Dynamic Occlusion Using Fixed Infrastructure for Augmented Reality

Scott Johnson, John Baker, Jaime Cisneros, and Juan Castillo
Chosen Realities LLC
Orlando, FL
scott.johnson@chosenrealities.com
john.baker@chosenrealities.com
jaime.cisneros@chosenrealities.com
juan.castillo@chosenrealities.com

Pat Garrity
U.S. Army Research Laboratory-Human Research and Engineering Directorate, Advanced Simulation Technology Division (ARL-HRED-ATSD)
Orlando, FL
patrick.j.garrity4.civ@mail.mil

ABSTRACT

The U.S. Army Research Laboratory-Human Research and Engineering Directorate, Advanced Simulation Technology Division (ARL-HRED-ATSD) performs research and development in the field of augmented/mixed reality training technology. Within training, the Department of Defense (DoD) has a strong interest in augmented reality (AR) for its ability to combine live and virtual assets to reduce cost, increase safety, and to mitigate unavailability of needed live assets. Answering this is the commercial sector, which is rapidly advancing a host of capabilities to support AR, such as head mounted displays (HMDs). An important capability of AR is the realistic occlusion between live and virtual objects based on their respective depth in the augmented reality scene. Existing solutions use pre-scanned environmental depth information to provide this capability; however, this is only useful for objects that will never move or *static objects*. This does not address live objects that move or *dynamic objects* (e.g., live Soldiers). Dynamic objects must be constantly scanned using a depth camera(s) to provide the occlusion information for an AR-enabled system. Of the few commercial sector vendors that provide depth cameras on their HMD, most are lacking the sufficient depth range to adequately support occlusion from the HMD – anything beyond roughly two meters has greatly diminished fidelity. This paper describes a solution that will add a network of fixed infrastructure cameras with a centralized occlusion server to merge the depth images from various sources. This then creates depth images suitable for occlusion on the HMDs at any range with realistic fidelity. The paper will report the use of commercial off the shelf (COTS) computers and cameras that instrument an area such that it can be used for occlusion in a training system. The paper will speak to the performance of fusing data, given resolution and volume. Finally, this paper will show scenes where virtual objects are added to a real scene and how the performance of the occlusion system affects the quality of the visuals for the participants.

ABOUT THE AUTHORS

Scott Johnson is the Chief Technology Officer at Chosen Realities LLC. He shared the 2012 Modelling and Simulation Award from PEO STRI for his work as Lead Software Engineer on the Dismounted Soldier Training System (DSTS). Scott has 12 years of experience in defense and 10 years of experience in the video game industry where he worked primarily in the areas of Animation and Physics. He earned his Bachelors in Electrical Engineering from Purdue University and his Masters in Computer Science and Engineering from the University of Michigan.

John Baker is the Managing Director and Chief Operating Officer for Chosen Realities LLC. He shared the 2012 Modelling and Simulation Award from the Army for his work as Chief Engineer on the Dismounted Soldier Training System (DSTS). John has 20 years of experience in defense, building a wide variety of systems in various capacities. He earned his Bachelors in Industrial Engineering from the University of Central Florida and his Masters in Industrial Engineering from the Pennsylvania State University.

Jaime Cisneros is a Principal Software Engineer at Chosen Realities LLC. He has 24 years of experience in constructive and virtual simulation and training. He was the simulation lead for the Dismounted Soldier Training System (DSTS). As a researcher, Mr. Cisneros has designed and developed advance prototypes for a next generation virtual reality dismounted soldier trainer and has published nearly a dozen papers in this field. He earned a Bachelors and Masters in Computer Science from the University of Central Florida.

Juan Castillo is a multi-discipline engineer and 3D artist at Chosen Realities LLC. He has 8 years of experience in virtual reality, gaming, and graphics industry. He earned his Bachelors in Computer Animation from Fullsail University.

