Scene++: Making AR Immersive

Authors: **Zhengyi Luo, Ziad Ben Hadj-Alouane, Henry Zhu, Liam Dugan**

University of Pennsylvania School of Engineering and Applied Sciences, CIS Department
zhengyil@seas.upenn.edu, ziadb@seas.upenn.edu, henryzhu@seas.upenn.edu, ldugan@seas.upenn.edu

Advisors: **Professor Stephen Lane, Professor Ani Nenkova**

Abstract

Current Augmented Reality (AR) gaming setups require a depth-sensing camera attached to a head-mounted-display (HMD) such as the Oculus Rift. This provides the developers with primitive surface detection features (e.g. floors), but not the ability to detect interesting real-world objects (eg. chairs). AR games are consequently restricted to displaying visual content on planes, limiting player immersion. We propose a simple, cross-platform framework (Scene++) that enables developers to utilize real-world objects in their games.

Using Scene++, the developer can query the location of real-world objects relative to the viewer ("cup is x meters away") and their pose ("cup is upside-down"). To accomplish this, we use a cloud-based architecture that analyzes images from the camera with cutting-edge deep-learning algorithms (Mask-RCNN [3], DenseFusion [4]). We then use the depth-sensing camera to better localize the objects. Finally, we demonstrate the capabilities of our framework by attaching a game onto an object (overlaying a virtual dartboard game onto a circular clock).

We found that, by offloading scene understanding computation to the cloud, Scene++ was able to improve the AR development workflow with minimal performance impact — maintaining at least 40 frames per second throughout gameplay. Thus, developers save time and effort by shifting focus to game development.

Motivation and Problem Formulation

Our project is motivated by our desire to create a unique entertainment experience that cannot be accomplished through traditional means (e.g movies, music, outdoors, standard video games, etc...). Specifically, we believe that pushing the bounds of Virtual Reality (VR) and Augmented Reality (AR) can alter the way people interact with each other. As such, one way VR/AR can improve and become more appealing is to create more engaging and customized content.

We observe that AR content is mainly limited by the basic scene understanding features AR platforms provide. To illustrate, most AR platforms only provide wall and floor recognition features. We believe that if developers were easily able to detect and locate interesting objects in the scene (e.g. cup, chair, table, TV, vase, etc.), they would be able to create more engaging content that allows users to interact with their surroundings in a more meaningful way.

As such, we have developed a framework (dubbed Scene++) to provide developers with a tool that:

1) Can be used on multiple HMDs (Oculus, Vive, etc.)
2) Can be used without sacrificing game engine and rendering performance
3) Abstracts away the computer-vision aspects of AR game development by providing developers with usable scene understanding information (location of objects in game world, type of objects, etc.)

As an example application, developers could automatically augment the user environment with fun and entertaining activities, akin to a recreational room. The application would detect plain feature objects (e.g table, circular clock, trash bin, cup, bed, etc.), and recognize that certain "activities" can augment these objects, and further enable the user to start these activities. The application would be able to detect a circular clock on the wall, and augment it with a dart board gaming experience, with which the user can interact with. It could also detect a rectangular table and overlay a ping-pong table on top of it, and have users play.

