ECE697 - Capstone Project

Dynamic Depth of Field with Eye Tracking

Neel Kelkar, Suchith Suresh, Nawal Dua
College of Engineering, University of Wisconsin - Madison, WI

Students: ndkelkar@wisc.edu, suchithsures@wisc.edu, ndua2@wisc.edu

Professor: matthew.malloy@wisc.edu

Abstract

In this project, we attempt to simulate human vision by generating refocusable videos that emphasize whatever a viewer looks at using Depth Estimation, Image Processing, and Eye Tracking. This project has the potential to create a display that is more akin to a window to another place than a video playing on a screen.

1 INTRODUCTION

Over the past couple of years, there has been a lot of research on increasing the immersion and realism of videos. One particular area of focus for increasing immersion is in conveying depth and highlighting a subject using defocus and depth of field.

Movies that have a cinematic defocus effect are made using extensive scripting and specialized lenses, both of which are not available for standard cameras and mobile phone videos. Mobile phone cameras always film with a wide depth of field because it is hard to predict what distance the desired subject will be at.

There have been attempts at simulating depth of field in smartphone cameras. One such attempt is "portrait mode", a feature now present in many smartphones, where the phone automatically blurs the background of a photo while keeping the subject (the foreground) clear. This effect was mostly achieved with binocular vision, where two lenses are used to calculate depth in an image, similar to how humans and other animals use two eyes to interpret depth. However, since most videos are captured on smartphones using only a single lens (monocular vision), depth estimation for videos has not seen commercial success. This in turn means that not many uses for videos with depth maps have been found.

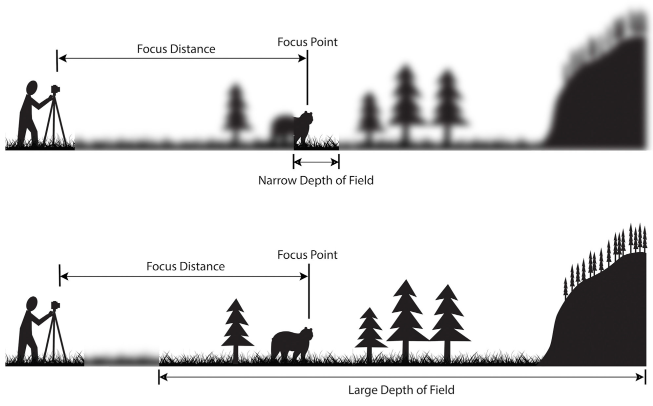
Recently, there has been significant progress made in regards to depth estimation using monocular vision. Google released their Pixel smartphone that was able to do portrait mode with a single lens, and also blur the background on images that were shot without using portrait mode, with the help of their image processing and machine learning prowess.

They were able to continue their research on this topic to generate a computer vision model that was trained on a dataset of mannequin challenge videos to generate depth maps for videos shot on a single lens. Their paper titled "Learning the Depths of Moving People by Watching Frozen People" [5] has pretrained models and code provided on GitHub. This model was part of the inspiration for our project and we use it to generate the initial depth maps.

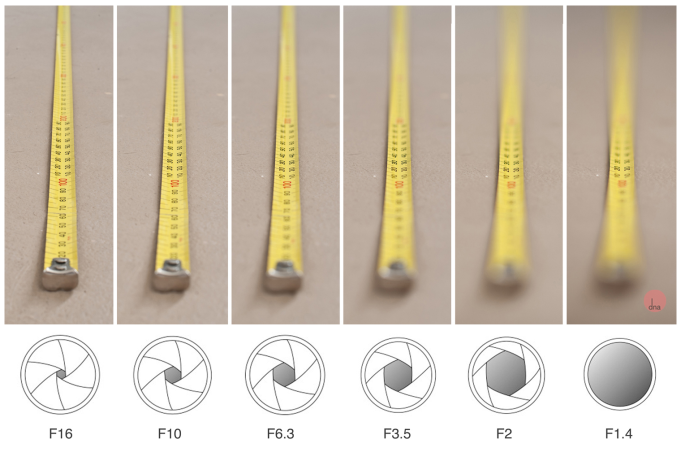
In our project, we use Google's mannequin challenge model to get depth maps on videos shot with a single lens and large depth of field. We propose to use eye-tracking to keep the subject of interest and all things with the same depth in focus while gradually blurring other objects in the background with respect to the object of interest. This will simulate a depth effect similar to the defocusing done by the human eye and will go a long way towards increasing viewer immersion. Please note that some of our choices in this project are made with the motivation of improving viewer immersion rather than using a hard and fast metric.

1.1 Depth of Field

In photography, Depth of Field (DoF) is defined as “The distance between the nearest and the furthest objects giving a focused image” [1].



(a) Narrow v/s Large DoF[8]



(b) Aperture affecting DoF[4]

Figure 1. Representation of Depth of Field

As shown in Figure 1 (a), the large depth of field is how most videos are shot on smartphones as they are more generalizable and can have everything in focus. However, a shallow/small depth of field is often used in cinematography to provide more emphasis and is also how our eyes work when focusing on objects of different depths. The human eye has a circular muscle called the ciliary muscle that expands and contracts to reshape the cornea and in turn, focus on objects of varying depths [6].

1.2 Mathematical Background

As figure 1 (b) shows, the aperture of the lens also contributes to the depth of field. The aperture width is used to calculate the blur of objects in relation to the center of focus. Accurate calculations of Depth of Field (DoF) use complex equations that are excessive for our application, hence we will use simplified equations [3] to estimate the blur levels necessary in the out of focus regions of each frame.

$$b = d * m\frac{x_d}{D} \qquad \text{Equation 1.}$$

where,

b	: blur
d	: aperture value
m	: magnification of the subject
x_d	: relative distance between the subject(region of interest) and the background or foreground
D	: distance between the foreground and background

The simplified equations assume paraxial approximation, hence we can use gaussian filters to simulate the expected blur.

2 GOOGLE'S MANNEQUIN MODEL

For the depth estimation part of our project, we used Google's "Depth from moving people" model. There are other models that claim to perform better, but Google's was the best one with readily available code. This model was trained on the mannequin challenge data set. This dataset is a collection of video clips from the viral Mannequin Challenge where people imitate mannequins, i.e. they are still and frozen in diverse, natural poses, while the camera moves around the scene. They were able to get around 170,000 frames of useable data from the dataset of around 2000 YouTube videos.

These videos were first preprocessed to a lower resolution and about a 60° field of view. This is so that all the videos were of the same quality and had the field of view of a traditional smartphone. These videos were then reprocessed to a higher resolution using Structure-from-motion. The depth was then calculated using multi-view stereo and Parallax for monocular lenses.

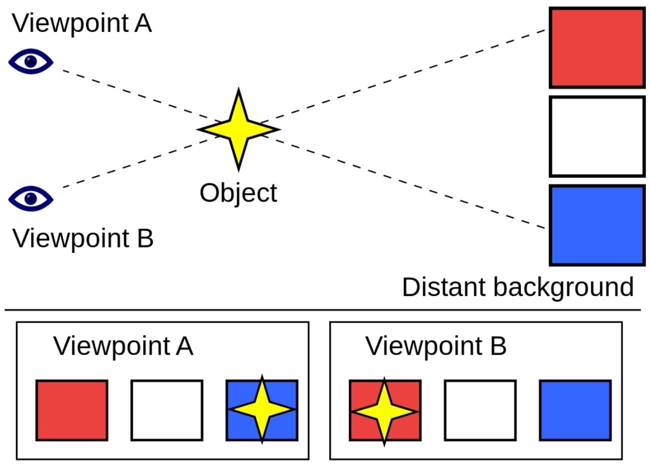


Figure 2. Object is at different locations when viewed from different viewpoints

Parallax is when the position of an object appears to be different when viewed from different positions [2]. It is used in numerous different ways and cases to estimate depth. Us as humans use our eyes in a similar way to parallax as both our eyes are at different positions, so the difference in the position of objects that we see from one eye compared to the other helps us to determine depth. Astronomers use parallax to estimate the distance of objects in space. The farther away an object is, the more displacement is observed between different points of observation. Google exploits this fact to estimate depth. As the camera moves around the scene, closer objects seem to move more between frames while further objects appear to move less.