Pat Garrity is a Chief Engineer at the U.S. Army Research Laboratory-Human Research and Engineering Directorate, Advanced Training and Simulation Division (ARL-HRED-ATSD). He currently works in Dismounted Soldier Simulation Technologies conducting research and development in the area of dismounted soldier training and simulation where he is the Army's Science and Technology Manager for the Augmented Reality for Training Science and Technology Objective (STO). His current interests include Human-In-The-Loop (HITL) networked simulators, virtual and augmented reality, and immersive dismounted training applications. He earned his B.S. in Computer Engineering from the University of South Florida in 1985 and his M.S. in Simulation Systems from the University of Central Florida in 1994.

Dynamic Occlusion Using Fixed Infrastructure for Augmented Reality

Scott Johnson, John Baker, Jaime Cisneros, and Juan Castillo
Chosen Realities LLC
Orlando, FL
scott.johnson@chosenrealities.com
john.baker@chosenrealities.com
jaime.cisneros@chosenrealities.com
juan.castillo@chosenrealities.com

Pat Garrity
U.S. Army Research Laboratory-Human Research and Engineering Directorate, Advanced Simulation Technology Division (ARL-HRED-ATSD)
Orlando, FL
patrick.j.garrity4.civ@mail.mil

PROBLEM

Occluding entities in an Augmented Reality system continues to be a difficult problem that is not being solved. When users see into an augmented scene, the depth of the nearest surface needs to be shown and the surfaces behind the closest one need to be hidden. The problem divides into two categories of occlusion. Static occlusion is for surfaces such as walls that remain constant with time. Dynamic occlusion is for moving entities such as other simulation participants, movable props in the room or doors that can open and shut. Static occlusion can be solved by prescanning the environment with a sensor that can provide depth such as a laser or a depth camera. Dynamic occlusion is a more open problem that is not being addressed well. In fact, a survey of the recent commercial Helmet Mounted Displays (HMDs) reveals that the manufacturers are struggling to provide visual field of view without lag rather than dynamic occlusion. As for what exists today, the Microsoft Hololens is already available in development kits and is always scanning it environment. We measure its scanning update rate at approximately five seconds so it could use some additional scanning sources in order to be used for training purposes.

Figure 1 below shows examples of occlusion. The leftmost frame shows a virtual OPFOR entity in a real scene. The next frame shows the same OPFOR hologram viewed such that he is statically occluded by the wall. Right of that shows a real person in front of the virtual entity and the dynamic occlusion is performed such that the real person occludes the virtual entity. Viewers think this happens automatically so the rightmost panel shows what happens to the scene when dynamic occlusion is not accounted for. If the rightmost panel is the first person view of a training participant then it is clear that dynamic occlusion is important for training.

Figure 1. Examples of Occlusion. Leftmost is a hologram of an OPFOR. Next the hologram is statically occluded by the wall. Right of that is a scene that demonstrates proper dynamic occlusion. The final scene is what happens when there is no dynamic occlusion.

The natural question is that if the HMD vendors are not going to solve the dynamic occlusion problem (at least in the upcoming generation of HMDs) then what can be done externally to the HMDs to provide the necessary scanning depth data for them? The occlusion problem can be solved external to the HMDs if external cameras can provide an approximate depth view from each HMD's point of view. The depth image created for each HMD can be rendered using a graphics engine such that when the virtual entities are drawn they will be occluded by the real entities in the depth image.

TERMINOLOGY

This paper and the system it describes deal heavily with images where each pixel represents the distance in meters along a ray from the camera center through the image plane and to a surface in the environment. This type of image can be called a depth image. When these pixels are joined with regular RGB data they are often called RGBD images. If certain intrinsic properties of the camera are known, it is easy to calculate a 3D location in camera space for each pixel. The collection of 3D data of all the pixels is called a point cloud. Depth images can be rendered by a graphics engine into a Z buffer to provide occlusion. When a depth image is used in this way, we refer to it as an occlusion mask. Depth images, point clouds, and occlusion masks are all slight variations of the same data.

SYSTEM DESIGN

The problem of providing depth information from multiple sources can be broken down into some major components (Figure 2). In the Depth Image Acquisition stage, the depth images are acquired from multiple cameras. Multiple cameras can be attached to the server as well as cameras that are connected over a network. In the Depth Integration stage, depth images are converted to point clouds and merged into voxels so that the cameras contribute their view to the overall understanding of the instrumented space. In Occlusion Mask Extraction, an occlusion mask must be extracted out from the unified voxels to each HMD based on the HMD's current location and orientation in the scene. In Occlusion Mask Rendering the extracted depth data will be rendered into the HMDs augmented scene so that it can occlude the scene. This step is handled by the Unity graphics engine, which has become a standard for the HMDs. We will visit each sub problem and see how each was approached in our system.

