Creating a Virtual Reality Experience for Firefighters

Emilio Segovia
University of California, Riverside
esego001@ucr.edu

David Zhang
University of California, Riverside
dzhan008@ucr.edu

***Abstract*— Virtual Reality (VR) has seen many advances through its recent years and it has great potential in simulating real life situations. Firefighting has not received much attention in the realm of VR. Recreating life-like scenarios such as burning buildings for firefighters in training can be costly.**

Our proposition entails developing a VR environment that can replicate various scenarios that firefighters in training must experience. Through the use of the Oculus Rift VR headset and the Unity 3D application, we want to find out if we can create a life-like simulation that can serve as an alternative training method for firefighters and be equally or more effective than current training simulations. In addition, we propose to use the VR environment as a testing grounds for virtual prototypes of innovative firefighting tool concepts. Our report will explain the concepts behind our project as well as how we implemented it.

CSS CONCEPTS
CCS → Human-centered computing → Human computer interaction (HCI) → Interaction paradigms Virtual reality

I. INTRODUCTION

Firefighters need to pass various training simulations before they can begin work in a real emergency. These training programs allow firefighters in training to learn the needed skills to become a firefighter. This includes being able to suppress fires, evacuate buildings, treat victims, and more [1]. To provide the best training experience as possible, programs will have trainees go under life-like situations with controlled fires called ”live fire training”. However, these scenarios can be costly and inefficient since fires consume the training materials [2].

There are virtual training alternatives such as ”Fire Fighting Simulator” which is an application that can teach trainees what its like being a firefighter [3]. However, trainees using this application look at a screen which means that the skills learned may not easily transfer to a real-life emergency. While the scenes inside the simulator can be life-like, it will not feel as realistic to the trainee as a VR experience. No doubt these trainings help but a unrealistic scenario will not properly prepare trainees for the physical stresses and adrenaline associated with a real emergency.

VR applications provide a step up in the immersion factor that we need for creating a firefighting experience. With the use of the Oculus Rift headset, we can place the player into a virtual environment without the need for setting up a costly controlled fires and other equipment. We could utilize the Oculus Touch controllers to track the players hands and allow them to interact with objects in the experience.

Creating the experience in VR can also allow us to prototype possible features and technologies that are not present in firefighting. Currently, firefighters use a forward-looking infrared radiometer (FLIR) thermal imaging device to navigate through fires. Even with handheld thermal imaging devices, which can display the heat signature of humans, there lies the problem of physically managing these devices which takes effort and time that the firefighters dont have in an emergency. Firefighters must also be able to quickly switch their focus between seeing a thermal view and a non-thermal view. Therefore, we propose a digital visor that allows firefighters to see thermal images and non-thermal images in front of their face. Since VR can provide a close representation of what a real emergency is like, we can experiment these with views and provide the trainee with a thermal imaging visor in the VR world. This will help us gather practical feedback from the experience to ensure an effective product.

The tasks we researched included:

1) Setting up a realistic VR environment
2) A Thermal vision overlay visor
3) Object detection bounding boxes
4) Shaders that simulated infrared light
5) Fire propagation simulations
6) Smoke simulations
7) Balancing the refresh rate for all of the simulations to optimize overall performance

II. BACKGROUND AND MOTIVATION

A major motivator for this project is the fact that the visor idea was part of a research proposal by Jiasi Chen. The research proposal explains the enhancements that can be made for firefighters to assist them in their work. In particular, this means enhancing current thermal imaging interfaces for more streamlined use during an emergency.

We want to test the effectiveness of a thermal vision overlay which can be projected onto a visor over the firefighter’s face. The current interface is a handheld device which firefighters must unclip, point towards the objects that

*This work was not supported by any organization

they want to analyze with thermal imaging, pull a trigger, then reclip the handheld device onto their person. An overlay visor is easily toggled on and off and keeps the firefighters hands free. The VR environment that we create also proves useful for training firefighters in a cost-effective manner. The overseeing professor for this project, Jiasi Chen, has connections with the Riverside Fire Department and they are willing to work with us to ensure our simulations are as realistic as possible. They will also be able to provide feedback on how effective the overlay visor may be and can suggest improvements. They have verified that there is demand for anything that can make search and rescue missions more efficient.