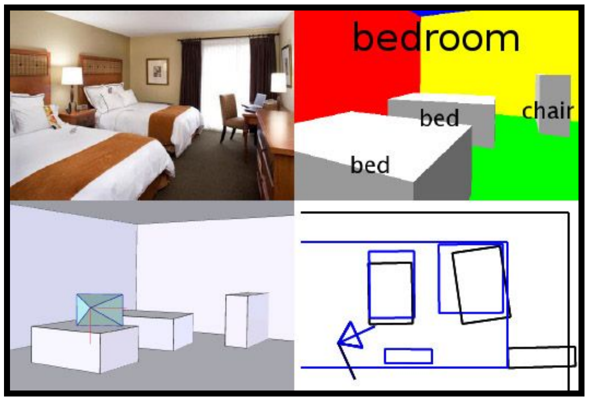

Fig. 0: Detecting Location of Interesting Objects in a Scene

Technical Approach

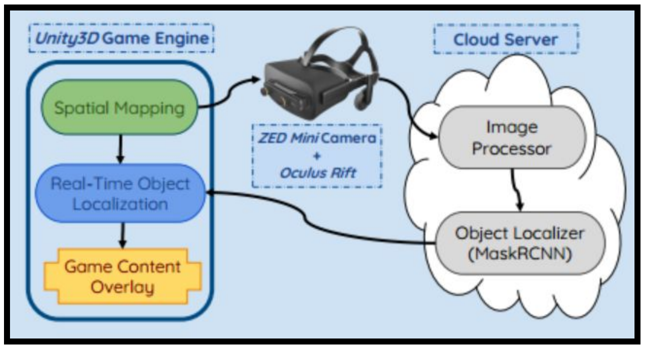
Unity3D Game Engine
Spatial Mapping
Real-Time Object Localization
Game Content Overlay
ZED Mini Camera + Oculus Rift
Cloud Server
Image Processor
Object Localizer (MaskRCNN)

Fig. 1: System Architecture Used to Run Scene++

To make Scene++ perform image processing and object localization component of the framework efficiently and without consuming local computing power, we offloaded computer vision computation to the cloud . Finally, the output of the cloud component is sent via a asynchronous communication protocol to the game-engine.

Cloud Computing Side

To enable us to offload the heavy computation tasks to the cloud, we used the architecture shown in Fig. 2. The architecture consists of two main parts: the client side, which resides on the PC and handles the image data transmission (as well as game simulation), and the server side, which runs on the cloud instance and executes the scene understanding algorithms.

1. **Server Side**: The server side is made of two main components, the Asyncio Server and the Deep Learning Object Detector.
   - Asyncio Server: runs websocket server for communicating with the client. The server receives the incoming data stream from the client side and send it to the deep learning object detector. After the deep learning algorithm is done, the server sends the detection results back to the client side.
   - Deep Learning Object Detector: analyzes the image byte stream and uses Mask-RCNN (object detection algorithm) to recognize objects in the current frame.

2. **Client Side**: The client side is made of four main components, the Camera Capture component, the Local Compression component, Asyncio Client component, and the Action component.
   - Camera Capture: uses camera library to capture image output from the device camera.
   - Local Compression: processes raw image data from camera and compresses it to reduce file size, then encodes the raw image data into a byte stream.
   - Asyncio Client: runs websocket clients and communicates with the server. The client sends the image data to the server continuously and then receives object detection data.
   - Action: reacts to the detected object in the scene.

3. **Pipeline Workflow Overview**: Each image frame captured by the Camera Capture component is compressed by Local Compression component. Then the compressed image is encoded into a byte stream and sent to the server through a websocket connection by the Asyncio Client. After the cloud Asyncio Server receives the image frame, it shares the image with the Deep Learning Object Detector for processing. Afterwards, the cloud Asyncio Server sends the detection results as a string through websocket to the client. The device Asyncio Client receives the object information and sends it to the Action Component for reaction.

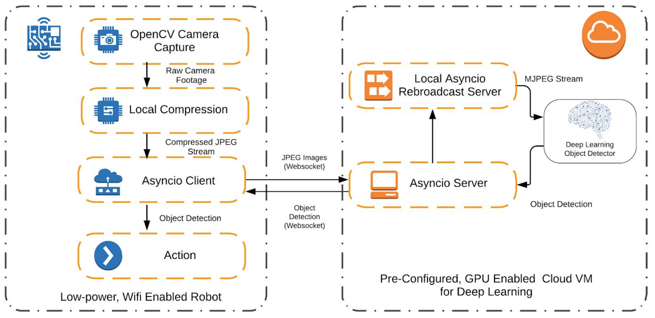

Fig. 2: Cloud Computing Architecture

4. **Communication Protocol**: To enable real-time performance, the communication protocol needs to be low level enough that it introduces minimal latency. Specifically, we chose websocket as our main communication protocol. Another option to ensure the lowest possible network latency is to use raw TCP, but the direct use of TCP has limited speed advantage [5]. Websocket allows multiple connections to a single server, and is easy to use in an existing web framework like the python aiohttp library. Since websocket ensures easy interfacing with the rest of the application, while also providing a more secure interface for the streamed video frames (WSS protocol encrypts streams using HTTPS) it is the best choice for our use case.