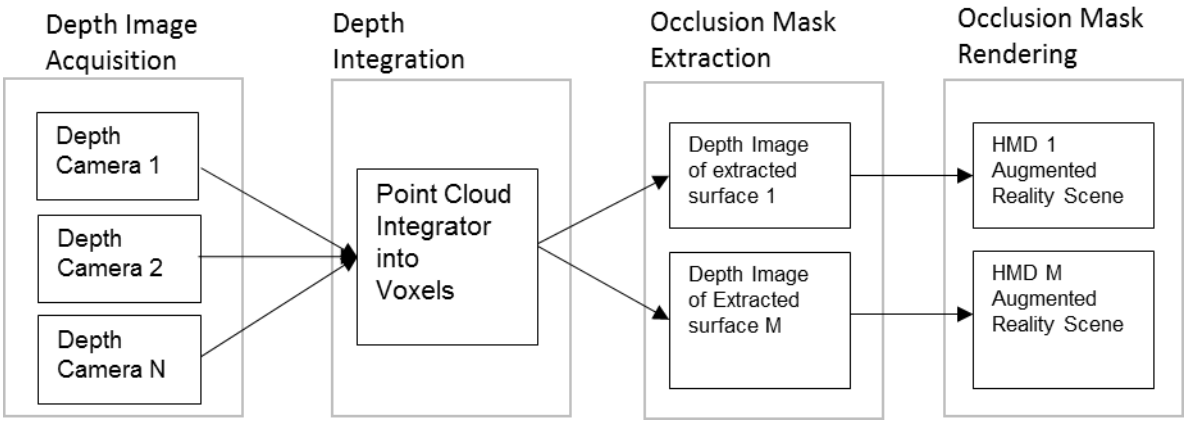

Figure 2. System Design showing the major Components

Depth Image Acquisition

We are creating a heterogeneous system that can use multiple brands of cameras simultaneously. In this paper, we are examining the Microsoft Kinect V1 and V2, and the Stereolabs ZED.

The Microsoft Kinects are an obvious choice because they are cheaply available ($50) and their precision and accuracy have been well documented. The Kinect V1 has a stated operating range of 0.5 m to 5 meters. Its precision is inversely proportional to the square of the distance (Khoshelam, 2012). The precision ranges from 2mm at 0.5m to 18cm at its maximum distance (Precision, 2017). It works by projecting a known light pattern into the real scene. Its stereoscopic cameras then read in the light points and compare the points with their expected locations in the image. Based on the light point's displacement in the acquired images with the actual pattern it is able to determine the depth at the point. The problem with it in a multiple camera system is the projected light patterns interfere with each other. A Kinect V1 expects only to see its own light pattern in the scene and gets confused if the pattern from another Kinect V1 is visible. Its successor, the Kinect V2, uses a time of flight scheme to measure depth so it does not suffer from interference problems. The Kinect V2 has a precision curve that is very

close to the Kinect V1 (Wasenmuller, 2016). Both Kinects function by emitting infrared light and measure it again from the environment thus making them susceptible to bright sunlight. We only use them indoors.

The Stereolabs ZED (www.stereolabs.com//zed/specs) is a relative newcomer. It uses stereoscopic RGB cameras to acquire the depth of a scene much like human vision sees depth. That means that there is no problem with interference between ZEDs. It is priced at $450 but it must be attached to a computer via USB3 that has an NVidia graphics card with CUDA. The ZED acquires depth at various framerates and resolutions that vary from 672x376@100hz to full 2208x1242@15Hz. The published depth range is 0.5m to 20m. The published depth range is not enough information and we need to know the precision at the depth range to make sure it is suitable for our purpose. We wrote an application to determine the standard deviation of the depth at each pixel and created graphs much like the existing Kinect data.

