Immersive and Interactive 360 Video Editing for Virtual Reality

Connor Nobbs

Abstract— **360 videos are an increasingly more common way of viewing content on the internet. They also pair very well with virtual reality, as VR offers a higher level of immersion that suits the video format when compared to a standard computer monitor. The usage of virtual reality is also on the rise, both in entertainment and education. Logically, editing 360 videos in virtual reality would offer this same immersion, potentially allowing for better edited videos. Current video editing tools are either not available in virtual reality or only offer limited functionality. Many of these tools also do not handle the unique challenges that editing 360 videos pose. Making edits with these limitations can lead to interrupted workflows and mis-edits. During this project a prototype interface has been built. This prototype allows for pixel-wise 360 image and video editing, previously untested in virtual reality. The project was built in Unity engine and uses two C++ implementations of video editing techniques: colour editing that differentiates between foreground and background, and a usage of optical flow that allows a user drawn piece of 'graffiti' to remain consistently placed between video frames. User testing has been done to test both the user experience and effectiveness of the prototype, as well as getting aesthetic opinions on the video edits. This user testing has been anonymised and it, as well as limitations of the system have been documented, giving a clear path for future work.**

Index Terms— **360 Video, Video Editing, Virtual Reality**

I. INTRODUCTION

The problem and motivation

Three-sixty degree video is steadily becoming a more popular form of entertainment and education. 360 videos offer more information (the full 360 view) and are more immersive than traditional 2d videos. 360 videos have now become freely and easily available on major platforms such as YouTube and Facebook. 360 videos are being used and shown in a variety of ways from teaching technical skills, to experiencing tourist locations without travelling, to film festivals.

In addition, the number of people using VR is steadily increasing. As of 2022 there is estimated to be over 171 million VR users worldwide[1] and there were over 16 million VR headsets shipped in 2022[2]. Meta, ByteDance and Sony are the largest companies in the VR/AR sector[3]. Other companies such as HTC and Apple are either preparing to release headsets or are building new versions of their current models. VR is used for a wide variety of tasks including education, gaming and training in healthcare, the military and the trades.

With the increased consumption of 360 videos, users need ways to create and edit these videos, quickly and easily. This project focused on the latter. 360 videos are typically stored in a 2d format meaning that they can be edited in standard video editors. But there are limitations to this process. When 360 videos are stored in 2d, they become distorted, meaning that the view of the video can be misleading to the user, leading to mis-edits.

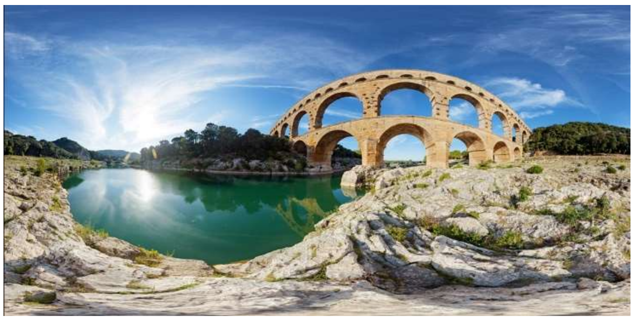

Fig. 1. An equirectangular image projected on a 2d plane demonstrating the warping of such a projection.

2d videos have 4 defined edges and no wrapping across borders. 360 videos do have wrapping on all sides, meaning that 2d video edits will not work correctly when a video edit reaches a border. Also, when cutting between different 360 videos the two clips must be rotated properly so the transition is smooth, and the user's view remains consistent, this is something completely unique to 360 videos. Finally, 360 videos cover the complete field of view of a scene and so they must have a much higher resolution to have the same viewing quality as a 2d video that only looks at part of a scene. Facebook recommends a 4096 by 2048 resolution for monoscopic 360 video[4]. The Meta Quest 2, a very common headset offers a resolution of 1832 by 1920[5] per eye so using high resolution videos is possible and recommend. As the field of view of these videos is so large the video quality ends up much lower than a standard video of the same resolution. So much larger videos are needed to keep the same viewing quality. Consequently, video editing techniques must be even further optimized than 2d techniques so that edits can still be made in a reasonable timeframe.

There are a range of video editing tools available that allow 360 video editing. Both Adobe Premier Pro and Apple Final Cut Pro can handle video edits for 360 videos and handle many of the previously mentioned challenges posed by 360 videos. Both of these tools however, are still used through a traditional 2d screen. To view your edits free of distortion, you must put on a virtual reality headset, check the changes, take the headset off, and repeat this for every change. Other issues with this process,

This project was supervised by Fang Lue Zhang

taken from interviews in [6] were having to regularly export titles to external pieces of software to alter them to work within 360 video and incorrectly aligning a clip would lead to incorrect alignment propagating down the entire movie. All these issues slow down the workflow and lead to fewer and potentially lower quality videos being produced.

There is one system, created by Adobe that allows users to make video edits in VR. This system is only able to make a limited number of simpler video edits, more complex edits must still be made through a 2d screen.

