Virtual X-ray Vision by 3D Scene Reconstruction for Work in Nuclear Containment

Guang Jie 'Eric' Wei[1], Jack Condon[2], Mark Ho[3], James Yong[2], Edward Obbard[1*]

[1]Univeristy of New South Wales, School of Electrical Engineering and Telecommunications, UNSW Sydney, NSW 2052, Australia
[2]S1T2 (Story First Technology Second) Creative Technology Agency, Suite 3/104 Buckingham Street, Surry Hills NSW 2010, Sydney, Australia
[3]Australian Nuclear Science and Technology Organization, Nuclear Analysis Section, Locked Bag 2001, Kirrawee DC, NSW 2232, Australia

Abstract

The paper presents a low-cost and more interactive implementation of a Real-Time (RT) imaging system for 3-Dimensional (3D) scene reconstruction within a small-scale space in nuclear containment such as a hot cell. The system enables remote inspection and robotic manipulation, or alternatively increases usability and intuitiveness of interaction with objects in a hot cell if the operator is using mechanical master-slave manipulators. 3D scene reconstruction has been highly developed with a vast amount of open-source resources and free platforms associated with computer games development but the technology is not widely applied in the nuclear industry. In our system, we reconstruct a 3D scene on a completely free gaming engine from depth images captured by low-cost depth cameras. By developing our system on the gaming engine, we can render our reconstructed 3D scene in either 1st/3rd person perspective, or Virtual Reality (VR) forms. Interaction between system user and the reconstructed scene will be largely increased when the user can navigate freely inside the scene. We present a prototype system that can reconstruct and render in RT a 3D scene in the gaming engine in good resolution. Crucially for nuclear containment applications, the depth cameras are in fixed locations, include redundancy, and do not require zoom/pan control. This technology promises to improve operator safety, increase the usability and flexibility of hot-cell facilities, and promises significant cost savings in conventional viewing options such as hot cell windows.

1. Introduction

Virtual x-ray vision is an idea of using an imaging system to capture and display an inner environment of a space to a user so that the user doesn’t need to look through windows or go into the space in person. For current hot cells, the main observation methods are either through lead glass windows or monitors displaying image captured by cameras installed inside the hot cells. Windows have advantages in providing intuitive observation experience and good reliability, but have high cost and long lead time due to limited suppliers. Monitors with cameras extend the view provided by windows and add redundancy to the system with relatively low cost, but the usage is not intuitive compared to windows. Therefore, our aim is to implement an imaging system combining the advantages from windows and monitors with cameras, while eliminating their disadvantages. An overview of our imaging system that provides virtual x-ray vision is shown in Fig.1.

* Corresponding author; E-Mail: e.obbard@unsw.edu.au

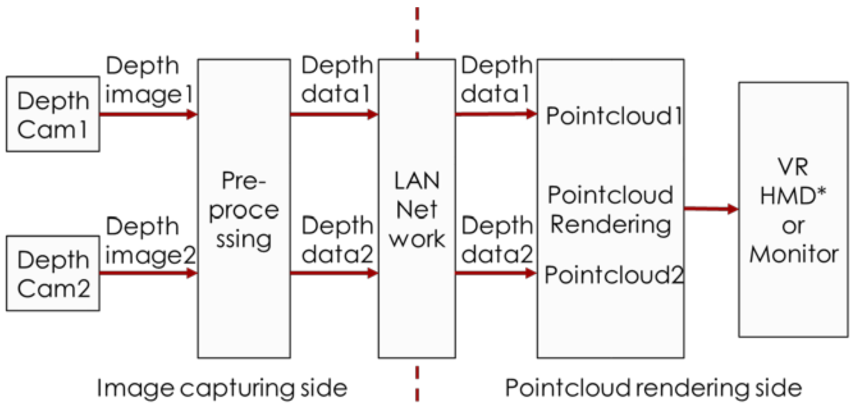

Fig.1 Process flow of our imaging system

The system can be separated into two sides, the image capturing side and the pointcloud rendering side, and both sides are connected by a LAN network. There are two channels in our system, which captures two depth images for a scene and constructs two pointclouds for 3D scene reconstruction from two different viewpoints. Tthe image capturing side will be closely attached to a hot cell, with cameras installed inside the hot cell. Location of the rendering side will be based on user's choice, which means the user can navigate and interact with reconstructed scene of interior of a specific hot cell in a remote area. Three important parts in our system are depth cameras (in our case, Microsoft Kinect), pointcloud and gaming engine (in our case, Unreal Engine).