Figure 3 reveals the precision of the ZED camera for an indoor and an outdoor scene. The indoor scene is important because the walls indoors tend to be plain and the stereoscopic algorithm for determining depth can struggle to find features in the scene to determine the depth. In contrast to the clean second order polynomial characteristic of the Kinect V1, the ZED precision does not fit a simple model. The right image in Figure 3 shows the precision of the ZED out to its published range of 20m for an outdoor scene. Again it does not fit a simple model but hints at an exponential curve until 15m out. We confirmed our data with Stereolabs' support staff.

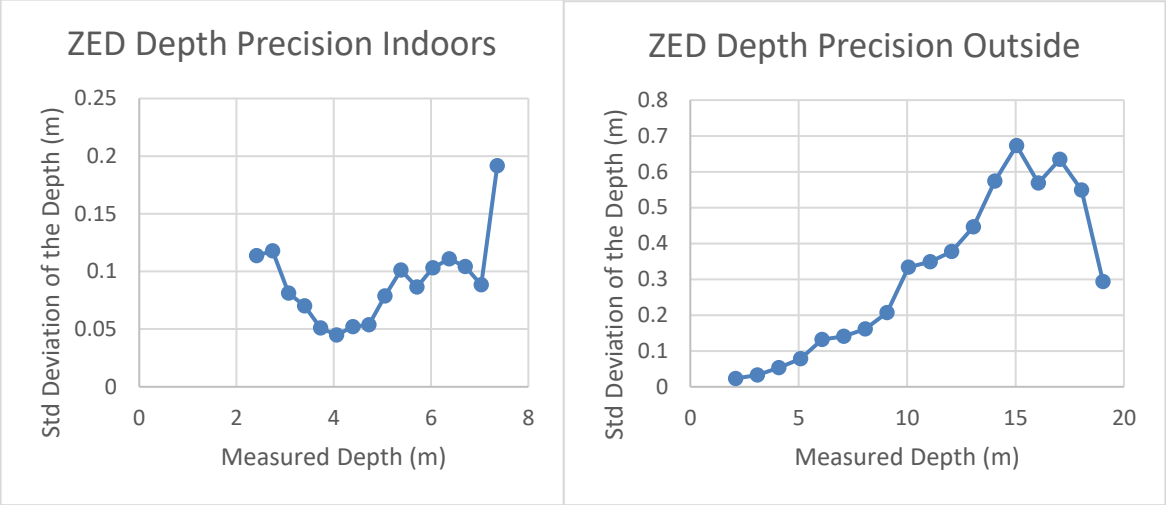

Figure 3. ZED Depth Precision captured Indoors (left) and Outdoors (right)

The lesson from studying these three cameras is that the published information from the manufacturers is not enough. The performance of the camera at each range is important for determining how well the depths will fuse together for the goals of this project. The ZED was supposed to be a critical component in our system but with late gathered data about its performance it may be left as the best camera to use outside where the light is bright and there are many corresponding visual points to produce a stereoscopic depth. The Kinects perform well indoors away from the sun but at the cost of a narrower field of view. We have yet to find how well the noise from one camera can be compensated by the view of another camera so we are keeping them all under consideration.

Another key observation about the depth cameras is that they are all very noisy. This is difficult to see when the data is a depth mask but very apparent when the data is viewed as a point cloud. The points in the point cloud move along their ray from the camera center quickly with an error expressed by plots such as Figure 3. Filtering that data into a smooth surface is one of the major tasks of the system.

Extrinsics Calibration

Another key sub problem of any system with multiple cameras is measuring accurately where the various cameras are in real 3D space and which way they are pointed. This is called extrinsic calibration. We leveraged the results

of an ongoing research project called OpenPTrack (http://openptrack.org). Its extrinsic calibration is detailed in (Basso, 2016) The OpenPTrack project solved this problem by extending the classic checkerboard calibration procedure (Zhang, 1999). A large checkerboard pattern is presented to groups of cameras at a time. Images from the cameras are all taken simultaneously and those that see the checkerboard report the 2D locations of the checkerboard corners to the master computer for processing. The master computer has an implementation of the Google Ceres (Agarwal) nonlinear optimizer. Ceres adjusts the pose of each camera and optimizes it against the measured locations of the checkerboard corners. It does so as a group so that the result is optimized across all the cameras and all the measurements simultaneously. The final answer is the location and orientation of each of the cameras in the system. The process is extendable to any number of infrastructure cameras.

