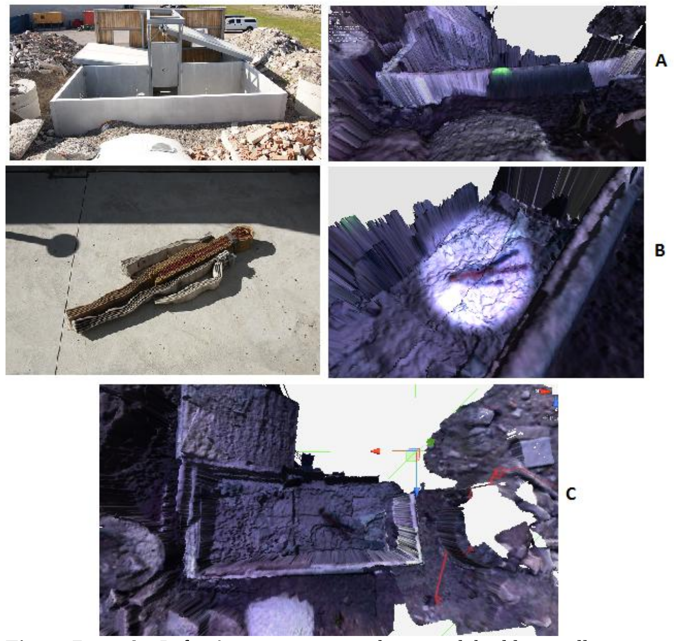


Figure 7. **A:** Left: A perspective at the staged building collapse in Bolton, Ontario. Right: Within the game engine. **B:** Left: A fire hose in the shape of a person within the building walls. Right: The found shape of the mock victim within the simulation using the added functionality of the flashlight. **C:** The top down perspective of the model showing fire hose victim on the model's texture.

When creating the mesh model from the point cloud, various holes were found in the resulting mesh. Fig. 7 part C displays various holes found within the mesh. The holes are due to not knowing what had already been mapped until post-processing the sensor data, thus, some areas of the environment are not mapped at all. Fig.7 part A displays a view of the terrain compared with the simulation. The simulated model matched the general shape of the walls and ground. Sections of the wall are not included in the model due to lack of data. Fig. 7 part B displays a fire hose mock victim (hose dummy) lying on the floor. The walls obstructed the view of the mock victim and could not be seen from the perimeter. The same image displays the located victim within the simulation using the added functionality of a "flashlight"; artificial lighting within the model. Our experiment clearly demonstrates the feasibility of potentially locating victims in simulation.

Perhaps the most pertinent observation is that this technique can be used to provide a usable, accurate and interactive 3D model within hours of the initial data collection. This means that such a model could be built and transmitted to USAR personnel while they are still in-bound to the disaster incident—allowing them to make decisions about the scene without actually having been there first. This could save valuable time and, potentially, save lives.

6. CONLUSION AND FUTURE WORK

We have demonstrated applying game and physics engine technologies to accurately create a model, supporting simple but common activities on a rubble pile. The traversal of terrain and the finding of points and other objects within a simulated disaster environment has been shown. Our time trial tests concluded that game actors can simulate realistic rubble traversal when using real world parameters such as walking speed and jump height. Added functionality such as a flashlight allowed viewing the model easily in dark textured areas.

Additional tests will be conducted to better simulate additional traits of real-world rescue personnel creating a more realistic experience while using the simulation. Other paths that have not been traversed will be tested within the simulation and then on the real world rubble. Using real-world parameters for a game actor may prove to be less useful than expected. Rather, a user would probably be more concerned with reaching a point of interest using whatever means necessary instead of accurately depicted movements. Whether or not this is true will be tested with user experience testing.

Using the rubble 3D model within Unity provides useful functionality and additional features can be implemented to assist in planning a search effort on the model itself. Probably of more interest to the search teams, rescuers and perhaps structural engineers is how the rubble actually behaves when interacted with in physical manner. We hope to explore this notion in future work. Destructible environments within game engines rely on models having materials classified prior to use of the application. We will attempt classifying the different areas of materials and sections automatically. Then the physics engine will be used to simulate interacting with the environment.

