Final Project Report

Introduction

The purpose of this project was to research and implement the tools needed to create a game with computer-vision based controls. This project was originally presented to me by Brother Richard Grimmett. The goal for the project was to develop tools that students can use in future projects as well as finish a spear-throwing game as part of a museum exhibit and teaching tool in the future. The primary tools I was meant to use were Unity and OpenCV. Unity is a game development engine that is popular for making games for a wide variety of platforms. Unity provides several tools and allows for both 2D and 3D game development. OpenCV is a programming library that provides a wide array of tools related to computer vision. The library can be used to set up anything from motion detection to face recognition to pose estimation. It is a powerful open source library with potential. Work done previously on the project included a mini-game with spear-throwing mechanics and mammoth non-player characters (NPCs), but this mini-game was not set up to use computer vision. There was also some work done with creating basic OpenCV motion sensing scripts, but these scripts were not written in a form that could be used in the mammoth game. My project presents details that will bring this mammoth game closer to using motion controls.

Methods and Procedures

There were two main areas that I had to focus on in terms of research: Unity and OpenCV. I had to spend a minimum of 100 hours within a 13-and-a-half-week period researching these tools so that I could then document my findings for future use.

For my research into Unity, I was given a link to a YouTube tutorial by Brother Grimmett: https://youtu.be/Sqb-Ue7wpsI. This 8-hour long tutorial demonstrated the creation of a basic survival-style game and was a useful example to start with. According to Brother Grimmett, this tutorial was also the original base for the current version of the Mammoth game. This tutorial demonstrated the most essential tools for creating a 3D Unity game. Tools and techniques showcased in this tutorial included the creation of lighting elements, terrain building, creating game objects and scripts, and other such tools. These tools are used to create player controller and attack scripts, NPCs, and various other game elements. This tutorial was made using a set of pre-made resources, such as 3D models, textures, and sound files, so it was unnecessary for me to develop those resources on my own.

Game scripts are the actual code portion of Unity projects, they allow logic to be applied to game objects. For example, you can create a player-character object and attach scripts that preform movement and camera control. You could then attach more objects to that character representing the tools or weapons your character uses and write scripts that register button prompts and handle item interactions. In Unity, all these scripts normally need to be written in C#. Because OpenCV normally requires Python code, the next step was to figure out one of two things: how to write a Unity script in a language other than C# or how to use OpenCV with a C# Unity script. These tasks required research into OpenCV.

OpenCV was the most difficult part of the project to work with. There is an asset in the Unity Asset Store called OpenCV for Unity, but there were some issues with this asset. This asset allows for easy integration of OpenCV into Unity, but this resource costs $95.00. I did not have a budget for this in my project, so this asset was not a viable resource and I had to find my own solution. The most common and easy language in which OpenCV is used is Python, but Python scripts do not work in Unity, so my job was to find a way to use OpenCV in another language.

I began looking into ways to use OpenCV in C++, as I am most familiar with the C++ language. Through this research, I discovered tools in Visual studio that allow the user to include references to additional libraries such as OpenCV. The process can be somewhat complicated, but I will expound more on the exact steps in the results section of this document. This research did give me a way of using OpenCV in C++, establishing an additional set of tools for students who work on this project and similar projects in the future. There are ways, such as a method I was looking into called "marshaling", that C++ code can be applied to C# projects, but I was unable to finish working out these methods.

Another tool I discovered while searching for ways to connect OpenCV to Unity is a wrapper called Emgu, specifically Emgu CV. Emgu CV is .Net wrapper for the library. A wrapper is essentially a container which translates a library's existing interface into a separate interface compatible for a different situation. Emgu can be used to wrap OpenCV so that it can be accessed via a number of different languages, including C#. I only discovered this tool in the last few weeks of my project and was unable to finish studying the documentation and configuring it. I found that most of the tutorials and example code sections offered on the Emgu CV website worked with older versions and would present multiple bugs when trying to run example code. All the same, Emgu CV has is a powerful tool and may be the final key to finishing this project.