OpenPTrack leverages the Robot Operating System (ROS) in Ubuntu Linux for much of its functionality. ROS provides the functionality of a multicomputer system with all the polished tools that users need to manage a distributed camera system. ROS manages a set of distributed processes as a set of local and remote nodes. The nodes can be run or stopped from a single master computer. The nodes can easily publish and subscribe to data from each other. ROS includes a visualizer that can show the feeds from different depth cameras and show their point cloud data all in the same space. ROS is so widely used that the camera vendors all support ROS so that their camera plug and plays with the rest of the system. It is a massive cooperative environment between researchers and it is all entirely free and open source. OpenPTrack and ROS were a clear starting place for the work to create a heterogeneous system based on infrastructure cameras. But our trouble with software dependencies later forced us to abandon using them directly.

Depth Integration

Once we know where the cameras are in space and we know how precise the depth data is then we can start to merge the depth data from various cameras into a single source. We initially guessed that we would be merging point clouds into voxels and set out to solve the problem of how to simultaneously filter the incoming depth data into a smooth surface and combine the data from multiple sources.

It turns out that there is thirty years of research of using point clouds to create geometry (Niessner 2013). The research we found was mostly for how to scan a static scene using a depth source such as a laser scanner. Then the point cloud data is reduced into a clean set of polygons. There were many approaches but one area where many papers agree is that the problem of smoothing the incoming range data into a surface was solved by Curless and Levoy (Curless 1996). The paper shows how a point cloud can become a set of voxels and then how later point clouds can update the accuracy of those voxels even with noisy depth images. A key fact is that the sensors tend to be accurate when they have a direct view of a measured surface. Their fusion algorithm gives a voxel greater weight to a sensor whose view is more direct. The heart of the algorithm is that each voxel maintains a signed distance function to its nearest surface. A surface can be extracted from the voxels by finding where the signed distance functions cross zero between the voxels.

Since the merging of depth data into voxels is solved, how the voxels can be managed efficiently remains. The problem is that the number of voxels is proportional to the volume of the measured space. If the voxel size is 1cm, there are already 100x100x100=1 million voxels in one cubic meter. The algorithm for finding which voxels contribute to a measured depth surface is very important, as is the algorithm for extracting an occlusion mask from the voxels to feed to the HMDs. VoxelHash (Niessner, 2013) is an application that shows that an efficient way to use voxels for representing depth surfaces is through hash tables. Each voxel is represented by an entry in a hash table rather than a node in an octree. It heavily uses NVidia CUDA to process depth into voxels on the GPU and then to use a raycast renderer to extract depth images out from a particular view. VoxelHash was written to show how a single moving depth camera can quickly and accurately scan a large static scene. It validates the authors' notion that a hash table can represent voxels efficiently in parallel on the GPU.

Occlusion Mask Extraction

We assumed for a long time that we would be extracting polygons from the voxels for each HMD and then the HMDs would be rendering the polygons into their Z buffers for occlusion. But VoxelHash instead implements a raycast renderer of the voxels to extract the nearest surfaces from the point of view of an HMD. It is important to

note that it is not the voxels themselves that are rendered. It is the zero crossing of the signed distance functions that are inside the voxels that represent the surface.

In detail, VoxelHash casts a ray through each pixel from an HMD's point of view. It traverses the 3D space from the camera along the ray and stops when the signed distance function of voxels along the ray cross zero. It has to enumerate the voxels along the ray's path all the way from the camera to the first surface it encounters. This will be important when we discuss performance of the system.

DESIGN

So now the major sub problems are solved and a software design for the system takes shape. Our idea was to extend VoxelHash to acquire depth from multiple static cameras instead of one moving camera. We would perform the extrinsic calibration to get the location of each camera so that the depth from each camera can be transformed into the unified space of VoxelHash. VoxelHash would not know the difference between its single moving camera and multiple fixed cameras. We would then have each HMD in the scene report its own location and orientation to VoxelHash so that VoxelHash could extract an occlusion mask from the HMD's point of view and then send it over the network to the HMD for use into graphics engine as a depth mask in the HMD.