Solution

The solution is to build a complete system that allows a full range of 360 video edits to be made directly in VR. This was far beyond the scope of this project so instead a system was built that allowed for user input and video editing that previously had to be done through a 2d screen, to be done in VR instead. This user input and video editing was all pixel-wise editing, namely edits that require individual pixels to be selected and manipulated. Four main requirements were defined for this project:

- Build a system that allows pixel-wise video edits to be made to 360 videos in VR. Pixel wise edits are the unique video editing techniques that the project was concerned with so the system must be able to handle them entirely in VR.
- Implement at least two video editing techniques to use with the project. These would allow the system to be thoroughly tested. They will also be useful research in their own right, handling some of the issues discussed with applying 2d video editing tools to 360 videos.
- Build a system that allows additional video editing techniques to be easily added. This system will not complete and so building it in an open fashion will make future development easier and make it simpler for researchers to use the system with techniques they wish to test.
- Collect feedback on both the system and aesthetic opinions of 360 video. Collecting feedback on the usability of the system gives future developers a good starting point on where to improve the system and collecting feedback on aesthetic opinions will help validate (or potentially invalidate) further research in this space.

II. RELATED WORK

Benefits of 360 video

360 video and VR work very well together, creating experiences far more immersive than a standard computer setup. Research has shown that 360 videos of family/friends/loved ones 360 videos seen through VR can lower students negative emotional affect [7]. 360 videos, viewed in VR were used to teach participants how to tie knots in[8]. The participants learning with VR and 360 video were more likely to learn the skill than those taught with 2d videos. Finally, the study in[9] compared two groups of medical students observing gentle Caesarean Sections, one group observing in person and one through 360 video. The two groups did not have a significant difference in knowledge gained, meaning that 360 video can potentially be used for remote learning of these procedures or similar ones.

Usage of VR for detailed tasks

For pixel-wise video edits accuracy is an important factor otherwise video edits will not turn out as intended. The experiment run in[10] tested participants ability to select regions on a 3d object using either a mouse, hand tracking in virtual reality or a virtual reality controller and stylus setup. They concluded that using hand tracking is the worst option due to the hand detection being inaccurate. Using a mouse was the most accurate but has limitations since it only has 2 dimensions of input.

Vremiere and CloverVR

In 2017 Adobe released CloverVR, an interface for Adobe Premie**r** Pro that allows some video editing directly in virtual reality. This was likely built off Vremiere[6], the presenter for the 2016 preview [11] of CloverVR was one of the authors of the Vremiere paper. CloverVR is an interface that allows users to make edits such as cutting between clips, trimming videos, and rotating 360 videos so they line up when transitioned to. CloverVR does not have the full functionality of Premier Pro so other, often more complicated edits must still be made in the standard 2d application.

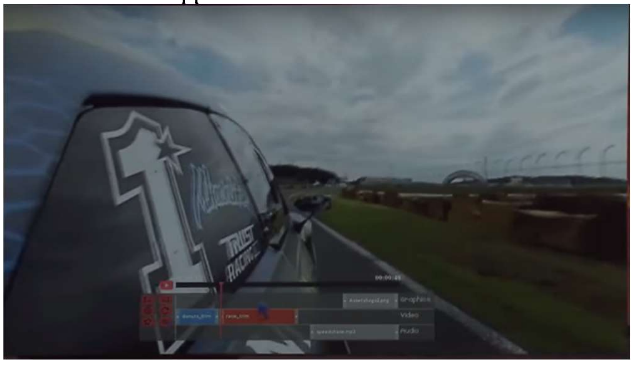

Fig 2. The users view in CloverVR showing the videos timeline interface at the bottom.

III. DESIGN

Requirements

This project is a system that allows users to make video edits to 360 videos directly in VR. It consists of an interface that allows user input and viewing 360 videos, and video editing techniques created primarily to demonstrate the functionality of the system and for effective user testing.

The project had 4 main goals all of which were achieved:

Build a system that allows pixel-wise video edits to be made to 360 videos in VR:

The system is very far off being a polished or commercially viable project, but a user can successfully view a video, select pixel regions of the video, input other data such as a color value, make the video edits and view the edited video only using a VR headset and controller. There are limitations to this that are discussed in the limitations section of this report.

Implement at least two video editing techniques to use with the project:

Two video editing techniques required for the project were successfully built, one that tracks a user drawn piece of graffiti throughout the video and another that separates the foreground and background of a video and applies color edits only to the background. These are both pixel-wise video editing techniques, other techniques like cutting a videos length would have been possible but making techniques to test the unique features of this project was a logical choice.

Build a system that allows additional video editing techniques to be easily added:

The interface has been kept as separate from the video editing techniques as possible. The two techniques integrate with the interface as executables. Future techniques can be added in a similar style, requiring about 10 lines of code to be added to the interface to handling calling the executable. This adaptability has already been partially integrated with another piece of research. If a technique requires user input beyond what the system offers, much more work will need to be done to allow for this.