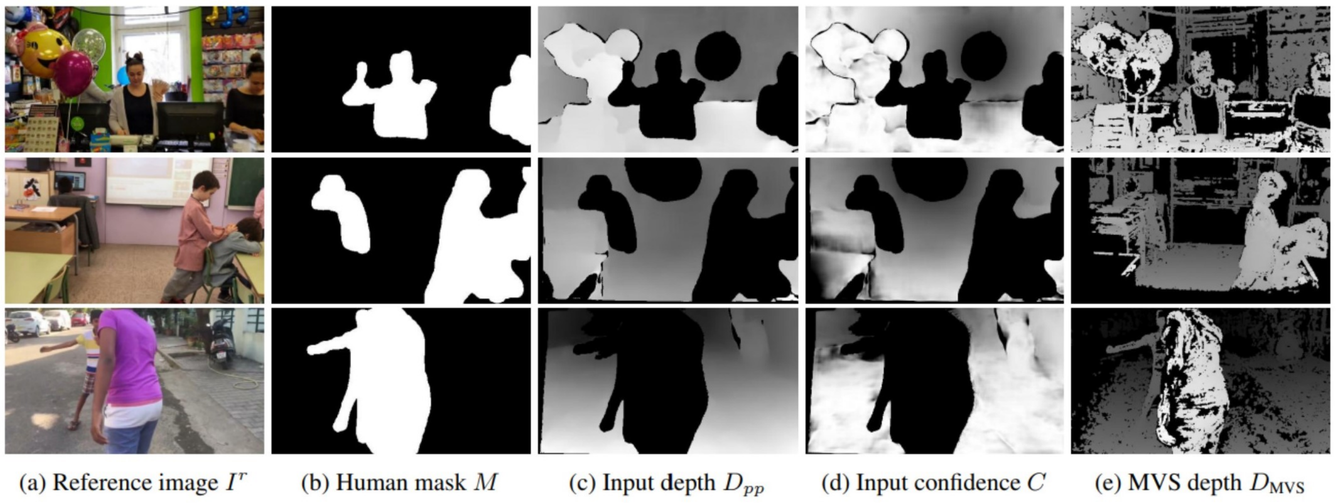


Figure 3. The 5 different inputs to Google's model

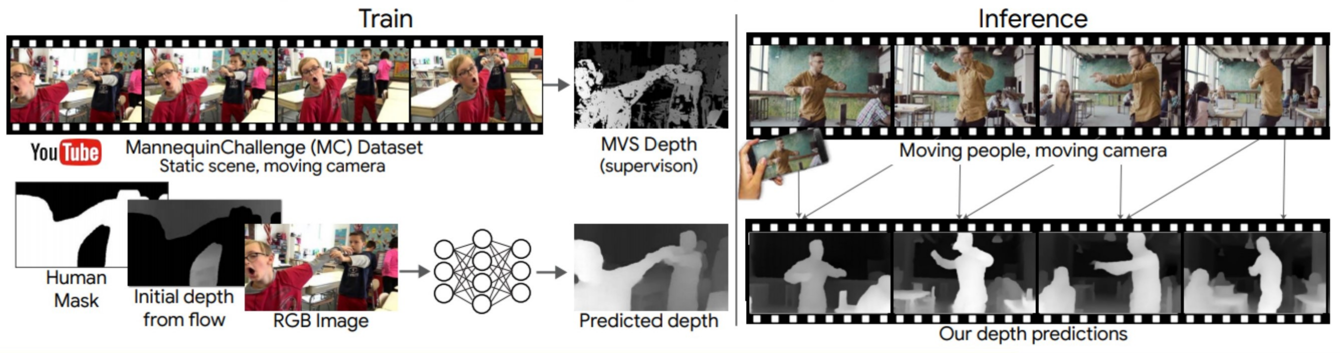


Figure 4. A rough overview of how google's model works

2.1 Modifications for our project

Google's model outputs each frame as an image of the depths of the objects. However, for our project, we required the data of the depth maps. Hence, we had to alter their code a little bit to save the depth disparities as a NumPy array. The edits can be found in our project on GitHub.

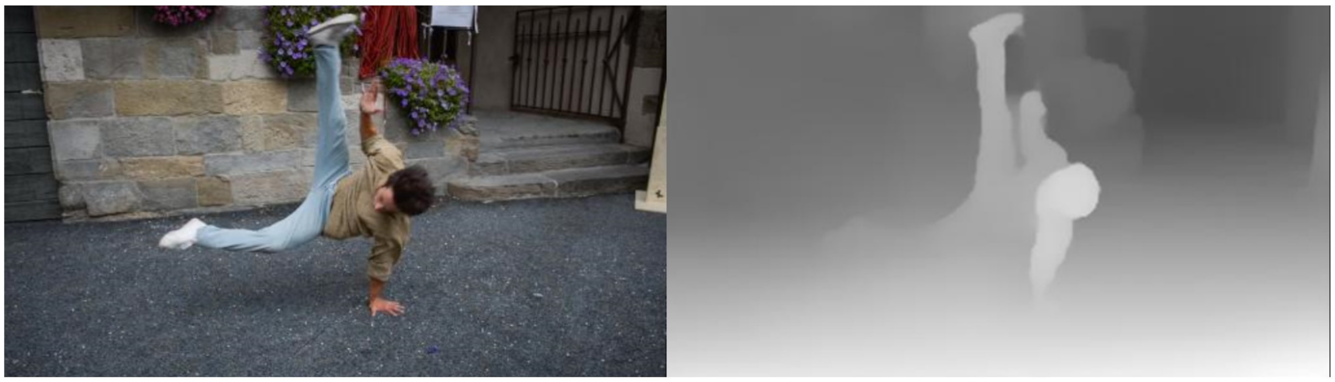


Figure 5. The original(unchanged by us) output of Google's model

3 CORE FUNCTIONS

The core functions created for the project can be broadly divided into four main parts, viz.:

1. Discretization of depth
2. Variance and Gaussian kernel size calculation
3. Blurring and Look-up Table
4. Output Functions

3.1 Discretization of depth

The depth maps that we obtain from the google model contain real numbers between 0 and 1, where the farthest objects are at depth level 1 and the nearest objects are at a minimum value in (0,1).

For the sake of speeding up computation, we divide the depth in this image into ten evenly spaced discrete levels between the minimum and maximum depths. This does not affect the end result as long as the level of blurring is not too exaggerated.

3.2 Gaussian Kernel

The main goal is to emulate a cinematic Depth of Field in common videos, therefore, our first aim is to calculate the blur to be applied to the available depth zones of any given frame.

To achieve this we calculate the region of focus or 'Focus Array' for each frame for each of the ten depth zones, which is used to divide the frame into 4 regions of blurring.[Example: {0. 0. 1. 2. 2. 2. 3. 3. 3. 3. 3.}, {0. 0. 0. 1. 2. 2. 2. 3. 3. 3. 3.},...,{3. 3. 3. 2. 2. 2. 1. 0. 0. 0. 1.}, {3. 3. 3. 3. 2. 2. 2. 1. 0. 0. 0.}].

The variance of the Gaussian blurring kernel for each of the 4 regions called the 'Variance Window' is calculated based on the user's variance parameter called 'Blur Variance' and the calculated step size. Step size is calculated as one-tenth of the difference between the largest and smallest non discretized depth map values of the frame rounded to 3 decimal places.