1.1. Depth Camera

The depth camera has a built-in infra-red sensor that captures an array of depth values and a RGB camera that captures an image. The camera can capture depth images with a resolution of 512 x 424 and 1080p colored images. At the image capturing side, the depth cameras capture images that contain an array of depth values of the target environment inside a hot cell and they are pre-processed before being transmitted to LAN network. Each depth image is in 2D coordinate, so each pixel in the image carries a 2D location and a depth value.

1.2. Pointcloud

A pointcloud is a group of points in a 3D space. If these points have the correct relative locations, they can represent the 3D outline of an object. The pointcloud is constructed in the gaming engine from the depth value and the 2D location for each pixel in the depth image.

1.3. Gaming Engine

Gaming engine is originally for game development. In our project, we develop a “game” to receive depth data from the LAN network, construct, calibrate and display pointclouds. Gaming engine has its own advantages such as opensource functions developed by game fans and developers as well as enable our project to be tracked in the gaming industry.

In this project, we have implemented an imaging system with the above three core components in which the user can view and interact with reconstructed 3D scene in the form of pointclouds using VR head mount device (HMD) as well as controllers. In our assumption, if the interior of a hot cell can be reconstructed and displayed with our imaging system, our user will not need to stand in front of the hot cell to operate manipulators, but navigate the reconstructed 3D scene in a remote room and complete all tasks with more pleasant experience.

In this paper, we discuss our method to implement an imaging system providing virtual x-ray vision in section 2, results we obtained from the system in section 3, and possible impact of the system in the last section.

2. Method

Depth cameras are responsible for capturing depth images of our target scene (e.g. interior of a hot cell) and passing those images to rendering side. On rendering side, we extract depth information from received depth images to construct two 3D pointclouds and display the pointclouds in a VR HMD with the gaming engine. In this section, we will further discuss the whole set up from image capturing to pointcloud rendering side.

2.1. Image Capturing

At image capturing side, we have two depth cameras connected to a computer responsible for pre-processing. The camera set up is shown in Fig.2.

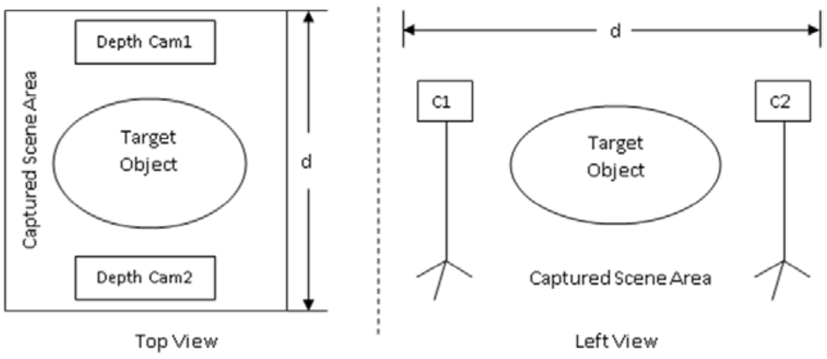

Fig.2 Cameras set up diagram

In the above configuration, we fixed the two depth cameras at the opposite sides of our target scene. With this set up, we can obtain two pointclouds without obvious occlusion. In terms of pre-processing, noise reduction is applied to each image after they are received from depth cameras. Next, depth value of each pixel in an image is mapped to a range of 0 to 255, which is approximately 1m to 5m distant from the camera. Lastly, mapped depth values are encoded into uint8 data stream for data transmission.

2.2. Depth Data Transmission

Depth camera used in our project can capture depth images with 512 x 424 resolution, and we need to have a frame rate over 90 fps to achieve a smooth user experience (without dizziness when moving user's head) in VR HMD. Applying simple mathematics for the above situation, we can expect nearly 20 million depth values are transmitted every second. Considering the data type of our depth values, we will have a data rate of 160 million bits per second, or 20 MByte/s. Compared to TCP/IP, which have extra process for ensuring successful packet transmission, we first chose UDP to achieve our desired image rate. However, during our experiment, we found that the unreliability of UDP transmission was beyond our expectation.