Collect feedback on both the system and aesthetic opinions of 360 video:

User testing was carried out at the end of the timeline. Participants were taken through a use case using the two video editing techniques and then answered interview questions after. The interview questions covered details and feedback for the system, the usage of VR for 360 video editing in general, aesthetic opinions of 360 videos (particularly the edits participants made) and how participants would use them.

Tools

While these are technical choices all these tool decisions were made early in the design phase and so affected other design choices. Hence why they are included here.

The interface was built using Unity Engine and C#. Using a game engine simplified development greatly and saved time. The choice of Unity Engine as opposed to its competitors was due to Unity being fairly simple and it having thoroughly vetted VR libraries. The video editing techniques were made in C++. The OpenCV library was well used for image processing as it is very powerful. OpenCV is available for Python, C++ and java. C++ was used due to prior experience with it and to optimize the runtime of video edits as much as possible. The hardware used was a Quest 2 VR headset and controller connected to a Windows PC. This choice was simply due to availability. The Quest 2 is also a good choice due to its popularity so this project and future iterations will be built for a common headset. The benefits of any alternative setup weren't worth the time or budget. The additional PC is required as the Quest 2 does not have sufficient processing power to handle the system. So, tethering it to a more powerful PC was required.

Design

The first major design choice was how to display and store the 360 videos. 360 videos are stored in their 2d format and then projected onto a 3d object. There are two main ways to store 360 videos: equirectangular and cubemap formats. Equirectangular frames are projected onto the inside of a sphere, cubemaps onto the inside of a cube. Cubemaps are technically 6 square images but they are all stored in a single image to group them as seen in Fig. 3.

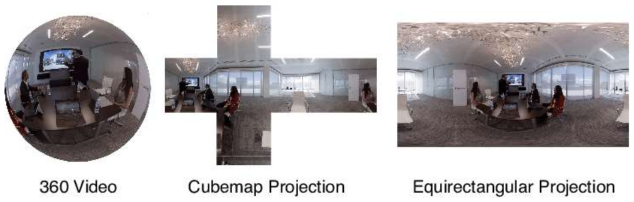

Fig. 3. An example of cubemap vs equirectangular images

This means that cubemaps can be stored in many different ways by shifting what faces of the cube are stored in what part of the image. Cubemaps store frames without distortion but have a more complex geometry to display, raycast to and make edits to. Finally, you can see in Fig. 3 not all borders are connected in the 2d frame, and the excess white pixels can also lead to mis-edits if complicated bounds checking is not implemented. Equirectangular images have a standard format when in 2d and have a simpler geometry in either of their forms, so this is the better choice. Finding publicly available equirectangular videos to use in the project was also quite easy.

Equirectangular frames are quite simple in either of their states but converting between the 2d and 3d representations of them is less so. Users of the system view the video in 3d and consequently select pixels in 3d. These 3d cartesian coordinates need to be converted to a 2d pixel so that selection masks can be created. 3d coordinates must first be converted to spherical coordinates (latitude and longitude) before being scaled and transformed. Finally mirroring and offsets were added so that the mask and the 2d equirectangular frame aren't out of phase.

Unity Engine has a limited frame rate and so user selection will only be picked up once per frame. A user drawing on the video will typically move the controller more than one pixel per frame leading to individual pixels being selected, not the lines of pixels the user has drawn. So, lines must be drawn between the current pixel and the pixel of the previous frame to make a continuous selection. The math for this is very common, with plenty of examples on the internet. Unity Engine does not have this functionality built into it however, so it still had to be written from scratch.

Pixel selection was inputted with the VR controller and a ray cast from it. The alternatives were using a mouse or hand tracking. The Quest 2 has the ability to track your hand instead of a controller and use various hand gestures instead of pressing buttons on the controller. Hand tracking is inaccurate as discussed in[10] and was immediately ignored. The decision to use a controller and not the more accurate mouse was made so as not to limit users. Using a mouse would force users to operate near a desk so they have a flat surface to use the mouse. It would also limit how much they could rotate. It is difficult to move a mouse and select pixels with it when you are facing directly away from the desk.

A lot of the interactions in the system are done through a menu. Components on the menu are selected with a ray cast from the controller, making it similar to clicking a menu with a mouse. Menus can be difficult to select with a VR controller. So, making a large and simple menu was important. The alternative was using various interactable objects around the area such as 3d buttons which activate upon virtual collision with a controller. Interactables are all unique GameObjects in Unity which makes creating them much more time consuming. They also take up more space meaning the view of the 360 videos would potentially be more obscured. Menus in Unity also have a lot of standardisations meaning that future developers of the system will have a much easier time adding new features.

A proper control schema is also important. Using as many user inputs as possible means that more functionalities could be removed from the menu. So, functionality such as pausing/unpausing the video, changing video frames and zooming in on the video (note zooming did not make it to the final implementation) were tied to buttons and other inputs on the controller instead of the menu. This also made user interaction faster as users did not have to go through the menu for certain functionality.