5. **Deep Learning Object Detection**: Deep learning based object detection algorithms have gained significant traction due to their rapidly improving performance on some of the most well-known image classification datasets such as ImageNet [1] and COCO [2]. For this work, we mainly focus on adapting the Mask-RCNN object detection algorithm to a cloud computing friendly configuration, which is shown as the Deep Learning Object Detector in Fig. 1. Given an image from the the client side, the following steps are performed on the image to detect objects, as shown in Fig.2.
   - Regional Proposal network: Images are passed through a Convolutional Neural Network (CNN) based regional proposal network to detect region of interest (ROI).
   - Classification Branch: Detected ROIs are passed through a classification branch to classify the type of the objects within the current ROIs. For the fully supported list of objects, see Appendix I.
   - Bounding Box Detection Branch: Detected ROIs are passed through a bounding box detection branch to predict the precise bounding boxes for the current object.
   - Mask Branch: Detected ROIs are also passed through a mask branch to perform the per-pixel classification for the ROIs.

   For more details about the method, please reference the original paper [4].

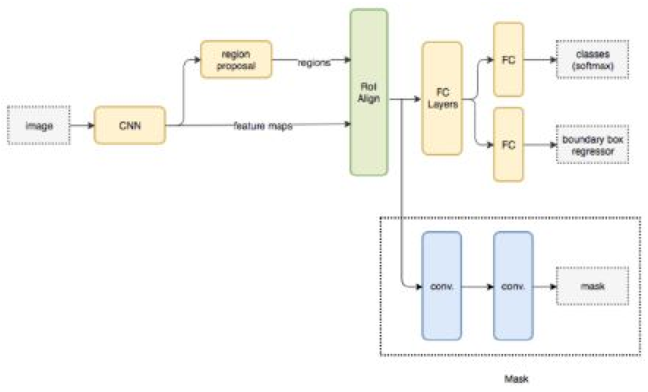

Fig. 3: Mask-RCNN Network Design

Gaming Side

To showcase how Scene++ can be used within a real application, we chose the following game development setup (as seen in Fig. 1):

1) Oculus Rift HMD [5]
2) ZED Mini depth-sensing camera (attached to the HMD) [6]
3) Unity3D [7]

We chose Unity3D as an example game engine for practical reasons: it is free for individual developers, is widely used by independent game development companies, and it interfaces well with popular HMDs (such as the Oculus or the HTC Vive) through readily available plug-ins. The following are the steps we take in our pipeline to run an example application:

1. **AR Display:** using the ZED Mini camera attached to the HMD, we replace the VR display with an AR display by forwarding the ZED Mini image data to the display screens in the HMD. This is facilitated by the ZED-Mini Unity3D plug-in.

2. **Spatial Mapping:** We leverage the ZED Mini's SLAM-based spatial mapping capabilities in order to build a mesh of the surrounding world with the world coordinates as the reference point. We later use this mesh for raycasting intersections that allow us to position virtual objects on the screen.