We then switched from UDP to TCP/IP for reliability with some sacrifice on the image rate, and currently our system can reach 60 fps.

2.3. Pointcloud Rendering

As mentioned in previous sections, our pointcloud rendering side is built in the gaming engine. By doing so, we can make full use of sophisticated functions in the gaming engine to manipulate and render our pointcloud, and display reconstructed 3D scene in VR headset with just one simple configuration. In this section, we will detailly discuss how we convert depth images from depth cameras to 3D pointclouds in the gaming engine.

2.3.1. Texture

In gaming engine, textures are images used in materials, describing appearance of objects [1]. When the depth values are received in the gaming engine, they are stored in a 2-dimensional texture with the same resolution 512 x 424 as that for depth camera infra-red sensor. Each pixel in the texture carries a depth value z. By extracting coordinate of each pixel and combine with the depth value and applying proper scaling, we then obtain a 3D coordinate (x, y, z) for a point. For a whole texture, we will have all 3D coordinate for pixels in this texture. For debugging reasons, we can also apply the texture on a simple object, such as a cube, and by observing the cube in experiment, we can tell whether we receive correct data from transmission side or not.

2.3.2. Material

After obtaining 3D coordinates for pixels in the texture, the next step will be putting particles to the corresponding position in a world space created in the gaming engine, and this is done in a component called material. We pre-make a group of small squares lining up in a line along x-axis as our particle chain, and with the number being the same as the number of pixels in a depth image (512 x 424 = 217088). Each particle has its own original location as shown in Eq.1 in the world space when imported.

$$Particle\ Initial\ Location = \left((x_1, x_2, \cdots, x_{217088}), y, z\right) \tag{1}$$

In material in the gaming engine, we will first obtain initial position (x, y, z) of a particle in the world space. We map the x value of the particle to a 2D coordinate system, which is called the uv coordinate. This mapping helps us to match our pre-built particles to pixels in our depth texture to extract 3D coordinate positions, which are world positions (WP) of the pixels. The mapping method is shown in Fig.3.

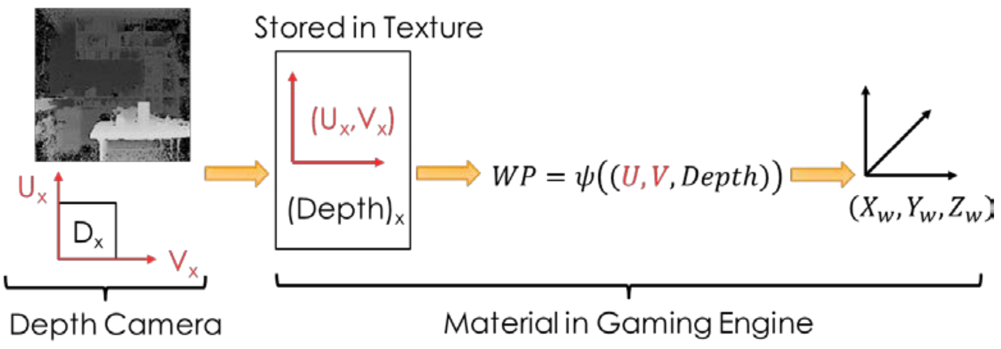

Fig.3 Mapping pixel location from 2D coordinate system to 3D coordinate system

Fig.3 illustrates when depth images are transmitted to the gaming engine, the uv coordinates and depth values are stored in textures, which are then used for coordinate mapping in materials.

2.3.3. Depth Scaling Compensation

The 3D pointcloud is generated from 2D depth images. This transformation introduces wrong scaling in x and y directions for objects in the images due to different depths, or distances to the cameras. The effect of transformation is shown in Fig.4.