The Gaussian kernel width is calculated in 1D as:

$$G_1(x, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \qquad \text{Equation 2.}$$

where,
σ : Standard deviation, square of which (σ^2) is called variance is used to determine the width of the kernel
x : Represents the distance from the region of interest and increases based on the step size

The calculated Gaussian kernel is rounded to the eighth decimal point and discretized to odd values for use by the Gaussian blurring function based on the 'Focus Array'. (Example: [1, 1, 3, 5, 5, 5, 7, 7, 7, 7], [1, 1, 1, 3, 5, 5, 5, 7, 7, 7],...,[7, 7, 7, 5, 5, 5, 3, 1, 1, 1], [7, 7, 7, 7, 5, 5, 5, 3, 1, 1] are arrays representing upside down Gaussian parameter arrays centered at the focus point. Here the value '1' means there is no blurring at those depth levels.)

3.3 Mask Generation and Look-up Table

This section involves the actual blurring operation and the generation of the look-up table for the output functions.

The first step, in either case, is the generation of a binarized mask for each of the depth zones. Binarization is achieved by determining the range of values of the depth map array and segmenting them into the required 10 depth zones based on the calculated step size. We eventually use these binary masks to reconstruct our final output.

The generated masks are then up-scaled using interpolation and are duplicated to form triple channel arrays to be used with the respective R, G, and B channels of a standard image.

Simultaneously, to differentiate between the different depth zones, the generated masks are recombined in a second variable by assigning a value individually from 0 to 9{0, 1, 2,..., 8, 9} and this produces the look-up table.

The original image is blurred based on the previously generated Gaussian kernel values to create 4 master blur images (as seen in figure 7 below) per frame to be used when reconstructing the 10 possible focus levels.

The masks representing the depth zones with the same amount of Gaussian blur are recombined and multiplied with the respective Gaussian blurred image this is shown in figure 8

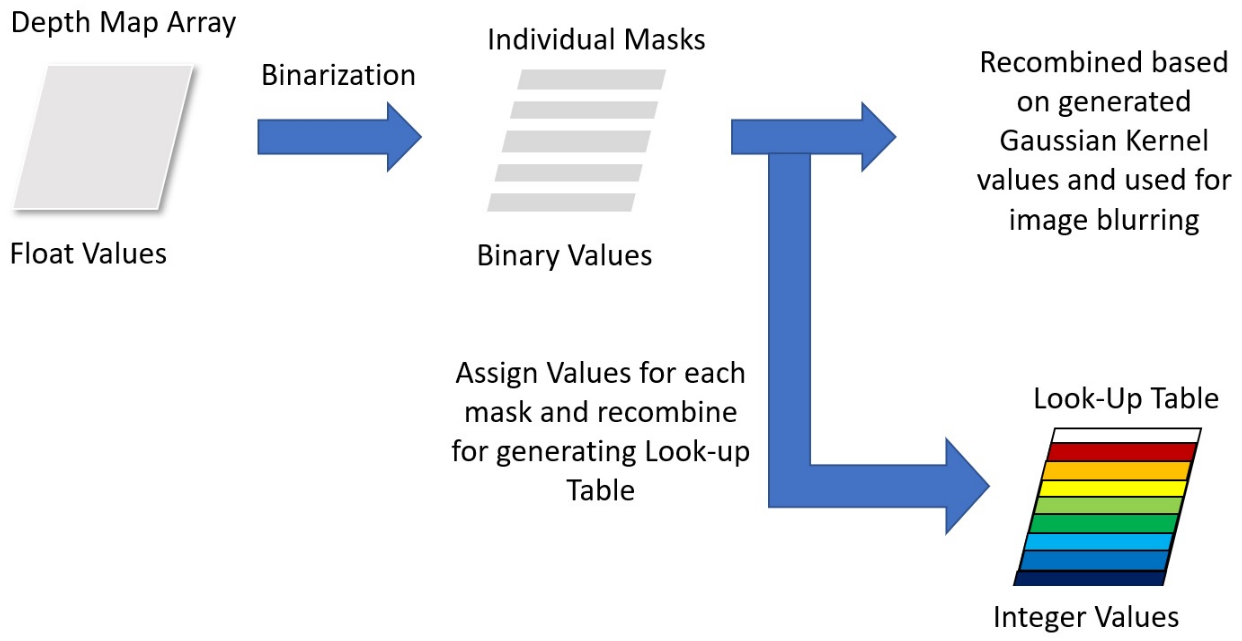
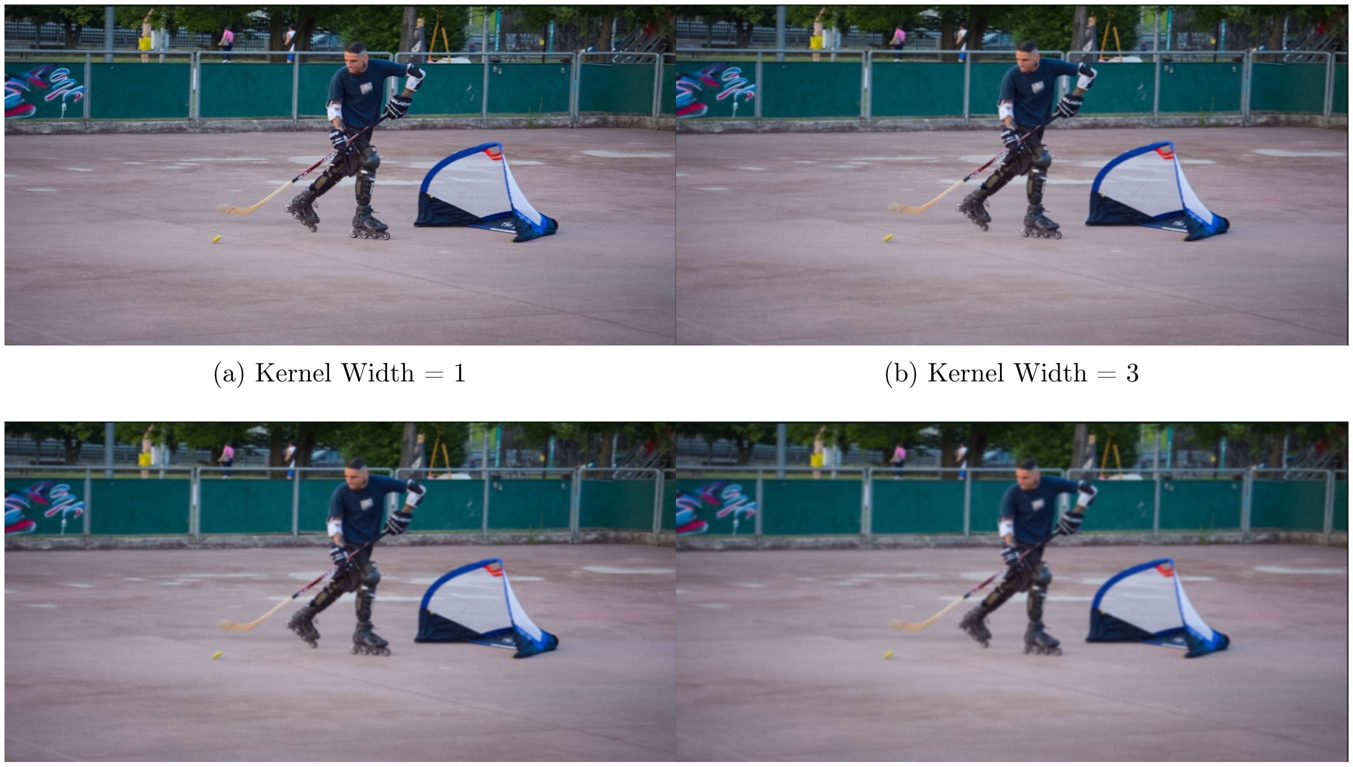

Depth Map Array
Float Values
Binarization
Individual Masks
Binary Values
Recombined based on generated Gaussian Kernel values and used for image blurring
Assign Values for each mask and recombine for generating Look-up Table
Look-Up Table
Integer Values