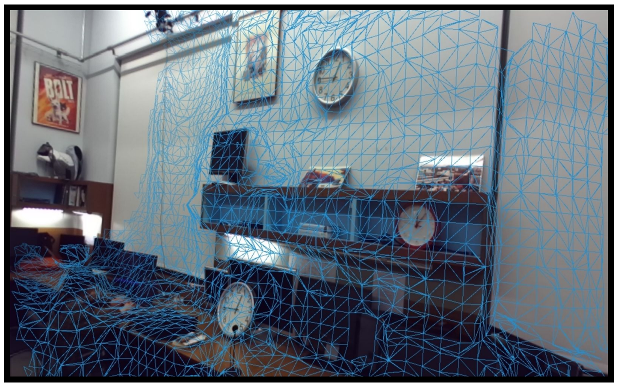
Fig. 4: Spatial Mapping on a Scene with Circular Clocks

3. **Image Data Communication:** after the spatial map is built, we proceed to send the image and depth-map data to the cloud server. This is done with WebSocketSharp[8], a websocket-based communication library for C# (the main scripting language for Unity3D). For each image sent, we save the camera state at the point in time the image is sent so we can tell the point-of-view of the user when we use the data later on.

4. **Object Localization Data Parsing:** upon receiving data from the cloud server, we parse it into multiple objects. Each objects holds: its id/type, the bounding box in screen-space, the confidence of the algorithm as to the object's presence, and a bitmap mask.

5. **Game Object Overlay:** given the parsed data, we proceed to raycast virtual rays into the bounding box of each object. If the ray hits the spatial map, then we proceed to overlay the appropriate game object in world space (e.g. the ray that went through the bounding box of a real circular clock hits a triangle on the mesh that corresponds to a location on the wall). We proceed to overlay a dart board that the user can interact with

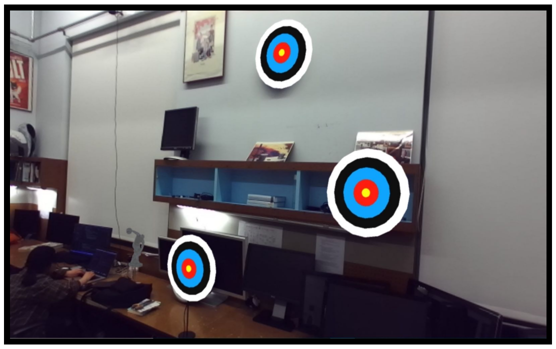
Fig. 5: Dart-boards Overlayed on Top of Detected Circular Clocks

Evaluation

We used the following configurations for our cloud server and our local game-engine client:

1. Cloud instance configuration:
    - Oct Core Intel(R) Xeon(R) CPU E5-2623 v4 @2.60GHz
    - 32 GB RAM
    - Paperspace Network
    - NVIDIA Quadro P4000, 8GB Dedicated Video RAM
    - Ubuntu 16.04.4 LTS
2. Client configuration:
    - Intel(R) Core(™) i7-8740H CPU @2.2GHz
    - NVIDIA GeForce GTX 1060 (Notebook)
    - 16 GB RAM
    - Microsoft Windows 10 Home

We evaluated Scene++ based on the following criteria:

1. **Object Detection Variety:** How many different objects are recognizable and classifiable by our platform? How robust are we to size and shape? How accurate are our labels? How does the network perform on unseen objects?

2. **Cloud Computing Latency:** What is the latency introduced when images are streamed between the local machine and the cloud? How soon after an object comes into frame is it detected?

3. **Spatial Mapping Fluidity:** Once an object is labeled, does it persist? Are the labels reliable over extended periods of play time?

4. **Frames Per Second (FPS):** What is the average fps during our test application? How stable is the framerate? Do people feel nauseous?

Discussion and Findings

Object Detection Variety: Our system can detect up to 80 different objects concurrently and the position is robust to sudden and fast head movements. Full list of supported objects can be seen in see Appendix I. Tags very rarely are disjoint from the item they are trying to label and even then the tag repositioning system works quite well, repositioning tags if the item strays too far from its original location. This also means that our tags accurately stay attached even to moving objects (assuming the velocity is sufficiently slow).