Results

I was able to finish a C++ motion detection script, but there is a bit of a process involved in configuring a C++ project to use OpenCV. OpenCV can be installed from the following link: https://sourceforge.net/projects/opencvlibrary/files/opencv-win/. The process of setting it up for use with C++ is as follows:

1) The first step is to add OpenCV to your computer's environmental variables. After installing OpenCV, go to your computer's environmental variable settings, as shown below in figure 1. First, open the "Path" variable and add the path to the bin folder for the version of OpenCV you will be using as shown in Figure 2. The path that I added was *"C:\opencv\build\x64\vc15\bin"*. Next, add the path to OpenCV's build folder to the System Variables section as shown in figure 3 and name it *"OPENCV_DIR"*. Adding these paths will make it easier for C++ projects to access OpenCV.

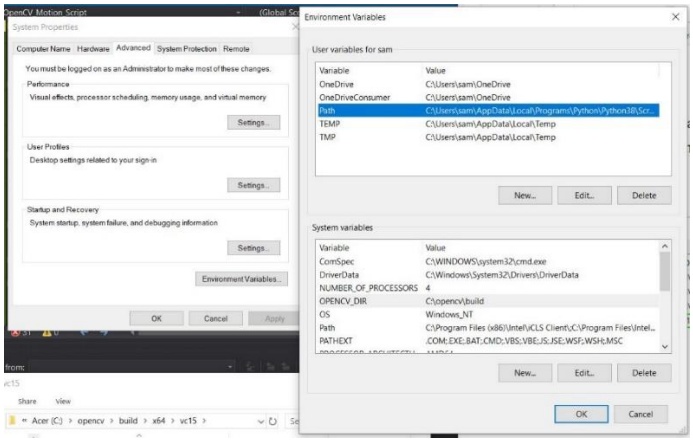

Figure1

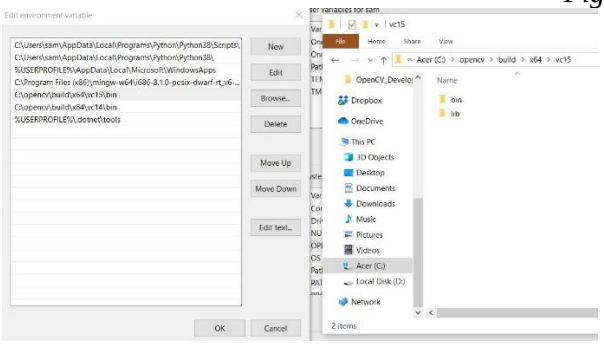

Figure 2

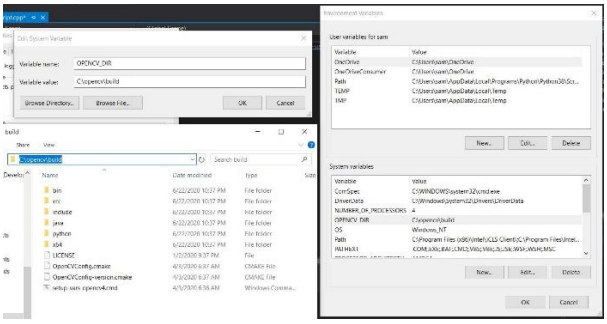

Figure 3

2) Create a new C++ project in Visual Studios. Open the project's properties from the project tab as shown in Figure 4.

