Virtual Reality Window

Pol Rosello, Cheng-Han Wu, and Jiayu Wu
Stanford University
{prosello, chw0208, jiayuwu}@stanford.edu

Abstract

We describe a virtual reality system which achieves the illusion of depth on an ordinary display, requiring no special equipment other than a webcam and a computer. The display simulates motion parallax and a changing field of view for an individual user, and in this way functions as a "window" into a virtual, three-dimensional scene. We use Haar cascade classifiers, camera models, and Kalman filtering to track the user's head position in 3D in real time and update the display according to an off-axis projection model. We extend our system with a gesture recognition pipeline that allows for object or scene orbiting using hand gestures. We compare various approaches to head and hand tracking both qualitatively and quantitatively. Our model processes frames faster than the camera's native frame rate and provides a convincing illusion of depth, as exemplified in our demonstration video. We implemented our project in C++ using OpenCV for computer vision tasks and OpenGL for rendering tasks.

1. Introduction

Virtual reality (VR) and augmented reality (AR) technology has recently become more accessible to the general public. Despite this, consumer models of VR/AR headsets almost exclusively target the gaming market, and other applications where VR/AR might be useful have been mostly ignored. A major roadblock to a more widespread adoption of VR/AR technology is that it requires very expensive and uncomfortable equipment which must be worn at all times. This is discouraging to potential users who may not be convinced that VR/AR could be useful for them or who cannot afford a headset.

One application of VR/AR technology is to be able to view three-dimensional models from multiple angles simply by shifting one's head. Being able to quickly and easily understand a 3D model is especially important in fields such as architecture, industrial design, mechanical engineering, and 3D art. We propose a method for viewing 3D models that only requires a single webcam, a computer, and a monitor (most modern laptops bundle all three) and allows the user to view the model from multiple perspectives by shifting his or her viewing angle without the use of any special headsets. We use the webcam to track the position of the user's head in 3D space, updating the monitor in real time to reflect what the user would see from that viewing angle. In this way, the monitor functions as an augmented reality "window" into a 3D scene, simulating motion parallax and a changing field of view as the user moves, thus providing the illusion of depth. This is similar to Johnny Chung Lee's Wii Remote Desktop VR display [1], but we remove the need for any type of headwear. While the illusion only works for one user at a time, this is also the case with current augmented reality headsets. We propose to further extend this interface with gesture recognition to interact with the scene by waving one's hand in front of the camera.

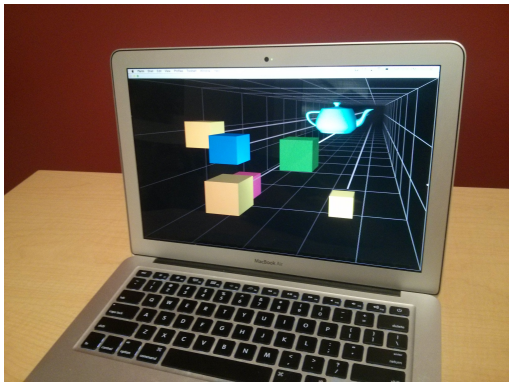


Figure 1: Our virtual reality window display.

2. Previous Work

The type of virtual reality display we describe was popularized by Lee's Human-Computer Interaction work at Carnegie Mellon. Lee's approach makes use of a Nintendo Wii Remote, which contains an infrared camera, to perform head tracking. In Lee's setup, the remote is placed near a display (such as a television) facing the user, and the user is made to wear clear eyeglasses fitted with one infrared LED

on each side. Because the Wii Remote's hardware is especially designed for tracking two IR points, the remote can be leveraged to achieve fast and accurate head tracking which can then be used to update a 3D scene on the display for a successful virtual reality effect. However, Lee's use of the Wii Remote limits its availability to a wider audience due to reliance on specialized hardware. Our proposed solution does not require any hardware other than a computer and an attached camera such as a webcam.

A second approach to virtual reality with no head gear was pioneered by Sandin et al. in [9]. In their work, the authors use a large-scale tiled array of 35 displays to provide a wide field-of-view for the user. Real-time head tracking is performed using artificial neural networks, requiring sixteen Linux PCs, each with two NVIDIA GeForce 7900 GTXs. The displays are fitted with a parallax barrier providing autostereoscopic viewing for the user. Although this system does away with the need for a headset and provides a better sense of depth than our approach, it again necessitates expensive special equipment. Furthermore, current advances in virtual reality technology mean that a high-end headset with better fidelity can be purchased at a fraction of the cost of their setup, so its intended target is unclear.