III. IMPLEMENTATION

A. Setting Up a Realistic VR Environment

The environment we chose to create in the virtual world is the interior of a burning house. This is a very common scenario for firefighters and it is easy to find 3D models to build a residential environment with the Unity application. In order to make the house look as realistic as possible we found open source assets which included 3D models of the house itself, furniture, a human, and materials / textures. The models themselves had proportions accurate to real life. The materials used physically-based rendering in order to model light as accurately as possible. After all assets were collected, the scene was constructed to look natural yet also leaving space for fires to spread and for the user to navigate around the area naturally.

B. Thermal Vision Overlay Visor

The effect of having a thermal imaging visor is done by setting up a second camera which views the world in thermal vision in addition to the regular camera. By doing this, we can swap between these two cameras to replicate the visor turning on and off. Simulating thermal imaging was difficult because 3D engines such as Unity, our chosen game engine, are built to render visible light. There is little to no existing models for rendering infrared light on 3D meshes. We researched a few possible options in which we could alter the rendering pipeline. A rendering pipeline calculates how visible light bounces off objects based on properties such as its reflectivity. If we had the ability to edit this, we could render objects' brightness based off of their temperature. The first option was using some other software (instead of Unity) which completely exposed the source code for the rendering pipeline such as Godot. However, we would sacrifice the benefit of unity's large library of 3D models and materials. This means that we would have to spend more time focusing on the visuals instead of the programmatic task at hand. So, we researched the features that Unity provided. Unity 2018 has an experimental scriptable rendering pipeline. However, since it is experimental, this is not a good idea because the implementation is highly in flux at the time of our research meaning that any experiments we performed would be difficult to replicate in the future. The next best option we would use to change the way that 3D models were rendered was programming **shaders**.

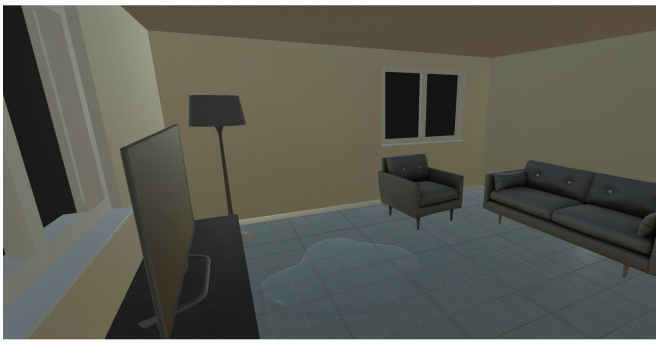

Fig. 1. Default Render of the current scene.

Unity has an API called **ShaderLab** which is built upon and is very similar to High-Level Shader Language (HLSL) [7]. A shader is a script which dictates what RGB color values are on that particular pixel on the 3D model. Shaders have access to world coordinates and data vectors with which it can do math to calculate the colors based on variables in the environment such as light, position, and angle of the pixel. In order to apply a second shader onto the 3D model for thermal rendering (in addition to the visible light shader), a new material has to be applied to the object. The problem is that only one material can be applied to any 3D model at a time. After considering different options, the simplest approach was to create a second mesh that intersected with the first. Therefore, every single 3D model in the scene is actually two 3D models. I shall take a table as an example. One version of the table as a normal material that looks like wood in normal light. The first table has a second table with the exact position and rotation of the first table but with a material that has our custom thermal shader. At any given time, you only see either the visible light textures or all of the thermal shaders. We accomplish this with utilizing the cameras **culling** setting which determines which objects in the scene to render and which objects to ignore. By default, the camera shows all 3D models regular textures only. When the user switches to thermal mode, the camera ignores all the 3D models with the regular textures and shows only the 3D models with thermal shaders. With static objects this is not a problem but with dynamic, moving objects, we had to be careful to make sure that the two parallel 3D models interacted the exact same way with other objects so that they stayed in sync with each other. Despite seeming like a strange approach to the double shader problem, having twice the amount of 3D models as originally intended does not have a significant negative impact on performance.