PRACTICAL CONSIDERATIONS

The problem with the idea is that the pieces exist across a large suite of dependencies. Half of the functionality of our potential system lay in Ubuntu Linux using ROS. The other half was in a Windows only application in VoxelHash and the upcoming HMDs we were interested in were Windows only. We made a dependency graph to track the software supply chain that represented twenty five of our biggest components. Practically speaking there is no Windows version of ROS and making a bridge would be full of lag. In the end we could not integrate the Stereolabs ZED into our ROS system based on Ubuntu 14.04 because the Stereolabs driver had dependencies that could not be resolved. Stereolabs said we needed to upgrade our entire system to Ubuntu 16.04. That broke the stalemate and we decided to go to Windows. The result was that we had to bring over the extrinsic calibration from OpenPTrack and Google Ceres to Windows.

CURRENT IMPLEMENTATION

We have a monolithic Windows application that implements data acquisition from the cameras, merging of point clouds, extraction of the depth images, and extrinsic camera calibration. The system today is being built back up after deciding to make the system based on Windows instead of ROS and Ubuntu. Two major pieces have yet to be replaced. The first is the network transportation of the depth images over the network. The ROS implementation quantizes the floats from the depth images and then PNG encodes them before sending them over the network. We are working on alternatives in Windows given that the HMDs themselves (such as the Hololens) need to decode the images in real time and they have mobile phone performance. The second missing piece is that without network transportation of the RGB and depth images we are currently limited to two cameras connected to a single PC. These restrictions will be resolved by the time of IITSEC 2017. Still at this point there are lots of performance metrics that are known.

Extrinsic Calibration Precision

The performance of the extrinsic calibration is easily visualized by showing the alignment of two point clouds. The system was configured with a Kinect V1 and a Stereolabs ZED camera arranged 90 degrees apart and the person stood two meters away from both cameras. Only with a proper extrinsic calibration will the point clouds transform into the same space and contribute information to a unified scene. Figure 4 shows portions of point clouds from a Microsoft Kinect V1 (left), a Stereolabs ZED (center) and the two of point clouds in overlapping in the same space. In the rightmost image, the points from the Kinect form the left side of the person and the points from the ZED form the right side of the person.

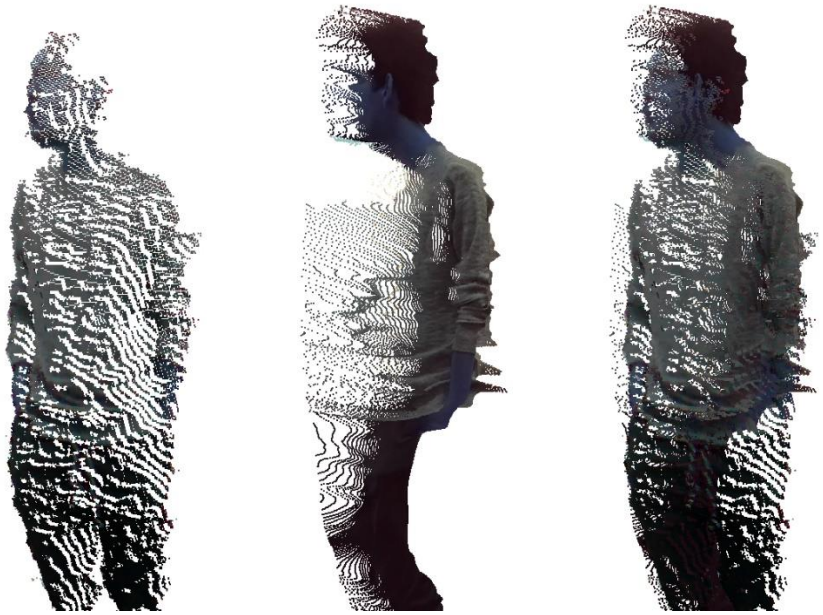

Figure 4. Point Clouds from two cameras shown in the same space. Leftmost is a portion of a point cloud from a Kinect V1. Center is a point cloud portion from a Stereolabs ZED. The rightmost image shows the two point clouds in the same space using the results from the extrinsic camera calibration.