Another important design choice was how to incorporate video editing with the interface. Unity Engine uses C#, my video editing techniques do not for previously mentioned reasons. There is also no guarantee of what languages will be used for future video editing techniques. Python is a common choice here as OpenCV and Pythons machine learning libraries allow image processing models to be built easily. Building the techniques into a binary format before integrating them with the interface means that any language can be used for development, and it can still be called from the interface. This was a key step in making the system adaptable, fulfilling a project requirement.

IV. Implementation

The interface takes full advantage of Unity's VR libraries in order to work with the Quest 2 headset and controller used throughout development and testing.

The structure of the interface was re-done throughout development. The system was first built using the start() and update() functions Unity runs by default. This meant all checks and code were potentially being executed once per frame. A switch was made to state machine inspired code as the interface has clearly defined modes (main menu, paused menu and viewing the video). Some of this state machine stayed as the modes are still present in the final artefact. Finally, a lot of functionality was shifted to event-based systems. Events allowed for more separation between components and made the code easier to understand.

The equirectangular videos were treated as standard video textures within unity. They can be applied as materials to a sphere and project correctly by changing a few material settings. That being said a custom made sphere was still made using Blender. Blender was the modelling as it is free. The sphere itself has a higher than average triangle count to smooth the projection as well as inversed UV values so the materials is applied to the inside of the sphere.

There are two spheres in the project. A slightly larger sphere displays the video and the smaller one has the texture showing the selected pixels. This second sphere is what the ray cast from the controller collides with. Having this second texture is necessary so a separate texture can be laid over the video. Otherwise every time a pixel is selected every frame in the video would have to be changed to show this update which would slow the system down significantly. Updating the videos to show the selection would also change the videos themselves, unless copies were made. The selections themselves should not change the videos, only the edits made utilising the selections.

There are three separate layers you can make selections with. These form 3 three different masks which can be utilised by editing techniques. The three masks are names background, foreground and layer 3 but they can be used for any purpose. Using specifically 3 masks was a primarily a technical choice as it makes displaying them simpler. The selection texture that displays all the selections to the user in VR combines the 3 layers, displaying each layer as one of the red, green and blue channels. This gives a clear color distinction between the 3 layers, handles multiple layers being applied to the same pixel by simply having two non-zero values for that pixel and makes deleting layers easier. To delete a layer the given channel for that layer is set to 0 for every pixel.

Accurate pixel selection was another challenge. There was an attempt to implement a zoom feature. Offering a smaller field of view and enlarging it would allow users to draw more accurately. This feature was not of the highest priority and the time it would have taken to implement it meant it was not finished. The zoom was a small circle displaying the view of a camera closer to the video sphere than the users view. Selecting pixels with the zoom would mean raycasting to the sphere, taking the current angle of the ray then transforming to fit the not zoomed field of view. Then finally casting this second ray would find you the coordinate on the video sphere. Another smaller bug encountered when making the pixel selection accurate was the ray not being displayed in both screens in the VR headset. Only your right eye could see it and so mistakes would happen. This bug was quickly solved.

The other major feature that had to be scrapped during development was the smooth rewinding and fast-forwarding of videos. Videos in Unity Engine are handled through VideoPlayer components. These load and run videos and pass the texture of each frame to a specified material. In this case it was the material of the video sphere. VideoPlayers have a playback speed variable which dictates how fast the video is playing. Setting this variable to a negative value is platform dependent[12] and was not available. You can directly change the current frame of a video player allowing you to rewind more manually. This often is slower than restarting the video and letting it play however. So, the functionality to restart the video and fast forward (at slow speeds) was added.

The video editing techniques themselves were built from scratch with OpenCV, a computer vision and image processing library. The system comes with a 'demo' of 5 360 degree videos. These are publicly available videos provided by the projects supervisor. New videos cannot be added in during runtime, but the demo of 5 videos is not hardcoded meaning that new videos can simply be pasted into the Resources folder and the system will handle them.

The ability to use either the left- or right-hand controller for input was added late in the project. This functionality was added by now checking for both controllers simultaneously and assigning them to unique GameObjects. Additional logic was added to check button inputs from both controllers and decide on an output based on these. Most outputs were simple or statements.

The graffiti tracking takes a black and white mask of the graffiti as its primary input. It calculates the center pixel of the graffiti and then tracks this position throughout the video using OpenCV's implementation of optical flow. An example of optical flow tracking is shown in Fig. 4.

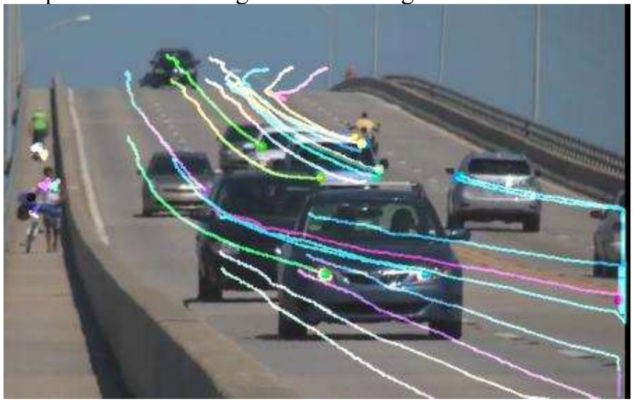

Fig. 4. A demonstration of optical flow. Each coloured line shows where a pixel has moved throughout the video.