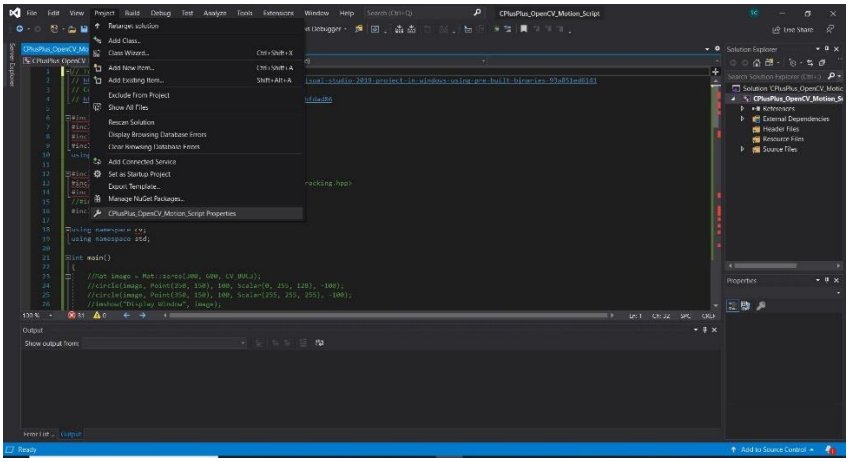

Figure 4

3) Ensure that the Configuration tab is set to "All configurations" and the Platform tab is set to "x64" as seen in Figure 5. Open the selection *Configuration Properties*, *C/C++*, *General*, *Additional Include Directories* as seen in Figure 5. Add the path to OpenCV's include folder. The path I used was *"$(OPENCV_DIR)\include"*, as seen

in Figure 6. The *"$(OPENCV_DIR)"* part of this path comes from the Environmental Variables we set up in step 1.

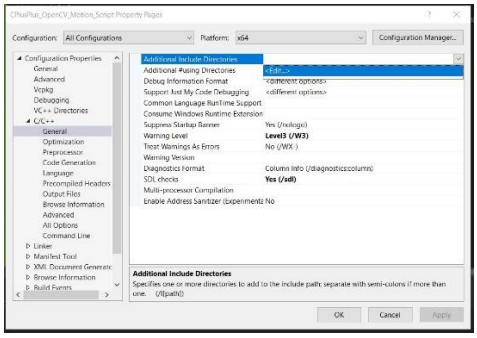

Figure 5

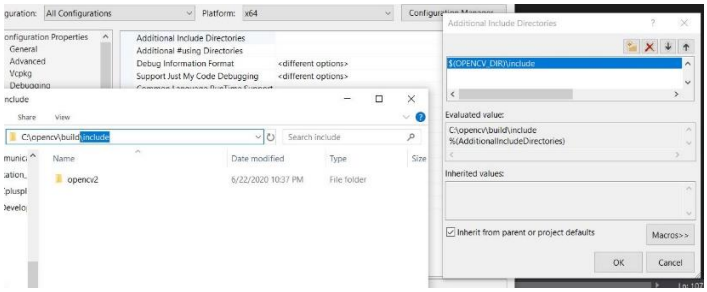

Figure 6

4) Open the selection *Configuration Properties*, *Linker*, *General*, *Additional Library Directories* as shown in Figure 7. Add the path to the lib folder for the version of OpenCV that you are using. The path that I added was *"$(OPENCV_DIR)\x64\vc15\lib"*, as seen in Figure 8.

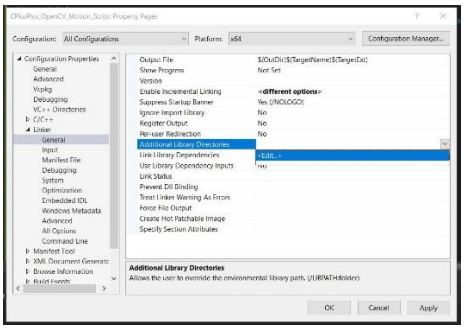

Figure 7

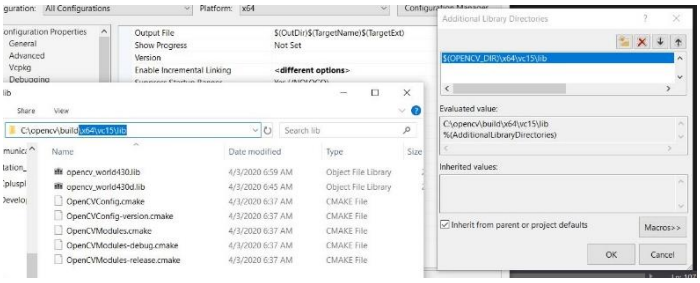

Figure 8

5) Open the selection *Configuration Properties*, *Debugging*, *Environment* as seen in Figure 9. Add *"PATH="* followed by the path to the bin folder for the version of OpenCV that you are using and end it with *";%PATH%"*. I used *"PATH=$(OPENCV_DIR)\x64\vc15\bin;%PATH%"*, as seen in Figure 10.