Performance (Speed)

It takes about 1ms to Gaussian filter the incoming depth images before integrating them into voxels. The integration into voxels takes about 1ms per camera. The vast majority of the time is spent on the raycast renderer that renders the voxels into an occlusion mask. This means that we can add fixed depth cameras to the system and each one will only add about 1ms of processing time. Raycast rendering emerged as the clear bottleneck in the system and will dictate how many HMD clients can be supported and at what framerate.

Recall that the raycast renderer follows the path of each ray from the HMD point through each pixel in its view and into the scene. The ray is traversed in 3D and the voxel hash table is consulted along the way to determine if there is an existing voxel there or if one needs to be created there. The amount of traversed space can be enormous in one view and the performance of the raycast renderer is proportional to the amount of space in the view. This was painfully obvious when we implemented background removal in the depth images prior to integrating them into voxels. It was thought that the performance would increase when there were far fewer voxels. If the only part of the depth image we kept was for the foreground dynamic scene then there would be far fewer voxels. However it meant that each ray had to be traversed to its maximum depth and the raycast rendering time doubled. If we could somehow limit the distance along each ray that the raycast would traverse we could speed up the raycast renderer. Perhaps we could render the voxels as sprites to a near Z buffer to determine the starting distance for each ray and then render them to a far Z buffer to determine the ending distance for each ray. As this idea solidified we found that the idea had already been implemented and mysteriously commented out by VoxelHash's authors. We put it back in because it makes sense and it improves the raycast render times.

Next we profiled the raycast renderer in NVidia NSight to see if it could be sped up. The profile revealed that the GPU threads are starved waiting on reading memory. That makes sense because all the threads on the GPU are trying to access the voxel hash table at the same time. That is fundamental to the algorithm. Speedups would have to come from improving hardware and NVidia had recently shipped its 10 series graphics cards.

Table 1 shows performance metrics on the slowest process in the system, the raycast rendering of the voxels into a depth image. The voxels are set to be 2cm on a side. The depth images coming from the cameras are down sampled to 640x480 and the final raycast rendering from the voxels to a usable depth image is also done at 640x480.

Table 1. Performance of the RayCast renderer which is the bottleneck in the system. The RayCast renderer renders the voxels into a depth mask that can be used by a graphics engine for occlusion.

NVidia Card	Raycast Rendering Time (ms)
GeForce GTX 965M	25ms
GeForce GTX 1070	15ms
GeForce GTX 1080	10ms

So at 10ms the system could support three clients at 30Hz (at this resolution). Our goal is to eventually support a squad of nine soldiers using HMDs at better resolution.

What may prove to be a critical factor is that the current augmented reality HMDs have very narrow fields of view. The Microsoft Hololens field of view is reportedly 30x17 degrees (http://doc-ok.org/?p=1223). With such a small visual area the occlusion server can reduce the resolution of the depth image used for occlusion and support a greater number of HMD clients.

Performance (Instrumentable Area)

Even with the two camera setup that we currently have we can instrument a large volume of space. The Stereolabs ZED provides 120 degree field of view. The scene in Figure 5 shows the extents of the area with just a Kinect and a ZED. Full 360 view can therefore be captured with just three cameras per room. Additional cameras provide coverage to reduce shadowing. The system scales very well with the number of cameras because the depth data from the cameras integrates into the voxels at a rate of about 1ms per camera.