Figure 6. Basic workflow of the Mask and Look-up Table generation



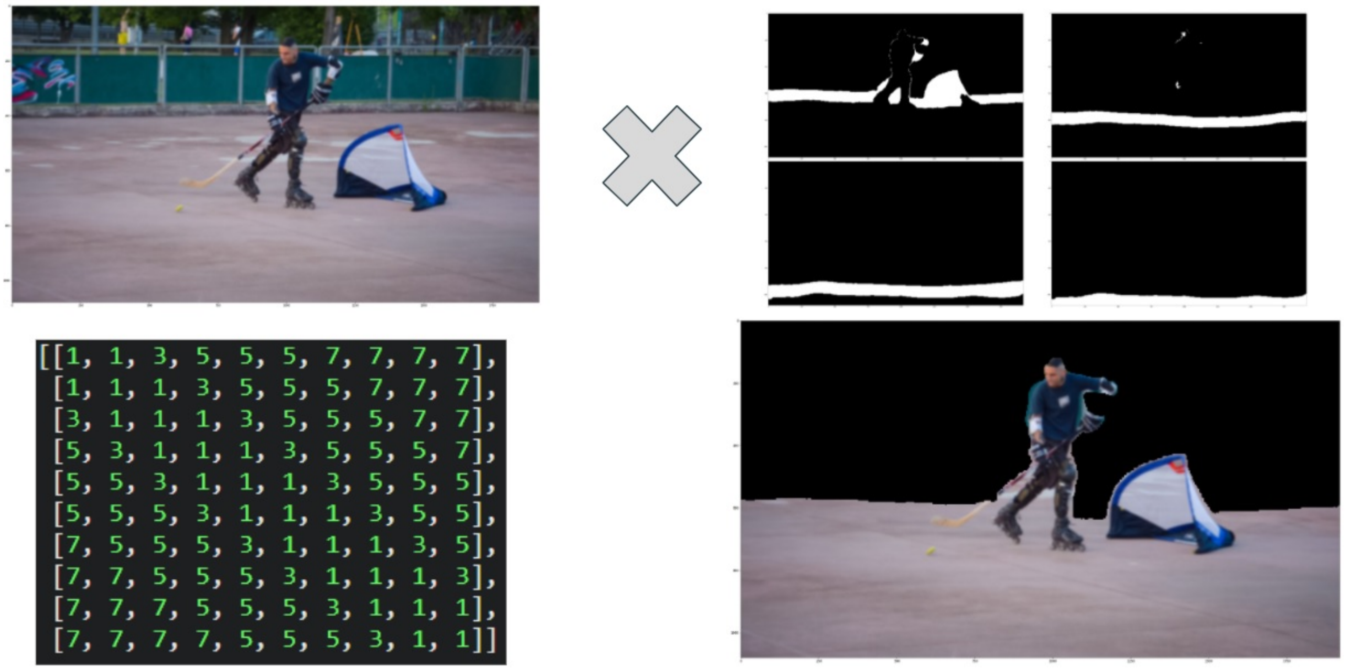
(a) Kernel Width = 1

(b) Kernel Width = 3

(c) Kernel Width = 5

(d) Kernel Width = 7

Figure 7. 4 Main Blurred images representing the 4 blur levels



```
[[1, 1, 3, 5, 5, 5, 7, 7, 7, 7],
 [1, 1, 1, 3, 5, 5, 5, 7, 7, 7],
 [3, 1, 1, 1, 3, 5, 5, 5, 7, 7],
 [5, 3, 1, 1, 1, 3, 5, 5, 5, 7],
 [5, 5, 3, 1, 1, 1, 3, 5, 5, 5],
 [5, 5, 5, 3, 1, 1, 1, 3, 5, 5],
 [7, 5, 5, 5, 3, 1, 1, 1, 3, 5],
 [7, 7, 5, 5, 5, 3, 1, 1, 1, 3],
 [7, 7, 7, 5, 5, 5, 3, 1, 1, 1],
 [7, 7, 7, 7, 5, 5, 5, 3, 1, 1]]
```

Figure 8. Figure represents the mask recombination step for the first of ten focus array's 4^{th} region represented by the value 7

3.4 Output Functions

The output functions are split into two main parts, viz.:

1. Mouse and Eye Tracking
2. Display or Frame Output
   This component is further divided in to the 'main' display function and the 'preview' display function with minor changes in functionality

3.4.1 Mouse and Eye Tracking

The output uses the 'openCV' event tracking function 'EVENT_MOUSEMOVE' to obtain the latest known position of the mouse pointer within the 'openCV' window. The function is used to determine the current x and y coordinates of the mouse pointer within a given window. The monitoring of the 'mouse move' event is initialized within the display functions along with the creation of an output window and is latched to said window.

The eye-tracking is achieved using the Tobii Eye Tracker 5 and the included software, the eye tracker is used to snap the mouse pointer to the region of the screen where the user is focused on without having to physically move the mouse.

3.4.2 Display Function

The software allows the user to preview the amount of blur per region and the difference in blurring between the 4 regions, hence the presence of two display functions in the code.

The display function performs the following tasks:

1. Initialize the mouse pointer location to the origin
2. Create an instance of the output window
3. Latch the 'mouse move' event to the output window
4. Initialize the frame variables including the first focus region, the frame counter and the frame itself
5. Begin playing the associated audio file
6. Display the frames continuously using the available 'imshow' and 'imread' function

The output function continuously checks for a keyboard interrupt to close the output window.

The principal difference between the two display functions is the check for availability of output folders and the code to loop through all frames in the main display function and the lack of audio playback in the preview function.

3.5 Initial Approach

Figure 9 represents the initial approach undertaken to create a cinematic depth of field

In the current solution, the process that has changed corresponds to steps 3 through 5. This approach represents a process for real-time accurate Gaussian blurring to achieve the cinematic immersion, it presented some initial success in rendering the cinematic Depth of Field frames at 25 frames per second(FPS) but this was limited to resolutions at 480p and lower.

Due to this solution's limitation on the video resolution and hardware dependencies, the solution required a rework to make it suitable for use with a minimum of 1080p resolution videos.

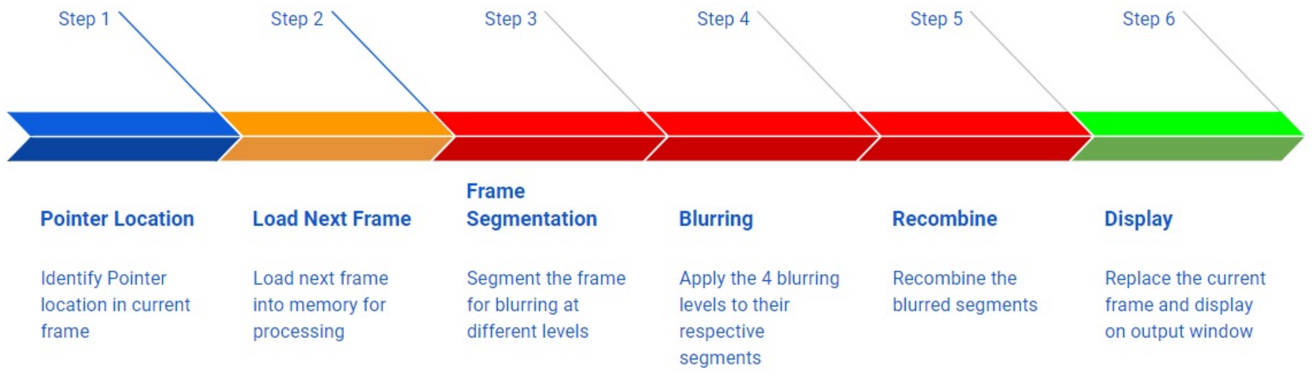

Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Pointer Location
Identify Pointer location in current frame
Load Next Frame
Load next frame into memory for processing
Frame Segmentation
Segment the frame for blurring at different levels
Blurring
Apply the 4 blurring levels to their respective segments
Recombine
Recombine the blurred segments
Display
Replace the current frame and display on output window

Figure 9. Figure represents the initial design considerations for the code implementation of the project

3.6 Trade-off/Optimizations

The design choices were made on the Trade-off considerations between the three components of Processing Power, RAM, and Long-term storage, hence the project prioritized the end user's viewing experience using the final processed video over real-time Gaussian blurring.