The graffiti is then shifted according to the vector calculated. Finally, the graffiti is colored, based on a separate input color, and added to the frame. This technique has a runtime of about 15 seconds, most of the calculations on the image matrix were converted to pointer-based loops (instead of using .at()) which greatly improved the runtime. The choice to base all the movement on a single pixel does typically lead to inaccurate tracking. This was the best choice both for optimization reasons and to avoid an exponential increase of difficulty in development. Truly accurate tracking and mapping of the graffiti would mean handling affine or perspective transformations. Accurately guessing these many dimensional transformations only given 2d pixel coordinates and pixel colors is a challenging task[13]. Good optical flow for 360 videos specifically is also a quite recent topic and while work is being done to train accurate models[14]. These were not available for this project. Work was done to make this technique wrap around so that the graffiti could move over the edges of the 2d projection. The first attempt was to convert the equirectangular frames to cubemap frames. The edges of all the squares neatly fit together so handling the edge cases would be accurate. Once the graffiti was shifted for a specific frame it could be converted back into an equirectangular frame. Four separate attempts were done, to convert to and from cubemap frames and handle the maximum of 24 edge cases (depending on the layout of the cubemap images certain edges can be lined up in the matrix so certain edge cases can be ignored). This method was dropped due to the time spent and the lack of results, the code is still present in the project, just unused. The second way used only the equirectangular frames. The left and right sides of equirectangular images map directly to each other so you can handle edge cases by extending the left side with some of the right side and vice versa. The top and bottom can be approximately extended also. To extend the top of the image take a section of the top part of the images (rows 0-100 were used in this case), flip this section in both the x and y axis the attach it to the top. The bottom of the image can be approximately extended using itself with the same process. This extension is in the final code but does not function correctly. When the pixel being tracked by optical flow goes over a border the code does fail. This is a solvable problem, just requiring time.

The background-separation color editing takes a black and white mask of a rectangle as its primary input. This rectangle roughly defines the background area of the image. This rectangle is used by OpenCV's GrabCut() function to create a more accurate mask of the background.

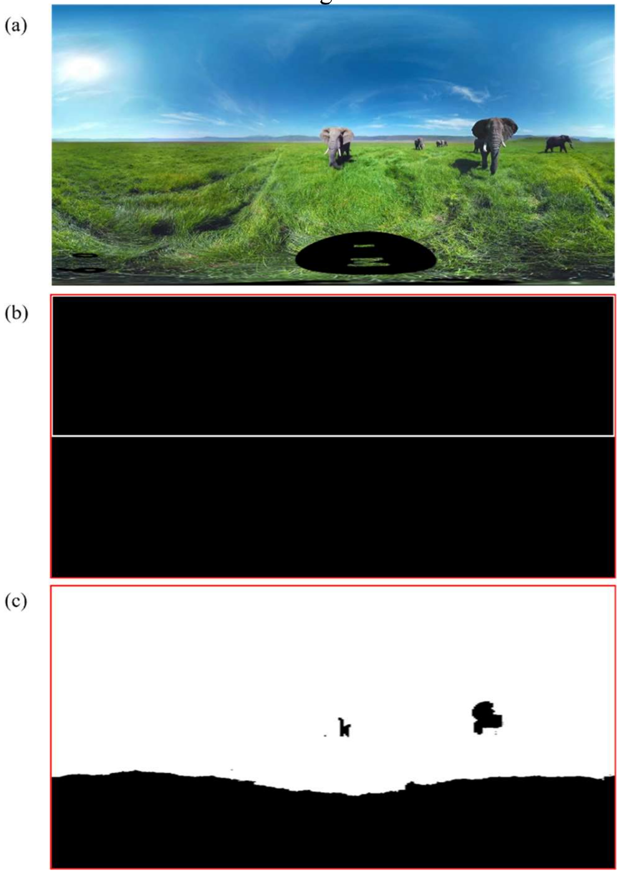

Fig. 5. A source frame (a), rectangle mask (b) and the separation mask (c) produced by GrabCut() given the two inputs. Red border in (b) and (c) added for clarity.

Then it applies a tint to the background region of the movie frame as defined by the calculated mask. As seen in Fig. 6.

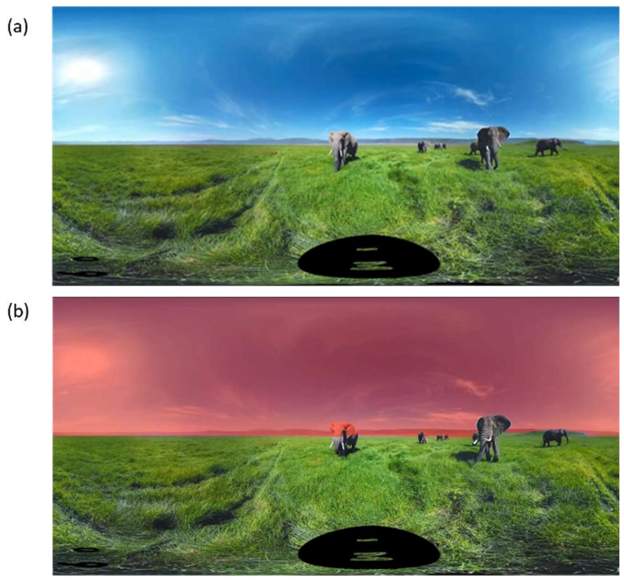

Fig. 6. A frame before and after being modified with the colour editing technique. Note how the largest elephants head is not changed despite it being in the background region.