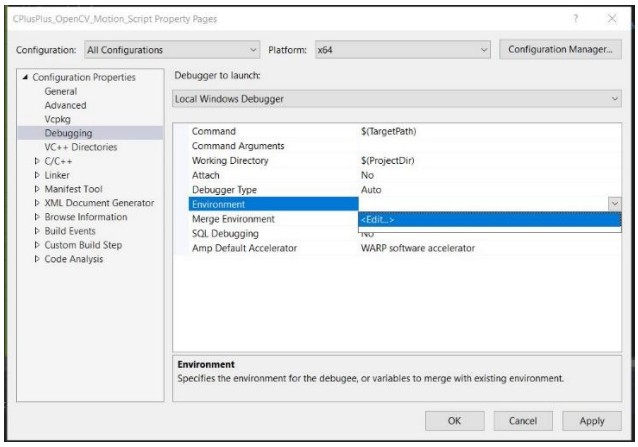

Figure 9

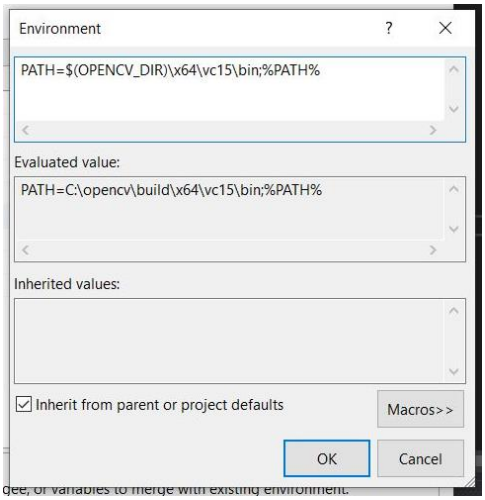

Figure 10

6) Open the Configuration Manager window and make sure the window matches what is shown in Figure 11. After closing this window, make sure to hit the Apply button for all the changes from the previous steps.

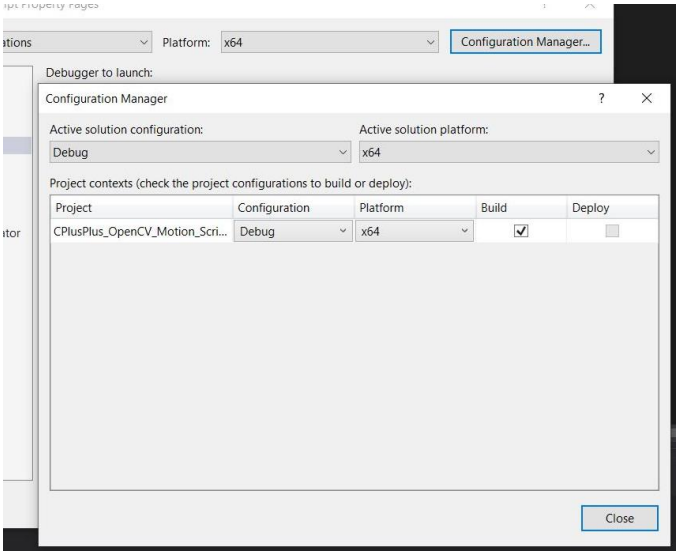

Figure 11

7) Change the Configuration tab to Debug as seen in Figure 12. Open the selection *Configuration Properties*, *Linker*, *Input*, *Additional Dependencies*. Add the *opencv_world###d.lib* file that is in the lib folder of the version of OpenCV you are using. The file I added was *"opencv_world430d.lib"* located at *"$(OPENCV_DIR)\x64\vc15\lib"*, as seen in Figure 13. Ensure that the file you add ends with "d.lib". Hit the Apply button after making this change.