Some of the major design considerations are shown below

3.6.1 Gaussian Blurring

Figure 10 represents the time taken for Gaussian Blurring at various kernel widths for a frame at 480p resolution.
The Gaussian blurring represented here uses the exact or perfect Gaussian blurring as opposed to an approximation. We see from figure 10 that, at smaller kernel widths convolution-based Gaussian blurring is faster and suits the needs of the project, but the time taken at higher kernel widths grows exponentially. Oppositely, the FFT Gaussian blurring maintains a relatively constant time taken across the various kernel widths, but the overhead involved in converting the image into the Fourier domain and performing the inverse Fourier transform will only show a noticeable benefit at larger kernel widths or larger image sizes. The difference in performance in terms of using built-in functions or coding the function is negligible as the built-in functions are significantly optimized. The smaller kernel widths are suitable for a smoother blurring and a more pleasing visual experience, thus the project uses convolution based Gaussian blurring.

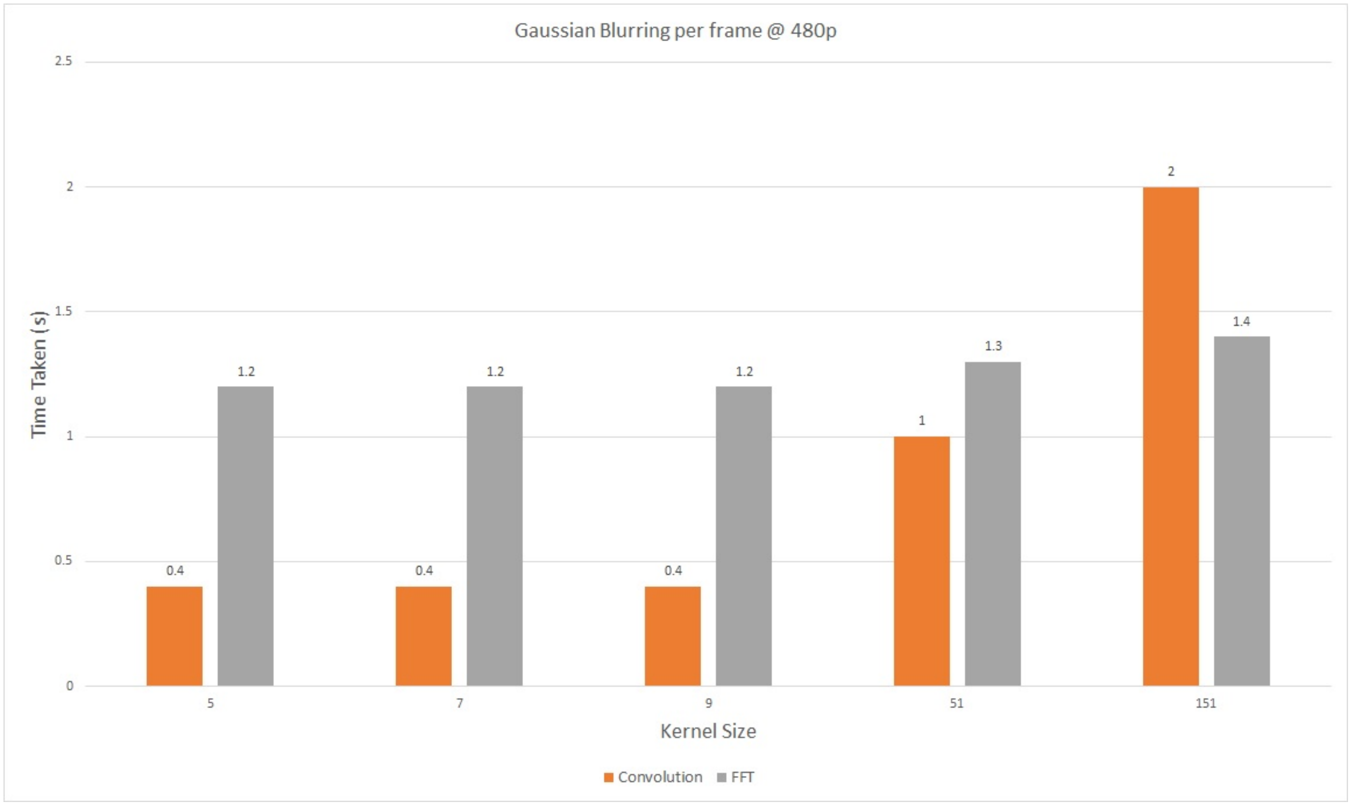


Figure 10. Time taken for Gaussian Blurring at various kernel widths for a 480p resolution frame

3.6.2 RAM

Figure 11 represents the Random Access Memory(RAM) utilization per frame at 480p and 1080p resolution during processing. The frames used during testing averaged 150KB for a 480p frame and 350KB for a 1080p frame. The large memory utilization can be attributed to the overhead involved in processing each frame, which includes the loading of the original frame(s), calculating the kernel widths, and holding the processed 10 images of each frame in memory before writing to long term storage.

Figure 12 shows a comparison between storing a 1080p image and its corresponding depth map NumPy array in storage compared to in the RAM as a python variable. We see from figure 12 that the NumPy array does not occupy more space in the RAM than in storage, hence there are no advantages to be gained in using one over the other in terms of memory used. Oppositely, loading an image (.JPG format) from storage to memory(python variable) sees an average of 20 times increase in memory requirement. This jump in memory usage is not conducive for processing in real-time.

These characteristics limit the number of end users capable of processing the videos due to the large RAM requirement.