Grabcut works in two steps: the initial run which takes a rectangle and calculates the background as best as it can, and then the user can mark parts of the image which the first run got wrong as incorrect, and the algorithm will iteratively build a more accurate mask. My initial code took approximately 8 minutes to run just for the first step of GrabCut(). The second step was not implemented (which does lead to some incorrectly colored parts of the movie) as the focus was on optimization. As with the graffiti tracking, whenever the matrix is looped pointers are used, but this only made small improvements to the runtime. GrabCut treats each frame of the movie completely separately meaning it can be run in parallel very easily. Multi-threading with six threads cut the runtime down to about three minutes. Finally, the frame and mask are now downsized before being passed to GrabCut() and then the mask produced is upsized again. This brought the runtime down to about 50 seconds, at the cost of a slightly more pixelated result.
The techniques were used in the final project as packaged executable files. These are called by the Unity interface as an external process as shown in Fig. 7.

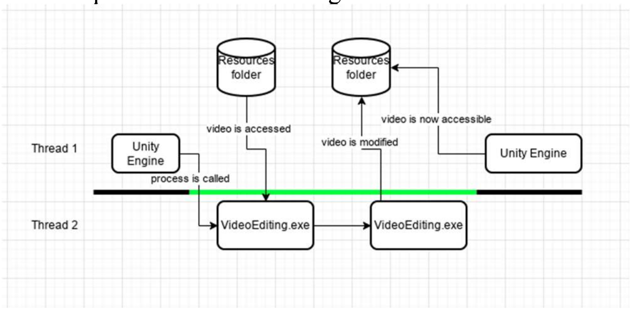

Fig. 7. Timeline of the video editing process

This was a clean way to bridge the gap between the two parts. This choice means that techniques do not have to be written in a specific language, giving future developers more options. It does limit the system to Windows, but within the university where this project will be used, non-Windows computers that have access to both Unity Engine and a VR setup are uncommon, so this is a non-issue.

V. SUSTAINABILITY

There are two scopes to discuss here: this project and a commercial VR 360 video editing tool.

This project has been built for future work, the requirement for adaptability has already been mentioned, significant work has been done to future proof it so that changes can be made on it well into the future. Recent versions of Unity Engine and OpenCV were used so that project will be supported well into the future. Work has been done to not lock the project onto a specific VR headset because each VR setup has its own ids for the buttons on their controllers. Within Unity these can be standardised meaning different controllers can be handled without making hard coded cases for each one. This standardisation likely applies to future controllers as well. This project was built with a Meta Quest 2 in mind but with the Quest 3 now available[15] and other headsets from various companies on their way, futureproofing this is important. These generics have not been tested due to a lack of varied hardware carry out the testing.

The current project uses a single controller for all its input meaning either hand can be used for the full functionality. As left and right controllers are mirrored this is also standardisable so users can use their preferred hand to make edits. This is all the accessibility offered with given the projects time and resources. A commercial product would have to consider accessibility much more seriously. The study run in[16] did an in-depth exploration of using VR with a range of impairments. Impairments such as involuntary body and eye movements, users limited range of movement, mobility aids such as wheelchairs and hearing/vision impairments were identified as accessibility barriers.

This is a system that does not scale, consequently its environmental impact is quite minimal, the optimizations made to the project are likely to have lowered its emissions slightly. Video editing is an intensive process, there is no getting around this, high end CPUs and GPUs are sometimes used for video editing, these units typically pull hundreds of watts of power. Minimising this power usage was done with optimisation and minimising the number of edits users need to make. Instructions to minimise the edits were given verbally during the interviews. A completed system would need to handle this within itself.

There are multiple economic reasons for 360 video editing in VR, as mentioned in my introduction. These are the primary motivators for this project, giving video editors tools to make their content as well and as quickly as possible will save time and lead to better final products.

VI. EVALUATION

Quantitative

The quantitative metrics are the runtimes of my video editing techniques. The goal was to get them as close to real-time edits as possible so users can see the results and make further edits as fast as possible, keeping the workflow smooth. The graffiti tracking technique was built in an optimized fashion from the beginning, using prior experience from CGRA352. The final runtime was about 15 seconds, with the library functions taking roughly 20% of this. There are likely more ways to optimize this further, but this would require much more additional skill and knowledge with C++. Also the diminishing returns of further improvements (considering the time required to make them) meant the runtime was left as it was. The background color editing originally had a runtime of 8 minutes with the library functions taking 95% of this time. As this technique treats each frame individually, it was easily parallelizable, bringing the runtime down to 3 minutes using a setup akin to simple batch processing. The second step was downsizing the mask being fed into the library function bringing the runtime to 50 seconds. The downsized mask contained a quarter of the original pixel count and so roughly quartered the runtime. This second optimization is lossy unfortunately.