Other approaches to this kind of virtual reality display have avoided the use of special equipment, but rely on color-based algorithms to perform head tracking. One such approach is from Goorts et al. in [10], who use the camshift algorithm to detect a head within the webcam frame and estimate its size. The downside to using color-based tracking is that it relies on fairly consistent illumination, and most importantly, it is susceptible to high rates of false positives with skin-colored objects in the scene.

3. Methods

3.1. Technical Summary

At a high level, the project has three main components: head tracking, gesture recognition, and scene rendering. Once the user's eye position in 3D space is obtained, along with any gestures he or she may be performing, the scene is updated to reflect what the user should see from their current position. Due to the possibility of the user not directly facing the screen, an off-axis skewed frustum projection must be used. We implemented our system in C++ and used OpenCV for computer vision tasks and OpenGL for scene rendering tasks.

3.2. Head Tracking

3.2.1 Camera Calibration

The first step in our pipeline is to calibrate the webcam to obtain an accurate focal length and camera center for head tracking. We capture 15 images of a checkerboard pattern with a known geometry from multiple viewpoints, extract the corners of the checkerboard, and run the global Levenberg-Marquardt optimization algorithm to obtain the values of the intrinsic camera parameters which minimize the total sum of squared distances between the observed feature points and the projected object points (i.e. the reprojection error). We then obtain the camera matrix K, which contains the focal lengths f_x and f_y and the camera center c_x and c_y as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

In practice, we set f equal to the mean of f_x and f_y for simplicity. For performance reasons, we reduce the resolution of the camera frames (see Section 4.3). We note that whenever the resolution of the camera frame is changed, these values should be scaled appropriately.

3.2.2 Haar Cascade Classifier

The first step of our head tracking pipeline is face detection using a pre-trained Haar feature-based cascaded classifier as in [2]. The basic working principle of this classifier is to learn a collection of Haar-like features which yield a high detection rate on faces. Each Haar-like feature considers adjacent rectangular regions in a window, sums up the pixel intensities in each region, and calculates the difference between these sums. A weak classifier is trained for each such feature by determining a threshold value for the difference which best separates faces from non-faces on the training set. By cascading several hundred weak classifiers in series, the accuracy of the system increases. We choose Haar cascade classifiers for this task because they are simple, and most importantly, fast enough to achieve real-time classification on a modern laptop. They are also more robust to illumination changes or cluttered backgrounds than color-based methods such as camshift. We convert each frame to gray scale before using the Haar cascade to detect the location and size of the user's head.

3.2.3 Obtaining 3D Coordinates

After the face is detected, we estimate the three-dimensional coordinates of the head (X, Y, Z) to determine how to update the scene. First, the point (x, y) between the user's two eyes is obtained based on the size and position of the detected face in the current frame. This is done simply through estimation based on a fixed ratio for the position of the user's eyes relative to the size of their head. We then estimate the distance Z of the head away from the camera based on the detected width of the user's head, using the pinhole camera model and the triangle inequality

$$Z = \frac{W_{known} f}{w_{meas}} \tag{1}$$

where W_{known} is the known width of the user's head, w_{meas} is the measured width of the head on the screen, and f is the focal length found in the camera calibration step. Then we compute the X and Y coordinates of the head in 3D as

$$X = \frac{(x - c_x)Z}{f} \qquad Y = \frac{(y - c_y)Z}{f} \qquad (2)$$

where (c_x, c_y) is the center of the camera.

3.2.4 Kalman Filtering

A major challenge we had to overcome is to provide temporally-smooth head tracking. Alone, the Haar classifier operates on frames independently, with no notion of continuity between frames. Small discontinuities in tracking the user's position and size between frames lead to a persistently jittery virtual scene and greatly detract from the augmented reality illusion.

A simple approach to reduce noise in the tracking algorithm is to compute a moving average of the head position across several recent frames. However, a moving average filter is slow to react to changes in the head position because it assumes the face is an object with a constant position. Ignoring noise, if we are averaging the head position across k frames and the head moves, it will take k frames for the filtered head position to converge to the correct location. In our experiments, this delay was perceptible by the user and detracted from the illusion of depth. A moving average filter also does not deal well with occlusions or false negatives in the head detection step: if the face is not found across several frames, the prediction of the current head position will converge to a static point even if the head was previously moving.