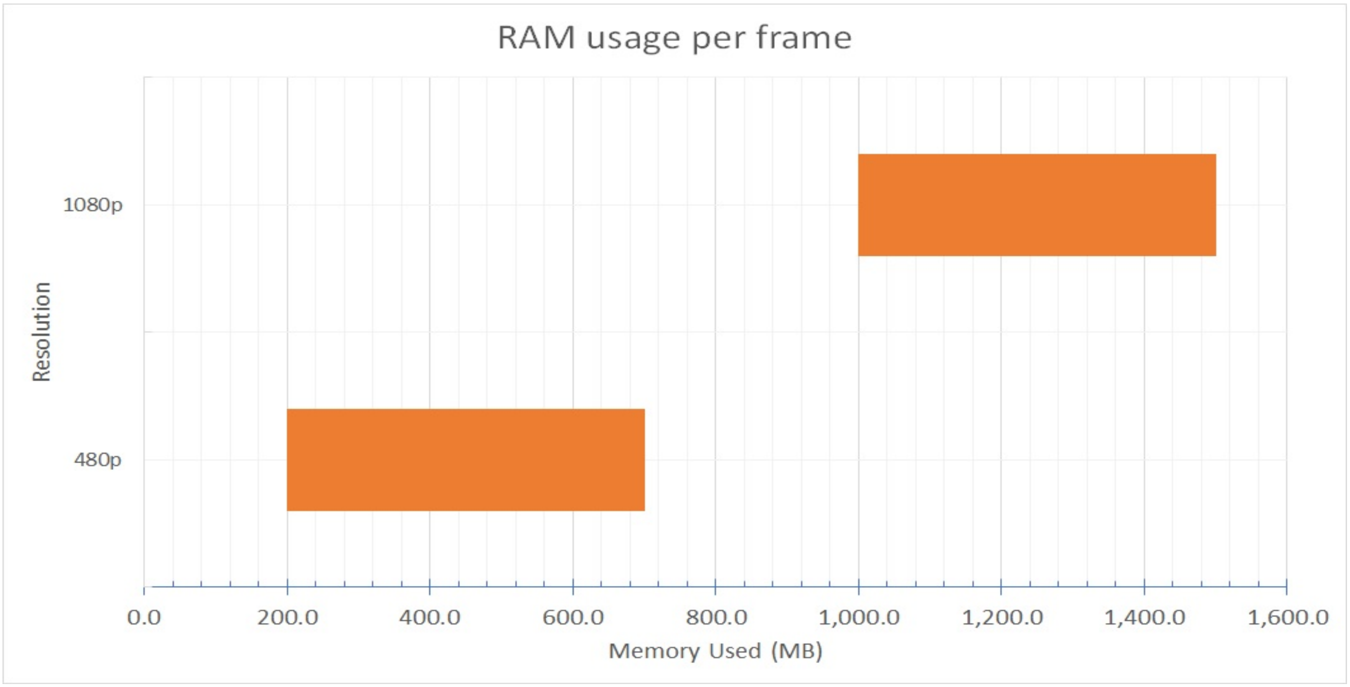


Figure 11. Memory usage by the software at 480p and 1080p resolution

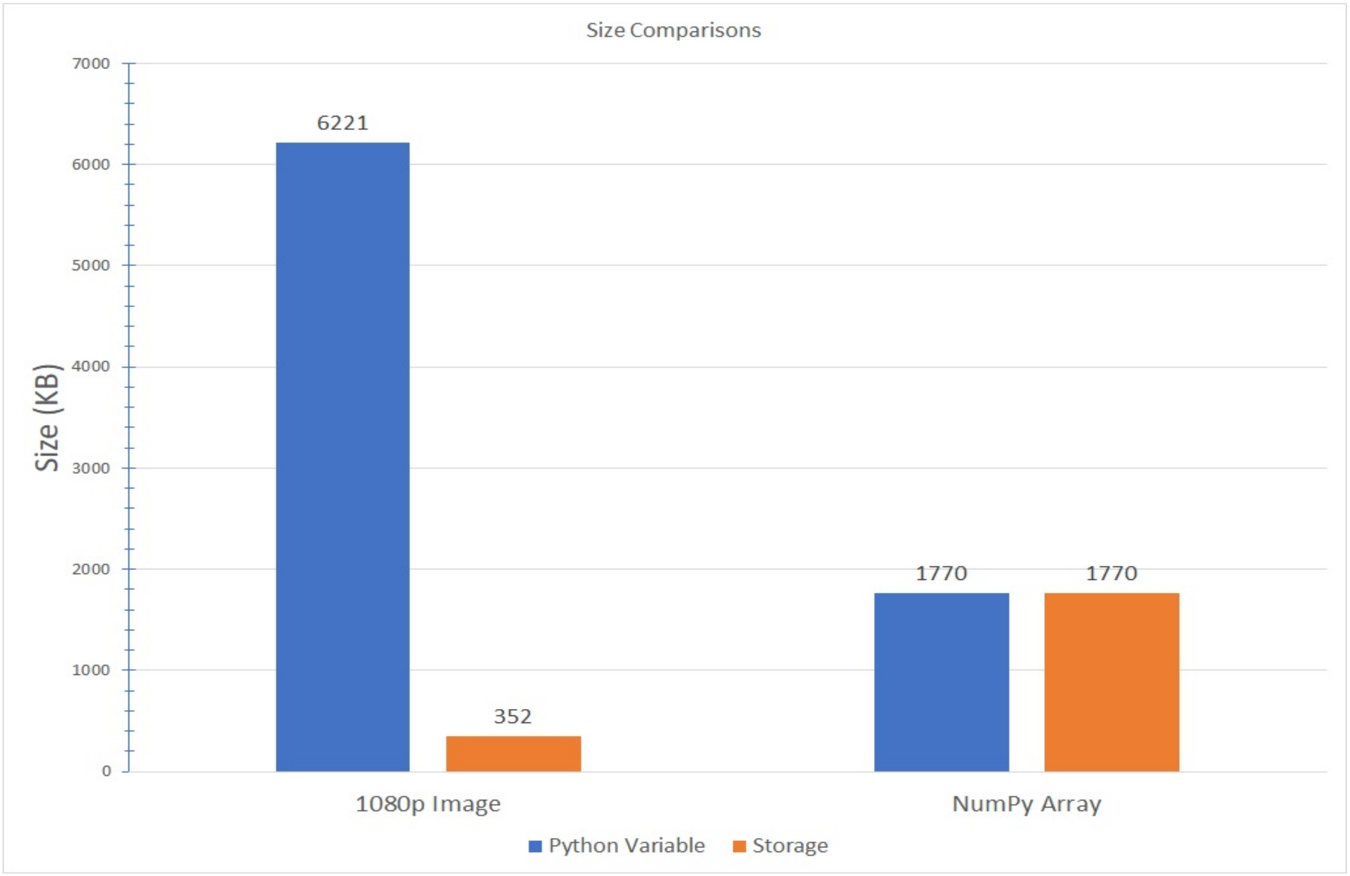


Figure 12. Memory used in RAM versus Long term Storage for an image and the depth map array

3.6.3 Storage

As seen in figure 13, by making the solution more dependent on the storage, we have enabled a larger number of users to be able to view the videos even if they are unable to process it. With this approach, the storage required increases on average by a factor of 10. The decision to make the solution more storage dependent was based on the ease of scalability of storage for the end-user, as opposed to the RAM or Processor.

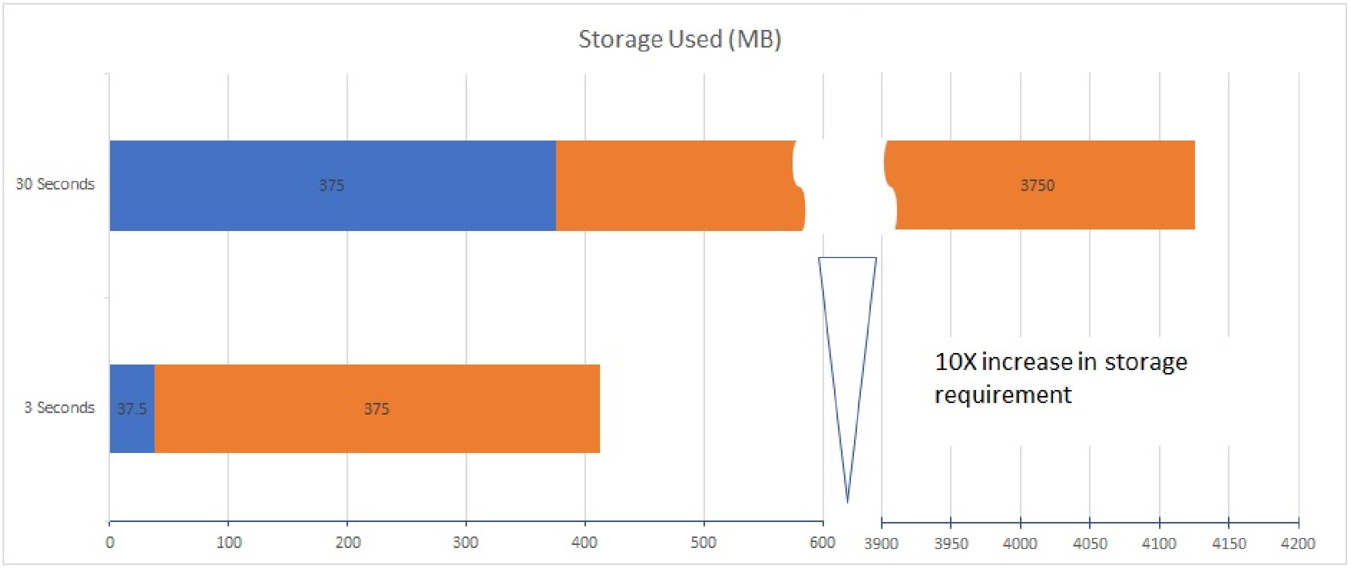


Figure 13. Long-term storage memory requirement for video increase by a factor of 10 for the current approach

3.6.4 Processor

Figure 14 highlights the sharp decrease in the number of frames processed as the resolution increases. A 5 to 7 times increase in resolution from 480p to 1080p sees an average of 10 to 15 times decrease in the number of frames processed. The processing power is key for both real-time applications and pre-processed outputs as we need to ensure the frames are processed in a timely manner.
One of the options for reducing the processing time is to use multiprocessing, some experiments were performed with this implementation but encountered issues when trying to share a common memory space for synchronization between processes, both during the processing of the frames and displaying the frames for the user. Hence both the original and the current solution use a serial approach of processing the frames, where all the required input data is first loaded into the RAM for shorter access and read times.