C. Object Detection

One feature that we wanted to simulate in our project is object detection to assist firefighters in detecting points of interest through thick smoke. Examples include humans to be rescued and doorways to navigate through. The objects to

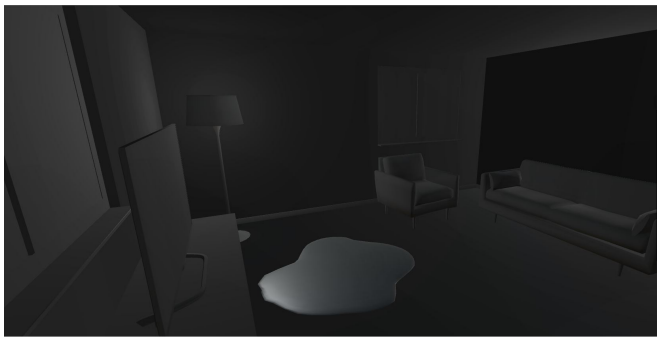

Fig. 2. Thermal Render of the scene with Culling and Shaders.

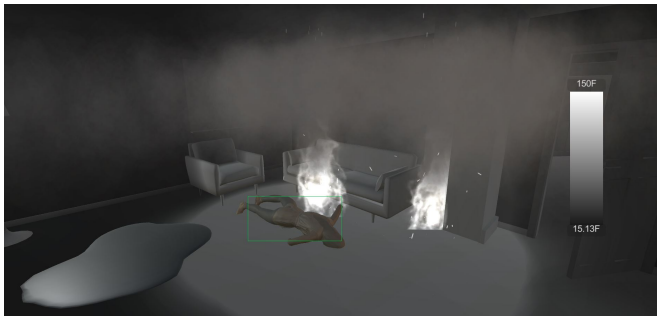

Fig. 3. Object Detection. Raycasts are shot out of the camera's viewpoint to detect specific objects, such as the dummy on the floor. When the object of interest is detected, we can draw a bounding box around it.

be detected are highlighted with a tightly-bound box outline that is bright green so that it is easily noticeable to the human eye. The bounding boxes are rendered after all objects in the scene are rendered. This means that once every object in the scene has been colored and textured with respect to the cameras perspective, a separate script takes a look at that scene to detect which objects are visible and need a bounding box. The post-processing script looks at the scene as if it were a snapshot and sends **orthogonal raycasts** outward from the cameras viewpoint in order to check all visible objects. An orthogonal ray is sent from every pixel of the screen (determined by screen resolution). The rays direction is normal to the view plane in the direction of the users gaze. Whenever each ray hits an object, it checks if that object is marked to be surrounded by a bounding box. If it is, then that pixel's position is saved in a dictionary data structure where the key is the objects unique name and the value is a struct that hold the leftmost, rightmost, highest, and lowest hit pixel for that particular object. Once all the pixels have been cast, each the four corners of each visible object are known and then lines are rendered to connect those points. This process of casting a ray for every pixel on the screen is computing-intensive so it does not occur every single frame. It instead occurs every 0.1 seconds. Another measure to improve performance is that not every single pixel is used to send a raycast; every 5 pixels are sampled instead. The 0.1-second delay between bounding box outline updates is noticeable and accurately represents the delay that the exists when actual machine learning is used to detect objects in real life.

Fig. 4. Thermal view. By looking at the temperature gauge, one can see how the shader can take specific values of an object and changes its pixels depending on the temperature of the actual object and its surrounding objects.