Neither of these techniques were real-time or near real-time but they were optimized to a point where the user testing was structured to have the longer runtimes running in the background while participants did other tasks.

Qualitative

This evaluation was a series of user tests and interviews to gauge opinion on the project, VR video editing in general and 360 video. Ethical approval was sought out and given for the user testing and interviews. The sample size was too small for statistical significance, but the semi-structured interviews collected a varied range of well thought out opinions on these topics. Participants found the controls and interactions logical for the most part. To handle the selection inaccuracies participants recommended adding movement smoothing, reducing the controller's sensitivity (this would have to be toggleable) and better hardware. Three out of the five participants said they thought the VR video editing setup was worth it, if they didn't have to pay for a new VR setup. The participants had a wide range of experience with video editing and most of them liked the techniques demonstrated. Opinions were given to use 360 video and video editing for entertainment, education and VR 'tourism', seeing places and tourist locations from around the world without needing to go there. These are all use cases previously mentioned in this report. A specific use case mentioned by a participant for the pixel-wise video edits was to use them to annotate videos and have the annotations move with the annotated content in the video. Some issues mentioned with the system were the video edits taking too long to run, the inaccuracy of pixel selection, the lack of in-app feedback and the need for a more user-friendly menu.

VII. FUTURE WORK

There are definite limitations with the final artefact. Unity Engine was the best choice for the interface but solely because of the time constraints of this project. Unity's inability to rewind videos and its issues with loading edited files during runtime mean a truly user-friendly system is impossible with Unity. Building this project in Unreal Engine would be a more complicated process, but it would allow for videos to be rewound and would give the option for C++ video editing techniques to be included directly in the project (although pre-built techniques would still need to be handled). An Unreal project would still have significant overhead however. Given enough time, building this project entirely in C++ potentially using OpenGL for rendering and another library for VR would also remove Unity Engines limitations. Building the project entirely in C++ would mean a much smaller, more optimized project that could natively run video editing techniques written in C++. Once again it would still need a way to run techniques written in other languages to keep its adaptability.

Video editing is an intensive process, The runtimes of the two video editing techniques were optimized to a degree, but neither of them are close to real time. Further optimization is certainly possible, a good direction to go in would be to utilize the GPU. Operations on images work very well on the GPU. Both optical flow and GrabCut can work on the GPU[17][18] and doing so would vastly improve the runtime. Computing on the GPU was not used for this project due to inexperience and the time constraints. ML models for both feature tracking and foreground-background separation also exist or could be developed. Once trained these models could potentially shorten the runtime.

The two techniques developed also need some work. The color editing does not produce perfect results as seen previously in Fig. 6. The background/foreground mask produced by GrabCut is not perfect and so the color is applied to some areas it shouldn't be. The fix is to use the second step of GrabCut mentioned in the implementation. Also, the rectangle region used in the first step of GrabCut is hardcoded. Giving users the ability to draw a rectangle in the interface and passing that to the technique would make it much more versatile.

The graffiti tracking doesn't handle two problems with 360 videos. First of all, it does not wrap around, work was done to allow for this, but it crashes most of the time when presented with an edge case. Secondly the graffiti mask passed to the executable distorts when shifted significantly throughout the video. Currently the graffiti is simply translated in the 2d space when being shifted. This leads to it being stretched when placed on the severely warped areas of an equirectangular frame and viewed in the interface. The graffiti mask needs to be similarly warped to the frame each time to prevent this.

V. CONCLUSION

So, to conclude this project has created an interface that allows users to select pixels of equirectangular images while the video is in its proper viewing form. They can also input other pieces of information such as a color value. They are able to do all this completely immersed in VR. They are also able to use video editing techniques and view the edits they make. Two pixel-wise video editing techniques were created and used along with the system. The system has been created in a relatively open fashion and has been documented for future development and utilisation of the system by researchers of new or improved 360 video editing techniques. Finally, user testing results encouraged further research in this area and left helpful feedback on where to begin improving the system.

VIII. Acknowledgments

Thankyou to Fang Lue Zhang. His knowledge of image processing was invaluable.