The current approach is thus influenced by these factors in generating the Cinematic Depth of Field.

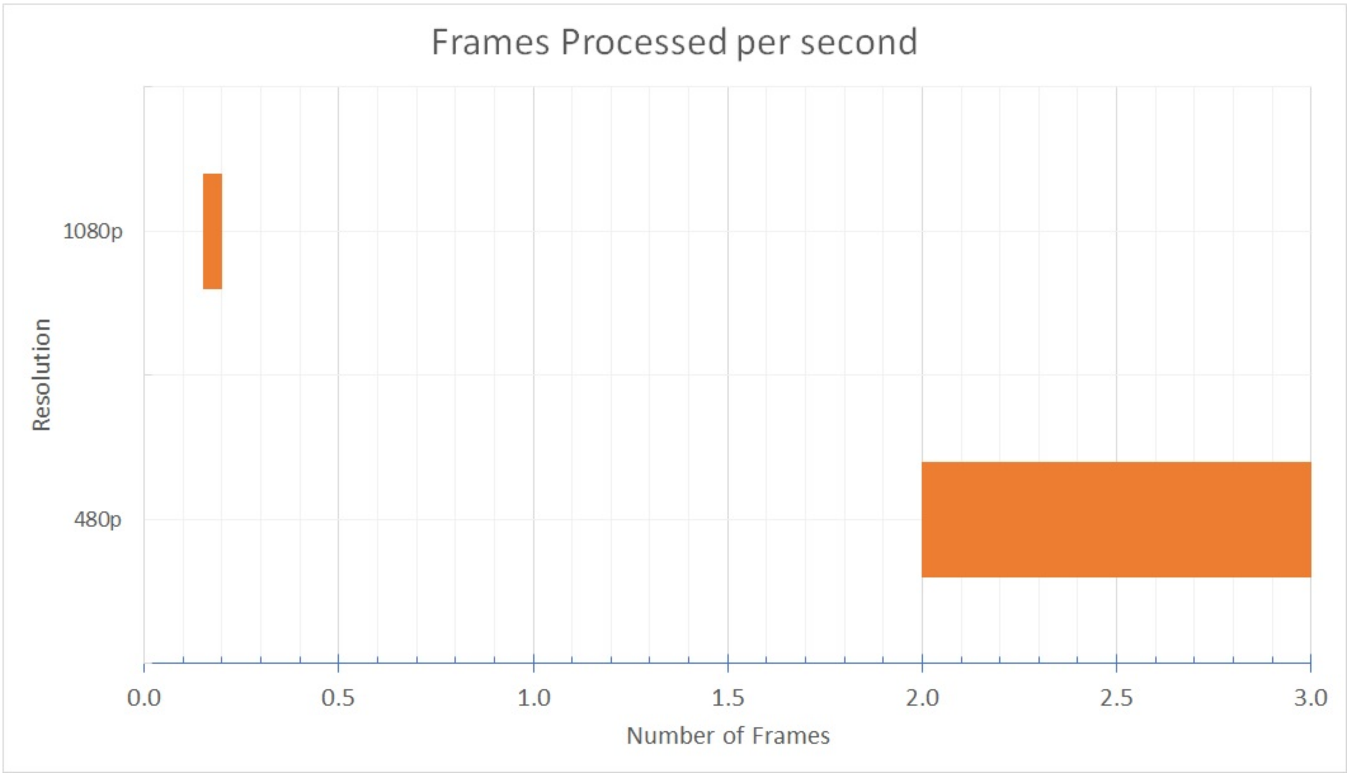


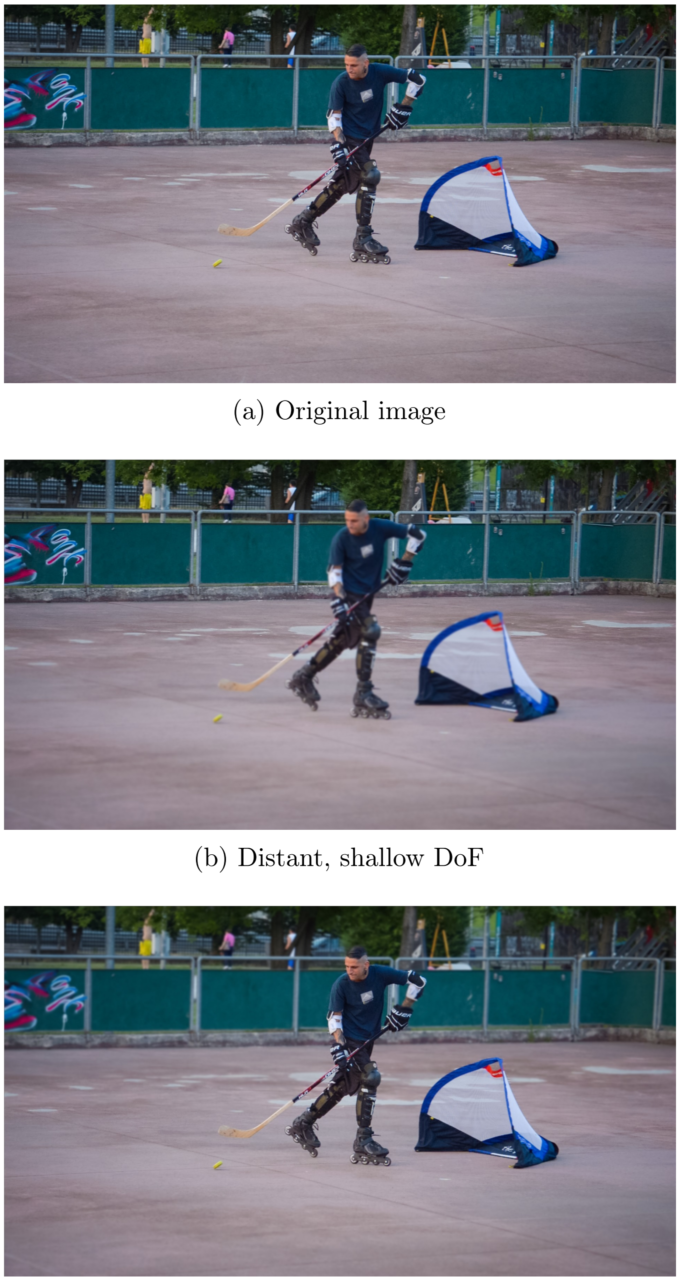
Figure 14. The graph highlights the drop in number of frames processed as resolution increases

The code design was influenced by the caveat that a larger number of users should be able to view the video even if they can not process a video. Hence, there are hardware restrictions in processing a video necessitating a larger memory(RAM) capacity and a powerful processor to be able to perform the video processing in a reasonable amount of time. Though with this approach the long term storage requirements for a video increases by a factor of at least 10. To overcome this drawback of increased storage requirements while simultaneously not taxing the processor and RAM, a shift to an approximation of the Gaussian blurring [7] is required. This process will ensure that we can achieve the desired blurring characteristics on a frame in real-time (0.4 seconds) to achieve a frame rate of 25 FPS.