D. Shader Mappings to Infrared Light

We designated certain objects to implement a **heat source** class which keeps track of temperature of the heat source and the world position in xyz coordinates. The temperature of the heat source is a value between 0 and 1. A temperature of 0 is multiplied to the pixel to make the resultant color pitch black since it it as cold as possible. A temperature of 1 is the hottest which would result in the brightest available shade of white on the receiving pixel. Similar to the temperature in the heat source, each 3D model with a thermal shader has a stored value called conductivity which represents how receptive the object is to surrounding heat sources temperatures. There is an adjustable value within the range of 0 and 1 on each heat source and there is also an adjustable value for each object. These variables help us represent heat sources of differing temperatures such as fires and light bulbs and different materials such as wood and metal. These values are multiplied together to determine how black or white the resultant pixel is. To make the thermal shader, we had to program each instance of the shader to keep track of all nearby heat sources. For every pixel, the shader calculates the distance between each heat sources world position point to the current pixels world position point. The shader then multiplies the 3D models thermal conductivity variable and the heat sources temperature variable and applies an inverse square formula based on distance to determine how hot that pixel is. For example, let's take a fire which has its heat source point set to (0, 0, 0) and a table which is two meters away. The pixel at the corner of the table is located exactly at (2, 0, 0), the table has a thermal conductivity factor of 0.5 and the flame has a temperature of 1. The heat factors (normalized between 0 and 1) are multiplied to obtain 0.5. Then the square distance would be 4, giving us a final temperature value of 0.125. All of these values are normalized because the RGB value of the pixel is also normalized between 0 and 1. This makes it easy to create the thermal color by applying the final temperature values to the RGB values of the pixel. So, the final color of the pixel would be (0.125, 0.125, 0.125) which is a dark gray. This process occurs for every pixel and for multiple heat sources.

Fig. 5. Fire propagation. The fire intially starts on the floor but eventually catches onto the table. It will continue to spread to flammable objects in a grid-like fashion until it cannot spread any further.

E. Fire propagation

Fire propagation is handled by allowing fire to spread in different directions across flammable surfaces. For this project, we used a grid-like layout where each square is exactly the same width and height. We start with a fire known as the **source fire** and allow it to spread across all adjacent squares in the grid. Initially, the source fire creates four more of the same fires in the x and z axes (along the ground plane). We do this via instantiation which allows us to spawn a fire at a given location. To ensure that fire propagation is true to real fires, the fires must connect and be close to each other so that we could trace the fire to its source. The fires must be close enough to seem like they are seamless since real flames tend to stick together with a sort of surface tension. We can also modify the intensity of the fire so that it can instantiate more fires some ”n” squares away from the origin of the source fire. This essentially sets the maximum propagation distance from the source fire.

The above case assumes that we do not want to spread fires. This class of fire is one which grows in size but is still considered to be one entity. We can extend the idea of propagation through spreading. Spreading occurs when a nearby object is flammable and a nearby source fire causes a completely separate source fire to be created on the flammable object. A separate source fire will be the center point for flames to propagate from. To allow for spreadable fires, we have to introduce the concept of flammable objects. In real life, certain types of objects will catch on fire. Different materials will light up in different rates. To do this, we add a **flammability index** to deduce how fast an object catches on fire. Furthermore, we also want to tag specific objects to be flammable and assign a material type to the object. The material type maps to a flammability index so that we know how fast a fire can spread over an object. From here, we research some material types such as wood and fabric and create associations between the material and its relative flammability index. We can take our basic case where we spread fire in four different directions. This time however, we will check if a flammable object is present in each of the adjacent squares. If such an object exists, we can create a fire at the location of the flammable object and allow that fire to spread to its adjacent squares. If an adjacent square is not flammable or is already on fire, we will not create a fire at that square.

When spreading fire, we take into account the material type of the flammable object. We can allow fires to spread faster or slower depending on its type. That is, we can make a leather sofa catch fire more slowly than a fabric sofa of the same size.

F. Smoke Simulations

Smoke is represented by using particle systems in Unity. By using the Shuriken particle system engine, we can tweak the settings of the particle system to create realistic smoke. That is, the speed, size, and position of the smoke is changed during runtime. We modify these settings in order to make our smoke behaves the way it should. Each particle takes in a particular texture, so an image of a small cloud of smoke is used for it. While one particle by itself will not look like smoke, having multiple particles grouped together achieves the results we desire.