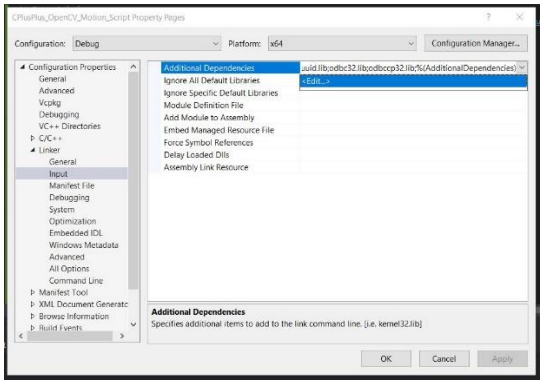

Figure 12

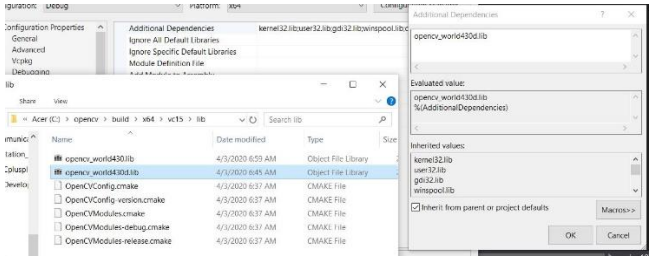

Figure 13

8) Change the Configuration tab to Release as seen in Figure 14. Open the selection *Configuration Properties*, *Linker*, *Input*, *Additional Dependencies*. Add the *opencv_world###.lib* file that is in the lib folder of the version of OpenCV you are using. The file I added was *"opencv_world430.lib"* located at *"$(OPENCV_DIR)\x64\vc15\lib"*, as seen in Figure 15. Ensure that the file you add does not end with *"d.lib"*. Hit the Apply button after making this change

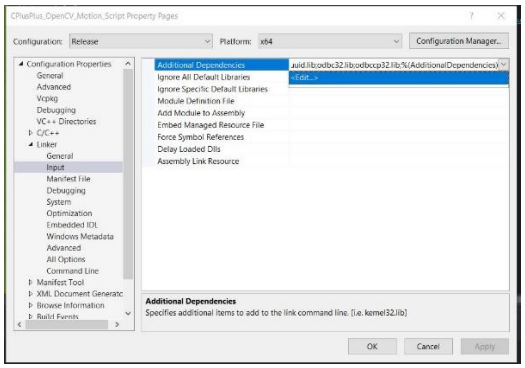

Figure 14

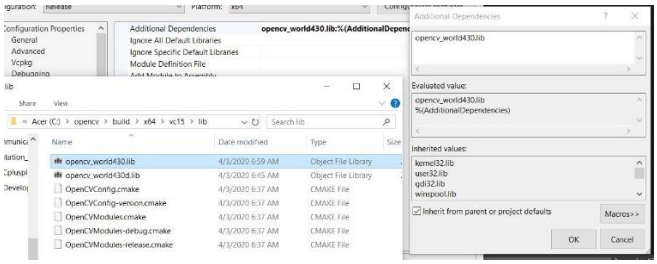

Figure 15

Once the C++ project has been set up, the following code can be added to implement motion detection with OpenCV. This code works by comparing two different frames captured by the computer's camera. There is a slight delay between each frame, meaning the differences between each is slight as well. Once the two images are captured, they are converted to grayscale then compared using the asdiff() function.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/video/tracking.hpp>
#include <opencv2/core/ocl.hpp>
#include <iostream>
#include <windows.h>
//#include <unistd.h> // some projects may need to add this include, I did not need to
```

```
using namespace cv;
using namespace std;

int main()
{
   Mat gray1, gray2, frameDelta, threshImage, frame0, frame1, frame2;
   vector<vector<Point> > contours;
   VideoCapture camera(0); // Open camera

   // Set video size to 512x288
   camera.set(3, 512);
   camera.set(4, 288);

   Sleep(3);

   // Capture the first image for frame 1 and frame 2
   camera.read(frame1);
   camera.read(frame2);

   while (true)
   {
      camera.read(frame1);

      Sleep(3); // Give a slight delay so frame 1 and frame 2 can be different

      camera.read(frame2);