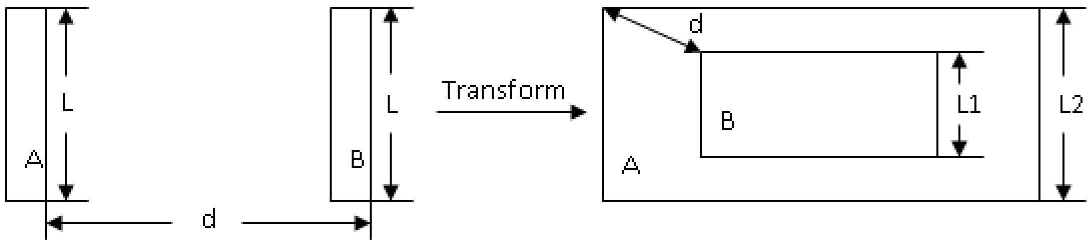

Fig.4 Object has different size due to different distance

The above figure illustrates that when two object A and B with the same height “L” and have a distance “d” are captured by a depth camera, their heights (and lengths as well) in a depth image will be scaled due to depth difference as “L1” and “L2”, and “L1<L2”. Therefore, we need to compensate this depth scaling to get the correct sizes of objects in generated pointclouds.

To compensate the depth scaling effect discussed above, we apply simple trigonometry as shown in Fig.5.

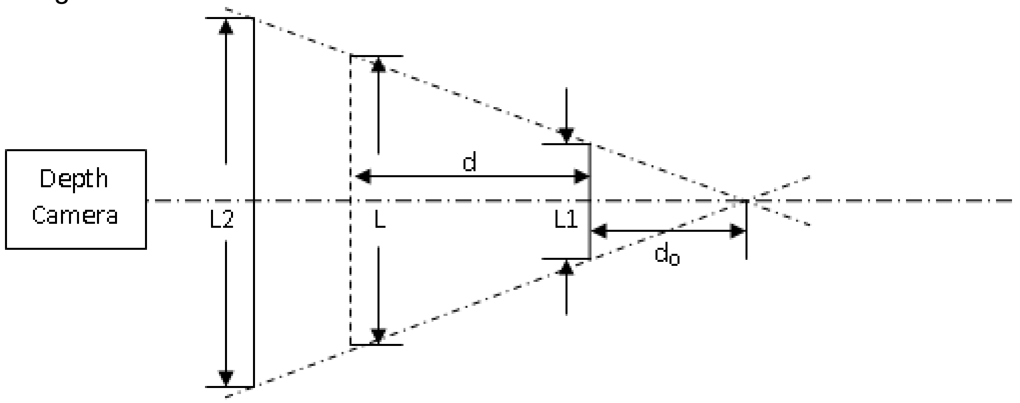

Fig.5 Size relationships of objects in different distances in the same image

In the figure, we take length “L1” as the farthest reference and find the corrected value for length “L” somewhere in the target scene. According to the code in image capturing side, the value of normalized depth is reversed, which means value 0 is the farthest distance and 1 is the closest distance to the depth camera. We first find the ratio between “L” and “L1” with Eq.2.

$$ratio = \frac{L_1}{L} = \frac{d_0}{d + d_0} \tag{2}$$

In Eq.2, d_0 is an estimated distance between length “L1” and top of the triangle illustrated in figure X. With this ratio, we can the work out the corrected length for “L” by applying Eq.3.

$$L_{corrected} = (L - 0.5) \times ratio + 0.5 \tag{3}$$

The term “0.5” is a shifting parameter for correcting the length from both tips (top, bottom, left and right) towards center. By repeating this process for every pixel in the depth texture in material, the depth scaling effect is compensated and obtain corrected positions for points in our pointclouds.

2.3.4. Pointcloud Control

To reconstruct a complete 3D scene, we need at least two pointclouds providing enough redundancy and eliminating occlusion. To calibrate the two pointclouds and form a complete 3D scene, a control system is built in the gaming engine. With this control system, a user can select, move and rotate pointclouds by using keyboard in normal gaming mode or using a pair of VR controllers in VR mode. Besides controlling pointclouds, user can also reset or save position parameters in game, and these parameters will be automatically loaded when the game starts.

3. Results and Discussion

Currently, with our imaging system, we have obtained the following results.

We have set up an image capturing system to capture images for target scene and pre-process them. Two depth cameras capture depth images at the image rate of 60 fps and transmit them to the pre-processing computer. In this computer, the depth images will be encoded into binary packets that are ready to be sent. Fig.6 shows two depth images captured by the two cameras for the same target scene.