Once we have realistic-looking smoke, we need to manipulate how it moves. Our basic case involves having the smoke rise in the air. By default, particles do not take collisions into account so we need to make sure that they can collide to the walls of our environment. Furthermore, we want to direct the smoke to spread along a wall once it collides into a wall. Since there are so many particles that spawn at the same time, we can have particles randomly move in different directions as they hit the ceiling of the simulated house. The ”Force Over Lifetime Module” [8] from Shuriken allows us to change how our smoke particles react overtime. As such, we use this module to let smoke rise in the air in the beginning and have it move in a particular direction. Ideally, we can make the smoke particles drop to the ground due to smoke being heavier than air once it cools.

There are some issues that lie within creating the most optimal smoke for our experience. Smoke particles can take up a significant amount of CPU memory as more and more particles are spawned. This can take a toll on the PC that is running the experience and, therefore, low framerates will ensue. Such a setback can potentially ruin the experience for the player since we want to achieve an optimal frames-per-second (FPS) throughout the experience.

Unity’s cameras experience clipping when the camera comes close to objects. Clipping occurs when an object is so close to the camera, that a portion of it is not rendered. It gives the illusion that you are inside the object and ruins the feeling of immersion. This means that there will be trouble moving through a room filled with smoke particles that each experience clipping. Due to the camera clipping issue, all surrounding smoke within the cameras view will not be visible. Furthermore, moving through the smoke becomes less realistic as the players camera clips through the smoke.

From the viewer's perspective, chunks of smoke instantly disappear when approached.

G. Optimizing Overall Performance

From the implementations explained above, it is clear that there are a lot of updates that occur within the scene which have the ability to slow down performance. These include multiple fire particle systems, multiple smoke particle systems, fire propagation and spreading updates, thermal imaging updates on a pixel-by-pixel basis, and bounding box outline raycasts. In order to minimize the impact on the simulations overall performance, we had the updates occur every few frames. We did not explicitly have simulations take turns round-robin style. Instead, each simulation has it's own independent timer. In the worst case scenario, all updates may occur on the same frame. Therefore, this frame may experience lag but it is not noticeable. To further help with performance, the particle systems are limited to have a maximum amount of particles that will not noticeably affect framerate.

IV. EXPERIMENTS

A. Firefighter Tests

We created an experience that allowed the player to traverse through a burning house. The layout was structured to show off the features we worked on to make the experience as realistic as possible. This included the thermal imaging view for both the visor and the FLIR, fire propagation, object detection, and an environment to traverse in with physically-based lighting.

With the support of the Riverside Fire Department, we were able to consult David, a firefighter, to test out our experience. The overall feedback from him was positive. He was immersed by the burning house and did not have trouble walking through it. He was also impressed by the thermal vision and being able to switch between normal and thermal views. He mentioned that the inverse square drop off of the thermal vision brightness was accurate compared to real life. While he thought the fire was realistic, he mentioned that the smoke could be improved by having it fully engulf the room that its in after a certain time. Currently, the smoke disappears after a period of time to avoid to particles from accumulating in the scene. However, we need to increase the lifetime of each smoke particle so that they have enough time to accumulate enough to fully obscure vision.

David suggested more ways to add onto the experience. He mentioned that there were different types of fire phenomena that can be emulated and would further enhance the experience if they were recreated, such as a flashover. He also explained how the FLIR camera can detect the hottest object in its current view, and a visual indicator can show where that point is on the screen.

V. FUTURE WORK

Despite creating an intractable firefighting scenario, the main issue lies on creating the realism of the experience itself. Aside from the graphics, there are ways that we can improve how certain parts of the experience work.

Currently, fire propagation does not actually spread in all the adjacent tiles as explained in the implementation section of the document. We want to improve this by making sure it spreads evenly. This will allow the fire to look more visually-appealing. We also want to figure out ways to emulate the behaviors of smoke. Currently, smoke spreads in the air, but it does not drop to the ground. The problem lies with how to manipulate its force during its lifetime. Even when we figure this out, we need to make sure we keep its realistic look. While we can make smoke rise up in unison, we need to make sure that it can also spread and accumulate to the highest point in a room.

Our shader algorithms will also need some improvement as well to fine-tune the pixel calculations for specific objects. This way we will be able to accurately depict the temperature values of any object and display them visually through our thermal display.