To address these issues, we use a Kalman filter based on a physical model of the head's movements to smooth out jittery tracking behavior. The Kalman filter is also able to predict the head's position even if the face was not found across several frames by relying on the model of its movement. We consider different physical models for the user's head and compare these in Section 4. The four models for the face we considered were:

1. An object in two dimensions with constant velocity and a constant width and height
2. An object in three dimensions with a constant velocity
3. An object in three dimensions with a constant acceleration
4. An object in three dimensions with a constant acceleration in the x and y dimensions with constant velocity in the z dimension

Note that the first model filters the detected head position *before* mapping it to 3D coordinates, while the remaining models filter the head position after this step. The inclusion of the last model is due to our observation that the greatest source of inconsistency in our Haar cascade classifier is the detected size of the head (affecting the z position of the face). Modeling the head as having a constant velocity in a particular dimension leads to smoother tracking in that dimension, although it reduces its ability to respond quickly to changes in the head's velocity (i.e. acceleration) along that axis.

3.3. Gesture Recognition

The gesture recognition system is composed of two layers: tracking and recognition.

3.3.1 Detection

Our virtual reality window incorporates a function for users to rotate the scene object using gesture recognition. In order to create an intuitive user experience, we use hand movement parallel to the screen plane for rotation. Hand movement is tracked using a fist detector. Similar to head tracking, we implement a Haar feature-based cascade classifier via OpenCV. We focus specifically on fist detection since we want the users to imagine grabbing the virtual object and perform manipulation. A pre-trained Haar cascade model of fists is introduced as training data for the OpenCV multi-scale detection function.

3.3.2 Recognition

The recognition layer handles data extracted in the previous layers and assigns the resulting groups its gesture classification, producing a command for scene rendering. Our method focuses on hand-driven control with information extracted through the tracking of fingertips or the centroid of the hand. Methods such as Principal Component Analysis (PCA) with priors of several gestures placed as training data have been used for recognition. Hidden Markov Models (HMM) have also been used as a way of categorizing different hand trajectories in order to prevent unintentional gesture movements [6].

For our experimental setting, we want to create an immersive user experience where a fist gesture is similar to a real-world grabbing motion. The system is designed such that only the grabbing motion will allow users to control specific objects in the scene. After several experiments with different fist-detection methods, we decided to use cluster smoothing with non-maximal suppression for smoothing the gesture locus. A set of five raw 2D coordinates is passed into a queue, used to calculate a new centroid and update the scene (see Figure 2). Due to the nature of the shape and contrast of fist compared to that of a face, our training data

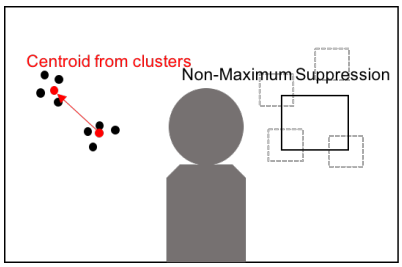


Figure 2: Gesture detection smoothing procedure.

is prone to rectangular objects with either vertical or horizontal contrasts. However, false positives are relatively easy to eliminate using non-maximal suppression since the user is normally the closet object to the camera. By choosing the largest detected element, we were able to successfully select fists. From our experiments when incorporating only half the full resolution (1280 x 1024) of a MacBook Air camera, system performance reaches 80 frames per second.

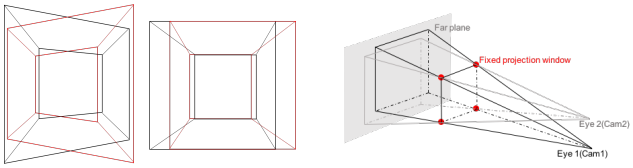
3.4. Scene Rendering

The major task of scene rendering in our project is to create a 3D see-through experience. In order to do that, we render 3D images in real-time based on the user's head position. We model the user's eyes as a virtual camera and treat their coordinates in 3D as the camera's extrinsic parameters. Once our head tracking pipeline provides us with an estimate for these parameters, we then update the image on the display screen, as seen in Figure 4.

3.4.1 Skewed Frustum Projection

We can use the OpenGL `glLookAt` function to perform a traditional projective transformation and modify what part of the virtual world is displayed according to the user's head position. However, if the user is not directly in front of the screen, this creates a tilt to the projection plane, leading to Keystoning effect where the user's perception of the scene is distorted (Figure 3a). Because the physical screen cannot move along with this changing view, a modification is needed by using `glFrustum`. `glFrustum` generalizes `glLookAt` by allowing the scene boundaries to be asymmetric about the z-axis of the screen. In other words, the view frustum from the eye to the screen may be unaligned with the viewing plane normal. Each time the head tracker updates the head position, the view frustum will be adjusted (Figure 3b). The goal of calling `glFrustum` is to position the viewing plane relative to the OpenGL camera in the same way that the physical window is positioned relative to our tracked head position.