References

[1] J. Katatikarn, "Virtual Reality statistics: The ultimate list in 2023," Academy of Animated Art, 02-Nov-2022. [Online]. Available: https://academyofanimatedart.com/virtual-reality-statistics/. [Accessed: 15-Oct-2023].
[2] "AR/VR headset shipments by market 2021-2026," Statista. [Online]. Available: https://www.statista.com/statistics/1301629/ar-vr-headset-shipments-by-market. [Accessed: 15-Oct-2023].
[3] "AR/VR headset companies shipment share worldwide 2022-2023, by quarter," Statista. [Online]. Available: https://www.statista.com/statistics/1407105/ar-vr-headset-companies-shipment-share. [Accessed: 15-Oct-2023].
[4] "Facebook," Facebook.com. [Online]. Available: https://www.facebook.com/help/828417127257368. [Accessed: 15-Oct-2023].
[5] Meta.com. [Online]. Available: https://www.meta.com/nz/quest/products/quest-2/tech-specs/#tech-specs. [Accessed: 15-Oct-2023].
[6] C. Nguyen, S. DiVerdi, A. Hertzmann, and F. Liu, "Vremiere: In-headset virtual reality video editing," in Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, 2017.
[7] C. P. Evans, E. Chiarovano, and H. G. MacDougall, "The potential benefits of personalized 360 video experiences on affect: A proof-of-concept study," Cyberpsychol. Behav. Soc. Netw., vol. 23, no. 2, pp. 134–138, 2020.
[8] S. Yoganathan, D. A. Finch, E. Parkin, and J. Pollard, "360° virtual reality video for the acquisition of knot tying skills: A randomised controlled trial," Int. J. Surg., vol. 54, pp. 24–27, 2018.
[9] V. Arents, P. C. M. de Groot, V. M. D. Struben, and K. J. van Stralen, "Use of 360° virtual reality video in medical obstetrical education: a quasi-experimental design," BMC Med. Educ., vol. 21, no. 1, 2021.
[10] H.-R. Rantamaa, J. Kangas, S. K. Kumar, H. Mehtonen, J. Järnstedt, and R. Raisamo, "Comparison of a VR stylus with a controller, hand tracking, and a mouse for object manipulation and medical marking tasks in virtual reality," Appl. Sci. (Basel), vol. 13, no. 4, p. 2251, 2023.
[11] Adobe Creative Cloud, USA. *#CloverVR. Adobe MAX 2016 (Sneak Peeks) | Adobe Creative Cloud*. (Nov. 4, 2016). Accessed: Oct. 15, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=tFkJXwH1VTE
[12] Unity Technologies, "VideoPlayer.playbackSpeed," Unity3d.com. [Online]. Available: https://docs.unity3d.com/ScriptReference/Video.VideoPlayer-playbackSpeed.html. [Accessed: 15-Oct-2023].
[13] A. Chauvet, Y. Sugaya, T. Miyazaki, and S. Omachi, "Optical Flow-Based Fast Motion Parameters Estimation for Affine Motion Compensation," Applied Sciences, vol. 10, no. 2, p. 729, Jan. 2020, doi: 10.3390/app10020729.
[14] Li, Y., Barnes, C., Huang, K. and Zhang, F.L., 2022, October. Deep 360∘ Optical Flow Estimation Based on Multi-Projection Fusion. In European Conference on Computer Vision (pp. 336-352). Cham: Springer Nature Switzerland.
[15] Meta.com. [Online]. Available: https://www.meta.com/nz/quest/quest-3. [Accessed: 15-Oct-2023].
[16] C. Creed, M. Al-Kalbani, A. Theil, S. Sarcar, and I. Williams, "Inclusive AR/VR: accessibility barriers for immersive technologies," Univers. Access Inf. Soc., 2023.
[17] K. Pauwels and M. M. Van Hulle Laboratorium voor Neuro- en Psychofysiologie, "Realtime phase-based optical flow on the GPU," Kuleuven.be. [Online]. Available: https://gbiomed.kuleuven.be/english/research/50000666/50000669/50488669/neuro_research/neuro_research_mvanhulle/comp_pdf/GPU.pdf. [Accessed: 15-Oct-2023].
[18] S. Jose and T. Stich, "Graph Cuts with CUDA," Nvidia.com. [Online]. Available: https://www.nvidia.com/content/gtc/documents/1060_gtc09.pdf. [Accessed: 15-Oct-2023].

Fig. 1. "Convert Equirectangular Projection to Cube Faces," Jamesfmackenzie.com. [Online]. Available: https://www.jamesfmackenzie.com/2016/10/18/convert-equirectangular-projection-to-cube-faces. [Accessed: 15-Oct-2023].
Fig. 2. Image captured from: Adobe Creative Cloud, USA. *#CloverVR. Adobe MAX 2016 (Sneak Peeks) | Adobe Creative Cloud*. (Nov. 4, 2016). Accessed: Oct. 15, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=tFkJXwH1VTE
Fig. 3. Researchgate.net. [Online]. Available: https://www.researchgate.net/figure/Visual-representation-of-equirectangular-and-cubemap-projections-for-the-captured-360_fig1_335258068. [Accessed: 15-Oct-2023].
Fig. 4. "OpenCV: Optical Flow," Opencv.org. [Online]. Available: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. [Accessed: 15-Oct-2023].
Fig. 5. Source frame from: "360 video database," VHIL. [Online]. Available: https://stanfordvr.com/360data. [Accessed: 15-Oct-2023].
Fig. 6. Source frame from "360 video database," VHIL. [Online]. Available: https://stanfordvr.com/360data. [Accessed: 15-Oct-2023].