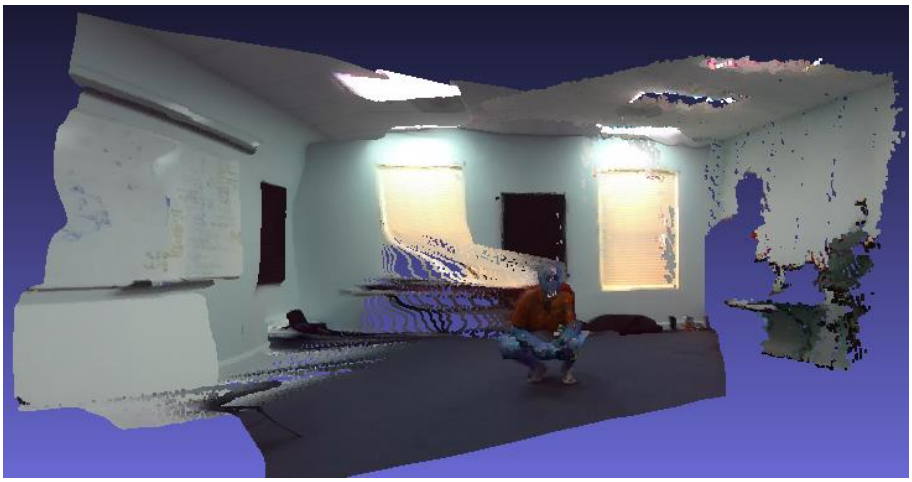

Figure 5. Capture Area with a just two Cameras : Kinect and a Stereolabs ZED. The ghosting artifact is from the Stereolabs ZED.

Dynamic Occlusion Results

Figure 6 is an example of the dynamic occlusion being provided by an extracted depth image. The virtual entity is deeper in the scene than the real person so the real person blocks out the pixels of the virtual entity. There is some artifacting where the person overlaps the virtual entity just under the rifle. That comes from the resolution of the extracted depth image which is currently 640x480 due to the speed of the raycast renderer. Surprisingly, it is not the size of and number of the voxels that limits the visuals. The system performs at the same speed and yields the same visual fidelity when we decrease the voxel size to 1cm. It takes more GPU memory to store more voxels but so far that has not been a problem with the worst card we have which is the GeForce 965M with 2GB of RAM.

Figure 6. Sample dynamic occlusion created by our system

CONCLUSION

We have detailed the development of a dynamic occlusion system using multiple camera types as input. We showed how to evaluate the cameras. We showed how the pieces of the problem were solved by modifying existing software components and how the practical considerations affected the system. We then revealed that the key bottleneck affecting performance is the raycast renderer and not the number of voxels. The raycast renderer sets the resolution that the depth image can be extracted and so it dictates the visual quality of the seams where the virtual and real entities overlap.

FUTURE WORK

The biggest weakness in the system as it exists today is that when people move in the scene it takes too long for the pixels in the rendered occlusion mask to update. This creates a ghosting affect such that people leave trails behind them when they move and it takes too long for the pixels to react when a person steps into them. The main factors that affect the persistence are the update rate of the system and the Curless and Levoy voxel weights. The update rate is again set by the raycast renderer. The Curless and Levoy voxel weights dictate how much weight each incoming point cloud contributes to the final voxel weight compared to the amount that has already accumulated there. There is a tradeoff that exists in all filtering where if you allow greater latency the filtering can be done over a larger number of samples. We are going to tune the system for less pixel persistence and more responsiveness.

REFERENCES

Agarwal S., & Mierle K. Ceres Solver from http://ceres-solver.org

Basso F., Levorato R., Munaro M., & Menegatti E. (2016) A distributed calibration algorithm for color and range camera networks. *Robot Operating System – Studies in Systems, Decision and Control*. Springer

Curless, B., & Levoy M. (1996). A volumetric method for building complex models from range images. In *Proc. Computer Graphics and Interactive Techniques*, ACM 303-312

Khoshelham, K., & Elberink, S. O. (2012). Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors (Basel, Switzerland)*, *12*(2), 1437–1454. http://doi.org/10.3390/s120201437

Niessner M., Zollhofer M., Shaharam I., & Stamminger M., (2013) Real-Time 3D reconstruction at scale using voxel hashing, *ACT Transactions of Graphics (TOG)*, v.32 n.6 November 2013

Precision of the Kinect Sensor. (n.d.) In wiki.ros.org. Retrieved June 12, 2017 from http://wiki.ros.org/openni_kinect/kinect_accuracy

Wasenmüller O., & Stricker D. (2017) Comparison of Kinect V1 and V2 depth images in terms of accuracy and precision. In: Chen CS., Lu J., Ma KK. (eds) Computer Vision – ACCV 2016 Workshops. *ACCV 2016. Lecture Notes in Computer Science*, vol 10117. Springer, Cham

Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientation. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, 1999, pp. 666-673 vol.1.