Model completeness will be our focus on future data collection experiments. We will also attempt modeling more complex environments (underneath rubble, trees, etc.). The 3D model mentioned in this paper contained voids in the mesh, indicating difficulty with the feature mapping algorithms used to map the terrain. Going forward, the Kinect Fusion and Kintinious APIs will be utilized in the model creation process. How these technologies will work onboard a UAV will be examined.

References

[1] Ferworn, A., J. Tran, A. Ufkes, and A. D'Souza. 2011. Initial experiments on 3D modeling of complex disaster environments using unmanned aerial vehicles. Paper read at 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), .

[2] *Meshlab*. Available from http://sourceforge.net/projects/meshlab/.

[3] Carpin, S., M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. 2007. USARSim: a robot simulator for research and education. Paper read at Robotics and Automation, 2007 IEEE International Conference on.

[4] Rathnam, R., M. Pfingsthorn, and A. Birk. 2009. Incorporating large scale SSRR scenarios into the high fidelity simulator USARSim. Paper read at Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on.

[5] Wang, J., M. Lewis, and J. Gennari. 2003. A game engine based simulation of the NIST urban search and rescue arenas. Paper read at Simulation Conference, 2003. Proceedings of the 2003 Winter.

[6] Semenov, M.A. 2007. "Simulation of extreme weather events by a stochastic weather generator." *Climate Research* no. 35 (3):203.

[7] Hays, R.T., J.W. Jacobs, C. Prince, and E. Salas. 1992. "Flight simulator training effectiveness: A meta-analysis." *Military Psychology* no. 4 (2):63-74.

[8] Onosato., Masahiko, Shota Yamamoto., Masahiro Kawajiri., and Fumiki Tanaka. 2012. Digital Gareki Archives: An Approach to Know More About Collapsed Houses for Supporting Search and Rescue Activities. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* : IEEE.

[9] *Unreal Development Kit*. Available from http://www.unrealengine.com/udk/.

[10] Tran, J., A. Ufkes, M. Fiala, and A. Ferworn. 2011. Low-cost 3D scene reconstruction for response robots in real-time. Paper read at Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on.

[11] Huang, A.S., A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. 2011. Visual odometry and mapping for autonomous flight using an RGB-D camera. Paper read at International Symposium on Robotics Research (ISRR).

[12] Izadi, S., D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, and A. Davison. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. Paper read at Proceedings of the 24th annual ACM symposium on User interface software and technology.

[13] Whelan, T., M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. 2012. "Kintinuous: Spatially Extended KinectFusion."

[14] *Havok Physics*. Available from http://www.havok.com/products/physics.

[15] *Nvidia PhysX*. Available from http://www.geforce.com/hardware/technology/physx.

[16] Boeing, A., and T. Bräunl. 2007. Evaluation of real-time physics simulation systems. Paper read at Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia.

[17] Zouhair, J., and D. Ellison. 2011. "Real-Time Physics Simulation Packages: An Evaluation Study." *WASET* no. 73:563-568.

[18] Cook, R.L. 1986. "Stochastic sampling in computer graphics." *ACM Transactions on Graphics (TOG)* no. 5 (1):51-72.

[19] Lancaster, P., and K. Salkauskas. 1981. "Surfaces generated by moving least squares methods." *Mathematics of computation* no. 37 (155):141-158.

[20] Lorensen, W.E., and H.E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. Paper read at ACM Siggraph Computer Graphics.

[21] Kazhdan, M., M. Bolitho, and H. Hoppe. 2006. Poisson surface reconstruction. Paper read at Proceedings of the fourth Eurographics symposium on Geometry processing.

[22] *Unity Game Engine*. Available from http://www.unity3d.com.

[23] *Autodesk 3ds Max*. Available from http://usa.autodesk.com/3ds-max/.