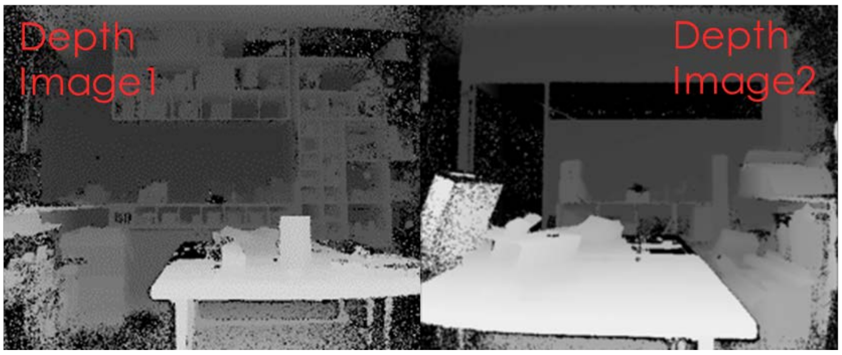

Fig.6 Depth images captured by two depth cameras

We have set up a server communicating between image capturing side and pointcloud rendering side, receiving image data from the pre-processing computer and sending the data to our rendering program in the gaming engine. This Python server can resolve packet drop problem when the gaming engine is communicating with external clients through LAN network.

We have implemented a program in the gaming engine to process received image data and generate two monochrome pointclouds corresponding to the two Depth cameras. In this process, depth data will be stored in a position texture. 3D coordinate of each point in our created world space is found and properly scaled by combining pixel location in the texture with the depth value. Then we have two monochrome pointclouds as shown in Fig.7. For better understanding the relationship between depth images and constructed pointclouds, we have put corresponding depth images together and point out some objects in both the images and pointclouds.

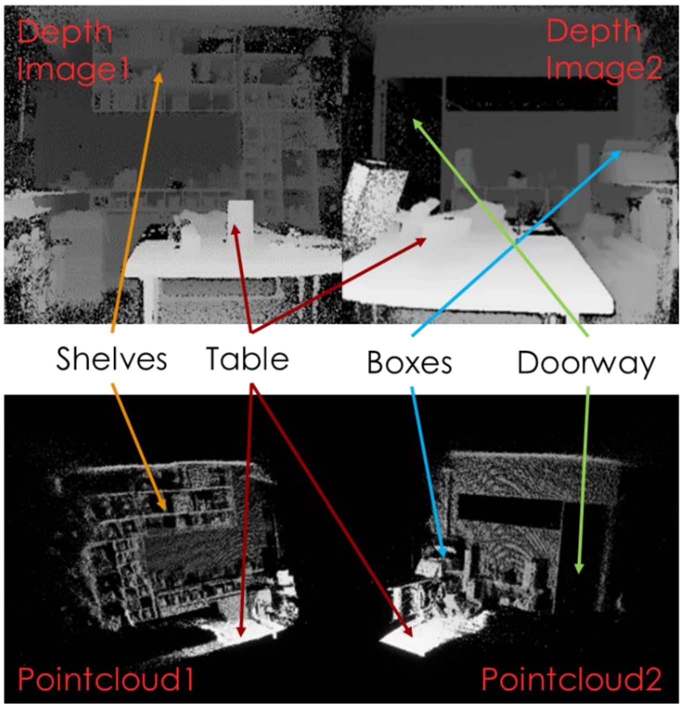

Fig.7 Two depth images and corresponding pointclouds

We have implemented a control system to calibrate the pointclouds to recover a 3D scene. When calibrating the two pointclouds, we fix one pointcloud and move or rotate the other with either keyboard or VR controller until both pointclouds match each other in their shared sections. Fig.8 shows a 3D scene reconstructed from two pointclouds. In this case, we add color to our pointclouds to better interpret location relationships between points in the pointclouds.