      // Convert frame 1 to grayscale
      cvtColor(frame1, gray1, COLOR_BGR2GRAY); // Converts frame 1 to "gray2"
      GaussianBlur(gray1, gray1, Size(21, 21), 0);

      // Convert frame 2 to grayscale
      cvtColor(frame2, gray2, COLOR_BGR2GRAY); // Converts frame 2 to "gray2"
      GaussianBlur(gray2, gray2, Size(21, 21), 0);

      // Compute difference between frame 1 and frame 2
      absdiff(gray1, gray2, frameDelta); // Compares gray 1 to gray 2
      threshold(frameDelta, threshImage, 25, 255, THRESH_BINARY);

      dilate(threshImage, threshImage, Mat(), Point(-1, -1), 2);
      findContours(threshImage, contours, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);

      for (int i = 0; i < contours.size(); i++) {
         if (contourArea(contours[i]) < 500) {
```

```
            continue;
        }

        putText(frame2, "Motion Detected", Point(10, 20), FONT_HERSHEY_SIMPLEX,
0.75, Scalar(0, 0, 255), 2);
        cout << "Motion Detected" << ".\n";
      }

      //imshow("previous Camera", frame1); // uncomment to display frame 1
      imshow("Main Camera", frame2); // display frame 2

      //imshow("previous gray", gray1); // uncomment to display grayscale frame 1
      //imshow("Main gray", gray2); // uncomment to display grayscale frame 2

      //imshow("Delta", frameDelta); // uncomment to display the image comparison

      if (waitKey(1) == 27) {
        //exit if ESC is pressed
        break;
      }
   }
   return 0;
}
```

Conclusions

In the end, I was unable to complete the project. The two goals were this: First, find and research tools and resources that other students can use to work with OpenCV in the future. Second create a learning tool/game (the Mammoth game) that will be used as a part of a museum exhibit in the future. I did succeed in finding useful tools for future projects, but I was unable to get the Mammoth game working.

This documentation will still be useful to other students, though. This project shows how to add OpenCV's tools to any existing C++ project. This example will also help familiarize other students with the process of adding additional resources to a project in Visual Studio.

As for the Mammoth game, this project could be finished in the future through one of the two methods I mentioned earlier: Marshaling or using the Emgu CV wrapper. Marshaling could allow for C++ code to be used in Unity scripts, but it would require a little more research. The Emgu CV wrapper has a lot of useful tools, though, and I feel like shows a little more promise in getting the Mammoth game working with computer vision.

References:

- freeCodeCamp.org. (2019, January 21). Unity FPS Survival Game Tutorial - First Person Shooter Game Dev [Video]. YouTube. https://youtu.be/Sqb-Ue7wpsI
- Fernando, S. (n.d.). Install OpenCV with Visual Studio. Retrieved July 21, 2020, from https://www.opencv-srf.com/2017/11/install-opencv-with-visual-studio.html
- Park, Y. (2019, December 24). Adding OpenCV 4.2.0 to Visual Studio 2019 project in Windows using pre-built binaries. Retrieved July 21, 2020, from https://medium.com/@subwaymatch/adding-opencv-4-2-0-to-visual-studio-2019-project-in-windows-using-pre-built-binaries-93a851ed6141
- By: Thomas, W. (2016, September 11). Unity and OpenCV – Part one: Install. Retrieved July 21, 2020, from http://thomasmountainborn.com/2016/09/11/unity-and-opencv-part-one-install/
- By: Thomas, W. (2016, September 12). Unity and OpenCV – Part two: Project setup. Retrieved July 21, 2020, from https://thomasmountainborn.com/2016/09/12/unity-and-opencv-part-two-project-setup/
- By: Thomas, W. (2017, March 5). Unity and OpenCV – Part three: Passing detection data to Unity. Retrieved July 21, 2020, from http://thomasmountainborn.com/2017/03/05/unity-and-opencv-part-three-passing-detection-data-to-unity/

Other Useful links:

- https://opencv.org/
- http://www.emgu.com/wiki/index.php/Main_Page