Finally, we need to fine-tune the performance of all of our simulations. There are ways that we can optimize our code so that we can avoid expensive operations that could slow down the experience. If the user notices a reduced framerate during their simulation, then they will feel less immersed in the experience and the overall experience will be less effective.

VI. CONCLUSION

The project helped provide a realistic proof-of-concept of the potential of firefighter technology for the future. By creating a VR environment and simulating various processes of firefighting, we can create safe and cost-efficient scenarios that can give firefighter trainees a realistic experience.

Along with this, we can see the potential of improving tools for firefighters. The VR experience provides prototyping tools that currently do not exist today, and will give us a better idea as to whether the technology is worth pursuing. This can be seen with the thermal vision visors which provides an efficient and convenient thermal view to the user.

The initial experiments of the prototype thermal vision interfaces were successful. By allowing a real firefighter to test the experience, we were able to obtain substantial data on how to improve the experience. The firefighter that tested the experience concluded that everything in the simulation was an accurate depiction of a real-life emergency. He noted how some parts of the experience such as the unrealistic smoke would take away from the immersion, but overall the experience would prove to be useful with improvements.

There is still a lot to improve on in the experience. The experience can be more realistic with enhancements to the fire propagation and smoke simulation. Our thermal view would also need adjustments to distinguish objects of different temperatures in a more intuitive manner. Finally, by improving the overall performance of the software to maximize the frame rate, we can provide a seamless experience to the user.

REFERENCES

[1] Fire Science. (2018). How to Become a Firefighter. [online] Available at: https://www.firescience.org/how-to-become-a-firefighter/ [Accessed 15 Jul. 2018].

[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135./

[3] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Smith, An approach to graphs of linear forms (Unpublished work style), unpublished.

[5] E. H. Miller, A note on reflector arrays (Periodical styleAccepted for publication), IEEE Trans. Antennas Propagat., to be publised.

[6] J. Wang, Fundamentals of erbium-doped fiber amplifiers arrays (Periodical styleSubmitted for publication), IEEE J. Quantum Electron., submitted for publication.

[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[8] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, Electron spectroscopy studies on magneto-optical media and plastic substrate interfaces(Translation Journals style), IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].

[9] M. Young, The Techincal Writers Handbook. Mill Valley, CA: University Science, 1989.

[10] J. U. Duncombe, Infrared navigationPart I: An assessment of feasibility (Periodical style), IEEE Trans. Electron Devices, vol. ED-11, pp. 3439, Jan. 1959.

[11] S. Chen, B. Mulgrew, and P. M. Grant, A clustering technique for digital communications channel equalization using radial basis function networks, IEEE Trans. Neural Networks, vol. 4, pp. 570578, July 1993.

[12] R. W. Lucky, Automatic equalization for digital communication, Bell Syst. Tech. J., vol. 44, no. 4, pp. 547588, Apr. 1965.

[13] S. P. Bingulac, On the compatibility of adaptive controllers (Published Conference Proceedings style), in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 816.

[14] G. R. Faulhaber, Design of service systems with priority reservation, in Conf. Rec. 1995 IEEE Int. Conf. Communications, pp. 38.

[15] W. D. Doyle, Magnetization reversal in films with biaxial anisotropy, in 1987 Proc. INTERMAG Conf., pp. 2.2-12.2-6.

[16] G. W. Juette and L. E. Zeffanella, Radio noise currents n short sections on bundle conductors (Presented Conference Paper style), presented at the IEEE Summer power Meeting, Dallas, TX, June 2227, 1990, Paper 90 SM 690-0 PWRS.

[17] J. G. Kreifeldt, An analysis of surface-detected EMG as an amplitude-modulated noise, presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.

[18] J. Williams, Narrow-band analyzer (Thesis or Dissertation style), Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.

[19] N. Kawasaki, Parametric study of thermal and chemical nonequilibrium nozzle flow, M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.

[20] J. P. Wilkinson, Nonlinear resonant circuit devices (Patent style), U.S. Patent 3 624 12, July 16, 1990.