The two main issues we need to solve for rendering is off-axis projection and XY-plane rotation. Because the user can move around, the vector from eye position to screen



(a) (b)

Figure 3: Skewed frustum projection. (a) Left: Keystoning effect of incorrect projection method. Right: Skewed frustum projection. (b) Skewed frustum in different eye positions with the virtual camera located at the user's location.

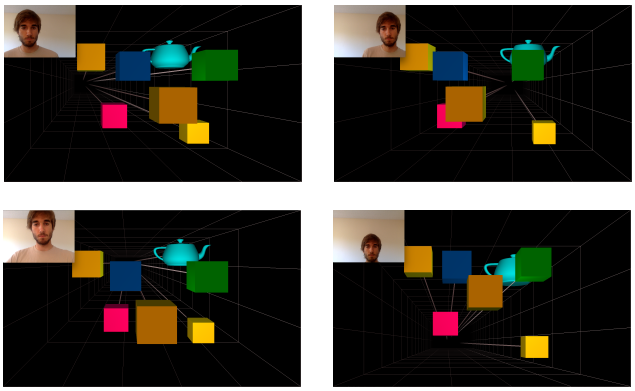
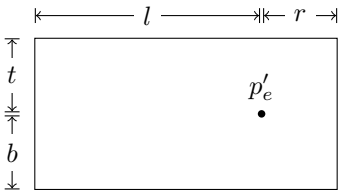


Figure 4: Scene rendering from different viewpoints.

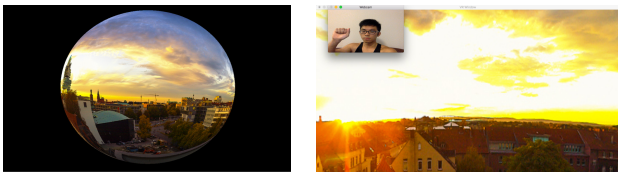
origin is not necessarily perpendicular to the screen. Furthermore, the user's eye position may not be parallel to XY-plane, which means we need to rotate the XY-plane to make it parallel to the user's eye position. Therefore, we need to find three transformation matrices: M, an off-axis projective transformation; R, a rotation transformation to adjust the XY-plane; and finally T, a translation transformation to align the frustum's apex with the user's eye position. We base this approach on [8] and describe it below.

Let p_a, p_b, p_c be the lower left, lower right and upper left corner of the screen, p_e be the center of eye position. Let v_r be the unit vector of $p_b - p_a$, v_u be the unit vector of $p_c - p_a$, v_n be the unit vector of $v_r \times v_u$.



We also have the parameters defined in the above figure. p'_e is the intersection of the line that follows v_n and passes through p_e and the screen plane.

We define the off-axis perspective projection M, the XY-plane rotation R, and the translation T as:



(a) (b)

Figure 5: Panoramic rendering. (a) Interpolating 360-degree panorama into spherical texture (seen from the outside). (b) Effect of the window. We perform scene rotation using the fist while rendering the perspective projection with respect to the user.

$$M = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3}$$

$$R = \begin{bmatrix} v_{rx} & v_{ry} & v_{rz} & 0 \\ v_{ux} & v_{uy} & v_{uz} & 0 \\ v_{nx} & v_{ny} & v_{nz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The final perspective projection transformation we use is MRT.

3.4.2 Panoramic Scene

We also added a second scene of a panoramic picture, reminiscent of looking out of an actual window (see Figure 5). In this case, users can use their fist to rotate the full scene while maintaining the functionality of the head-tracking perspective projection. Unlike the previous rendering environment, gesture rotation in this case will affect the whole scene instead of a specific object. If we follow the previous procedure and use a panoramic picture as an flat "wallpaper" at the far end of the scene, rotation may throw the picture out of bounds. On the other hand, if we reduce the range of rotation, part of the picture will remain unseen. Due to the issue presented above, we use a sphere as the scene, textured on the inside with a panoramic picture, and place the virtual camera inside the sphere. Under the assumption that the sphere radius is large enough that the scene can be seen as a flat picture, radial distortion does not heavily affect users' immersive experience.