As shown in Fig. 5, most objects in image are correctly labeled. We have observed rare cases of the system assigning erroneous labels to objects but this only tends to happen when that object is both one it cannot detect and one that is featured very prominently in the frame (e.g. a loop of cord when viewed very up close an on a monochrome background was once labeled as a tennis racket). However, upon inspection, the confidence given for these labels was significantly lower than most objects. Thus, if we desire precision, we can sufficiently tune our confidence interval to eliminate almost all instances of mislabeling.

Cloud Computing Latency: To measure the detection latency, we set up a test image which contains only a single person. The time between the image being sent to the server and the object data being received is reported in Table II. To accommodate network latency based on time of day, we measured the latency 6 times in the same day.

	Sunnyvale to San Francisco (41.5 miles)	**Sunnyvale to New York (2937.8 mi)**
Round Trip Time (seconds)	0.07661	0.21073

Table I. Round trip time for data size 13.5k Bytes

Time	**Total Delay**	**Time**	**Total Delay**
8:00 AM	0.562	6:00 PM	0.651
11:00 AM	0.687	9:00 PM	0.756
3:00 PM	0.585	12:00 AM	0.494

Table II. Time between frame of person showing up and receiving detection of person on client side (between Sunnyvale & San Francisco)

On the client side, a manual delay between sending frames was originally inserted so as to avoid the associated client-side lag, but sending the images at one fourth resolution asynchronously reduces that lag to a point where it is not perceptible to the user. This lets us send frames to the server without concern for client-side performance impact.

Spatial Mapping Fluidity: Scene++ is able to recalibrate and readjust the spatial mapping while the application runs, making object repositioning and labeling robust. Without real time recalibration, user interaction would not be enjoyable: there would be certain iterations where the application would try to synchronize the virtual objects with the real objects and the entire application would stop in order to have every object aligned. In addition, between each synchronize operation, the user would notice that certain object positions and labels are gradually becoming misaligned. Real time spatial mapping is critical to making Scene++ a user-friendly experience.

Frames Per Second (FPS): Scene++ is able to maintain a constant 40 or more frames, which means that the user can interact with the system nicely without becoming too dizzy. There are barely any lag or jumps in the visuals as well, making it comfortable for the user to work with.

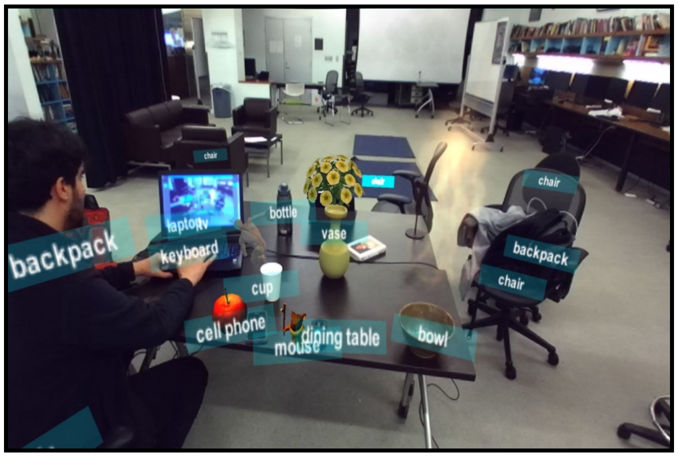

Fig.5: Spatial Mapping on a Scene with Circular Clocks

Major Challenges: We found that the main application bottleneck was The ZED Mini. Specifically, the ZED Mini was designed to be used for low throughput AR applications. In our case, rendering and sending image data to the cloud became costly. Our diagnosis shows that the primary issue resides within the bandwidth of the USB-C attached to the ZED Mini, which does not allow for frequent and efficient image data processing tasks. Although we have not experimented with it ourselves, we recommend using the high-performance ZED Camera instead.

Ethical Considerations and Societal Impact

1. **Ethical Considerations:** While Scene++ does not add significant ethical concerns to the AR industry, it is still exposed to standard problems related to video game violence. With AR, there is an added layer of reality attached to the gaming experience. One possible concern would be whether or not to allow sensitive objects such as weapons, people, and animals to be detected. In our framework, we can easily disable detecting such objects such that it is harder for developers to allow interacting with them.