4 RESULTS

As can be seen in figure 15, the original image is modified by our program to have an adjustable DoF. In figure 15 (b) the background is in focus while the hockey player is out of focus, while in figure 15 (c) the opposite is true. The full demo of our project can be found on our GitHub below.

https://github.com/nawaldua/Dynamic-Depth-of-Field-with-Eye-Tracking



(a) Original image

(b) Distant, shallow DoF

(c) Near, shallow DoF

Figure 15. Changing DoF based on cursor/eye position

5 FUTURE WORK

Our project may be complete, but it can still be improved greatly. Most importantly we need to work with a higher tier of eye-tracker. The eye tracker we purchased (Tobii 5) was a commercial use eye-tracker as opposed to an eye-tracker meant for software development or research. Due to this, we were limited in the SDK and data we had access to. The Tobii 5 only had a function to snap the cursor to the current gaze position by pressing a hotkey. This meant that we had to keep pressing the hotkey as we looked around. With an eye-tracker that gives access to gaze data, we can use the gaze coordinates to know where to focus the DoF similar to how we used the cursor coordinates.

Other improvements can be divided up into 3 main areas: Improve Depth Estimation, Add Parallax effect, and Code improvement and optimization.

5.1 Improve Depth Estimation

Initially, we tried improving on Google's depth estimation model using a histogram-based approach. However, this approach is currently limited to videos with stationary backgrounds. There is still a lot of work to be done in this direction to obtain acceptable results. As you can see in figure 16, there was some tearing and misclassification in the depth maps.

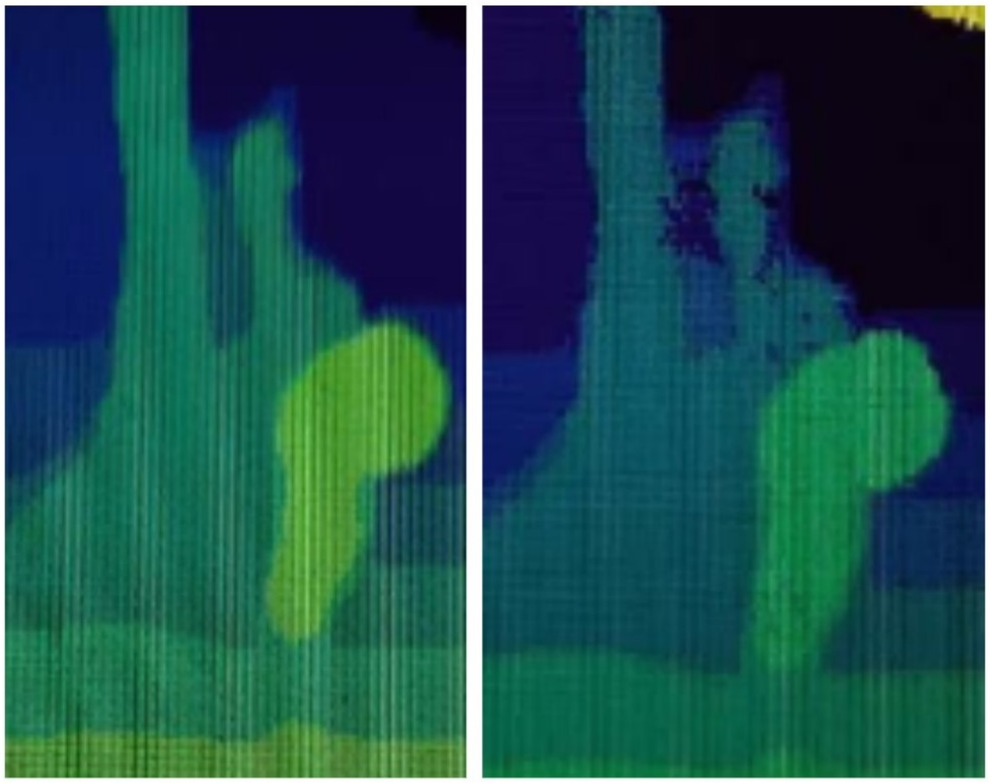


Figure 16. (left) Depth map from Google's model. (right) Depth map from histogram improvement.

Some other ways of improving the depth estimation would be to find other deep learning models or to use a non-deep learning approach to depth estimation. Facebook claims to have a model that performs better than Google's, however, their model and code are not available to the public and hence cannot be used for our project until then.

5.2 Add Parallax Effect

On top of your eye position, eye trackers can also detect the orientation and position of your head. We plan on leveraging this to add a parallax effect to our system. This would make it so that as the user moves their head and angle of gaze in regard to the screen, objects in the video that are deemed to be closer to the foreground (have less depth) will move a lot, while objects further back (having more depth) will barely seem to move. Work on technology like this has been done but not in combination with changing Depth of Field. Facebook has made 3D photos that add parallax to them based on the angle of the smartphone and position of the image on the screen [9]. Technology like this can be integrated with eye/head position data to add parallax to our project.

5.3 Code Improvement and Optimization

One way to improve our code in terms of performance is by optimizing it for time and space complexity.

Right now our project blurs the image using perfect Gaussian blurs. This algorithm blurs remarkably but has a time complexity of $O(n \cdot r^2)$. Ivan Kutskir has found a way to approximate Gaussian blurs with an error of only 0.04% per pixel but with linear time complexity, i.e. $O(n)$ [7]. This would make our project much faster and may provide us with the ability to process real-time, rather than having to preprocess and store our outputs.

6 CONCLUSION

Within the time frame of our summer semester, we have developed a system that has met our basic requirements of generating refocusable videos with changing depth of field with eye-tracking. There are still improvements to be made in depth estimation and memory optimization, and we plan on taking the project forward after this semester to get a more complete product. A promising direction the system can go in is with the addition of parallax to get a truly immersive experience.

7 ACKNOWLEDGEMENTS

Neel: Conceptualized the project. After a literature survey, found and implemented Google's depth estimation model. Created a method to improve depth estimation. Tested numerous blurring and masking methods. Worked on improving memory optimization.

Suchith: Supported the development of the code. Worked on Google's depth estimation model analysis and adaptation, Gaussian blurring implementation, Optimization of code and impact analysis. Took lead on development and optimization of the Output functions, Mouse Tracking, GUI and Audio implementation.

Nawal: Contributed ideas throughout the project. Worked on trying to get eye-tracking integrated into the project. Communicated with Tobii representatives to get information about SDKs, eye-trackers, and licenses. Worked on Intro, Depth estimation,

We would like to thank Professor Matthew Malloy for his invaluable guidance and we would like to thank our fellow students for helping us learn and grow together over the semester.

REFERENCES

[1] In: Lexico, powered by Oxford (). URL: https://www.lexico.com/definition/depth_of_field.

[2] In: Lexico, powered by Oxford (). URL: https://www.lexico.com/definition/parallax.

[3] Jeff Conrad. "Depth of Field in Depth". In: 2004, pp. 12–13. URL: https://www.largeformatphotography.info/articles/DoFinDepth.pdf.

[4] Bill Dobbins. "THE MYTH OF DEPTH OF FIELD". In: 2019. URL: https://onphotography.me/2019/02/17/the-myth-of-depth-of-field.

[5] Bill Freeman et al. "Learning the Depths of Moving People by Watching Frozen People". In: 2019.

[6] "How the eye focuses light". In: Science Learning Hub (2012). URL: https://www.sciencelearn.org.nz/resources/50-how-the-eye-focuses-light.

[7] Ivan Kutskir. "Fastest Gaussian Blur (in linear time)". In: Algorithms and Stuff (2014). URL: http://blog.ivank.net/fastest-gaussian-blur.html#results.

[8] Laviolette. "Depth of Field". In: 2017. URL: https://sites.google.com/a/ocsb.ca/teh-2016-awq2or-3/calendar/week-9-and-10-exposure.

[9] Maggie Tillman. "Facebook 3D photo: How to create and view perspective-shifting pics". In: Pocket-lint (2018). URL: https://www.pocket-lint.com/apps/news/facebook/146017-facebook-3d-photos-how-to-create-share-and-view-3d-photos.