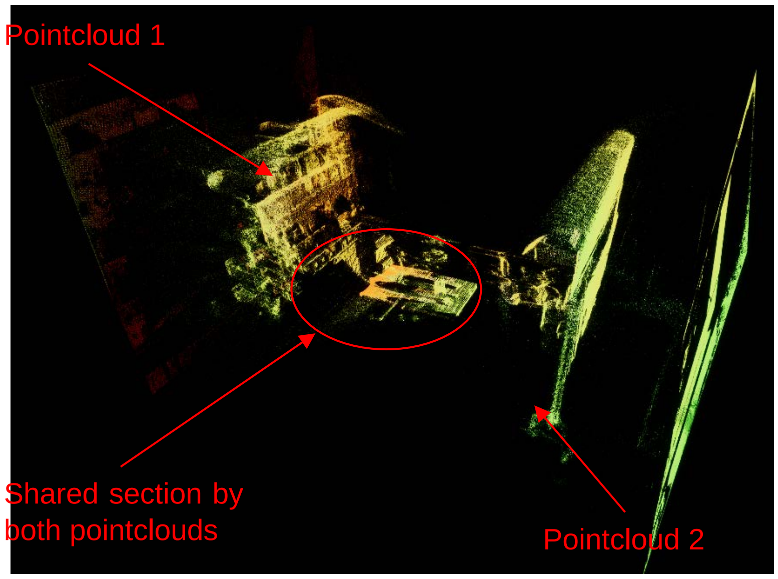

Fig.8 Result 3D scene after calibrating pointclouds

Two depth cameras are used to capture depth images for our pointcloud reconstruction, which introduces redundancy to the imaging system. For instance, if we block one of the cameras, the target scene can still be recovered by the remaining pointcloud, and provide sufficient details for users to interact with objects in the scene.

We have separated the image capturing side and the pointcloud rendering side in two areas. By wearing a VR headset, the users will be a character inside the gaming engine, and can navigate the reconstructed scene freely while they don't necessarily be in the real place in person. This indicates a possibility for remote operation with our imaging system.

We sample and reconstruct the target in a gaming engine, which enables us to have multiple viewpoint to the scene and multi-player access at the same time by applying simple configurations in the gaming engine. For instance, we can move or rotate a separate pointcloud or the whole scene with the control system to view details from different angle. We can also customize the location of our single or multiple users inside the scene so that the users don't need to walk too far to navigate the whole scene.

We have set up a working imaging system that captures a target scene and reconstructs a 3D pointcloud of the scene, and a local network for communication between capturing and rendering sides in the system. This indicates that by installing depth cameras inside a hot cell, our system can recover the interior of the hot cell for users to explore and navigate. Similar to the current camera system for a hot cell, our imaging system provides sufficient redundancy to maintain stability. However, our system improves usability by introducing a control system to adjust camera view and camera location in the gaming engine, in this case multiple viewpoint of the interior of a hot cell. Moreover, by combining our imaging system and remote-control system for manipulators in a hot cell, our users can get access to and operate a hot cell anywhere.

4. Conclusion and Future Works

In conclusion, we have implemented a real-time imaging system which captures and processes depth images and constructs a complete 3D scene of a target scene, and the system has achieved the following goals:

- Provides intuitive experience within the field of view
- Extends the field of view with multiple viewpoints
- Introduces sufficient redundancy to improve system stability
- Is reliable for long-time operation
- Has a low cost for both hardware and software

Besides the above achieved goals, we also gain some extra values from our implementation:

- Decoupling user and target scene locations by separating the image capturing side and pointcloud rendering side, which means user doesn't necessarily be in the same place as the target scene. This shows the possibility of full remote control with the combination of remote control system (e.g. manipulator control) and remote vision system (our imaging system).
- Decoupling viewpoints of user and locations of cameras and windows which can simplify reconfiguration process of hot cells and enables user to change viewpoint arbitrarily and navigate freely inside the constructed scene.
- Mixing virtual reality and real world which can be used for realistic training and demonstration as well as information overlaying such as HMI information.

In future development, better hardware can be used and better methods can be implemented (such as data compression algorithm for depth data) to improve performance of the imaging system.

Acknowledgement

The authors would like to acknowledge Mr. Liam Stephen and other colleagues in S1T2 for kindly being our technical support and sharing fresh ideas as well as possibilities. We would also like to thank Nvidia for the GPU grant program [2], from which our system gains solid performance.

References

[1] Unreal Engine, *Textures* [Online].
Available: https://docs.unrealengine.com/latest/INT/Engine/Content/Types/Textures/.
Last accessed: 4:48PM, 26th Sept 2017.

[2] Nvidia, *GPU Grant Program* [Online].
Available: https://developer.nvidia.com/academic_gpu_seeding. Last accessed: 4:49PM, 26th Sept 2017.