2. **Societal Impact:** The main societal benefit Scene++ brings is enhanced and immersive AR entertainment

experiences. However, our platform can be extended such that it applies in more practical industrial settings such as aiding human laborers. An example of this would be detecting perished food items in a supermarket, or sorting through raw materials in a factory. With Scene++, this can be done with high accuracies even at low resolutions.

Conclusion and Future Works

Through this project, we have demonstrated the capacity of this technology to be effectively deployed to modern AR/VR consumer devices. With cloud computing, the performance impact of object detection and localization is kept to a minimum, allowing us to effectively run the software on hardware that was not specifically engineered to perform this task. Additionally, the versatility and robustness of our labels over long periods of game interaction demonstrates the capability of the technology to be implemented in a real-world scenario without adversely affecting the end user experience.

This accomplishment of a playable frame rate, despite using only general-purpose hardware, shows that this technology has potential to become widely used in consumer applications, especially on devices with dedicated hardware. This dedicated hardware should include a built-in front facing camera with depth-sensing capability and a low latency method of displaying the frames to the end-user through the HMD. With hardware such as this, the 90 or higher fps baseline for consumer VR applications should be readily achievable.

Once the hardware becomes sufficiently sophisticated, future work should explore the possibility of incorporating features such as pose estimation into the object detection pipeline. Pose detection would allow for much more meaningful game logic. As an example, if we had a scene where a real-world cup was filled with virtual liquid, the game would be able to determine if someone had knocked over the cup and would then spill the in-game water that it contained. This type of immersive real-world interaction would open the door for a variety of novel game concepts that would give AR a much larger selection of unique and memorable content with which to market the burgeoning platform.

References

[1] ImageNet: imagenet is an image database organized according to the wordnet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images.http://www.image-net.org. Accessed: 2019-04-25.

[2] COCO: coco is a large-scale object detection, segmentation, andcaptioning dataset. http://cocodataset.org/. Accessed: 2019-04-25.

[3] He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2017–Octob*, 2980–2988. https://doi.org/10.1109/ICCV.2017.322

[4]Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., & Savarese, S. (2019). DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion. Retrieved from http://arxiv.org/abs/1901.04780

[5]Oculus Rift: Oculus Rift is a virtual reality headset developed and manufactured by Oculus VR, a division of Facebook Inc., released on March 28, 2016. https://www.oculus.com/ Accessed: 2019-04-25.

[6]Zed Mini: Meet ZED Mini, the world's first camera for mixed-reality. https://www.stereolabs.com/zed-mini/. Accessed: 2019-04-25.

[7] Unity3D: Unity is a cross-platform real-time engine developed by Unity Technologies, https://unity.com/. Accessed: 2019-04-25.

[8]WebSocketSharp: WebSocketSharpis a C# implementation of the Websocket protocal, https://github.com/sta/websocket-sharp. Accessed: 2019-04-25.

Appendix

Appendix I: List of Supported Objects:

person	elephant	wine glass	dining table
bicycle	bear	cup	toilet
car	zebra	fork	tv
motorcycle	giraffe	knife	laptop
airplane	backpack	spoon	mouse
bus	umbrella	bowl	remote
train	handbag	banana	keyboard
truck	tie	apple	cell phone
boat	suitcase	sandwich	microwave
traffic light	frisbee	orange	oven
fire hydrant	skis	broccoli	toaster
stop sign	snowboard	carrot	sink
parking meter	sports ball	hot dog	refrigerator
bench	kite	pizza	book
bird	baseball bat	donut	clock
cat	baseball glove	cake	vase
dog	skateboard	chair	scissors
horse	surfboard	couch	teddy bear
sheep	tennis racket	potted plant	hair drier
cow	bottle	bed	toothbrush