4. Experiments

4.1. Head Tracking Evaluation

4.1.1 Qualitative Evaluation

We qualitatively evaluate the performance of our head tracking pipeline by using our system and noting any effects which detract from the illusion of depth. Figure 6 shows the appearance of the screen from different angles. As evidenced by our demonstration video, the most helpful addition to our system in achieving the illusion of depth was the use of filtering. Without Kalman filtering, scene rendering is jittery even when the user is standing still. This is due to inter-frame variation caused by noise leading the Haar cascade face detector to return slightly different head sizes and positions between frames. Without any filtering, the system is also unable to predict the head location whenever the Haar cascade misses a face detection, causing the scene to momentarily freeze until detection is successful again. Both of these behaviors greatly detract from the illusion of depth. With the addition of Kalman filtering, the jitter between frames is greatly reduced and the illusion of depth is more successful.

We experimented with different physical models of the face when performing Kalman filtering and found that modeling the head as an object in three dimensions with a constant acceleration was the most convincing approach from the user's perspective. Although this was not the model with the lowest RMSE in our quantitative evaluation (see Section 4.1.2), we suspect RMSE may not correlate well with user experience. The models that treated the face as an object with a constant velocity were noticeably slow in updating the display whenever the user would accelerate by moving to a different viewpoint. In our evaluation videos, the face was moving relatively slowly and with a constant velocity, which would explain why Mode 2 fared better quantitatively.

Perhaps the most noticeable issue with our system is that our display is not stereoscopic. Although motion parallax and a changing field of view are a significant part of human depth perception, stereo vision plays an important role as well. When viewed with one eye closed, the scene appears more realistic than with both eyes open. This is most apparent when comparing the results in monocular first-person videos of our system (taken with a single camera) with our real-world experience with it. While the scene updates rapidly and convincingly, there is a reduction in the illusion caused by the same image being projected to both eyes. We contemplated rendering a stereoscopic scene and requiring the user to wear anaglyph red-cyan 3D glasses. However, we did not consider this to be a satisfying solution due to the addition of external hardware, as discussed in Sections 1 and 2.

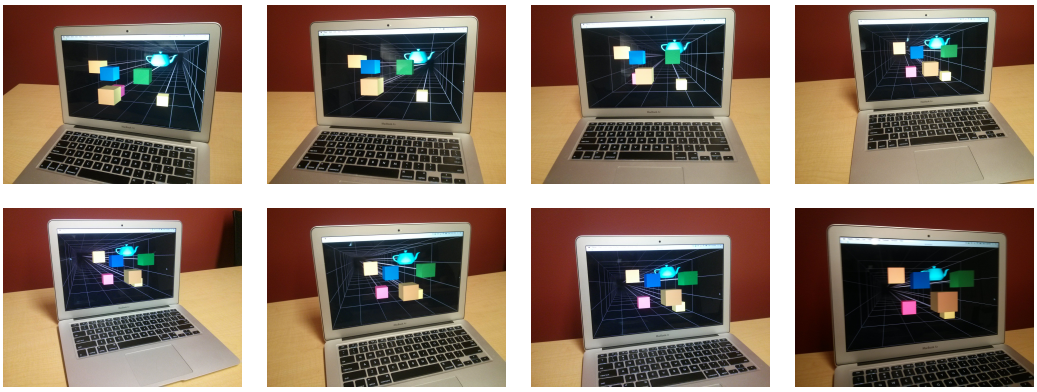


Figure 6: Display appearance from different viewpoints. Reflections diminish the illusion.

Other issues we noticed were caused by a dirty display or an inappropriate screen brightness. With smudges or reflections on the screen, it is easier to notice that the objects in our virtual reality scene are actually lying flat on the plane of the display. These issues are easily fixed by cleaning the screen and adjusting its brightness before use. Despite these problems, the illusion of depth is strong enough for our system to be successful.

4.1.2 Quantitative Evaluation

We evaluate our model's head tracking performance quantitatively by manually annotating the head and eye positions in sample videos of a user using the virtual window, and calculating the average distance between the model's detected head position and the true position across all frames. If our evaluation videos have N frames, where the position of the head in frame i is (x_i, y_i) and our model's detected position is (x'_i, y'_i), our error metric for head tracking will be the root-mean-square error:

$$RMSE = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \qquad (6)$$

In our figures, we normalize RMSE to the width of the frame in order to compare tracking error using different frame resolutions.

We evaluate our system by comparing the RMSE of the detected head location using different smoothing techniques and lighting conditions. First, we input a full-resolution pre-annotated video to the system and use RMSE to evaluate 5 different tracking modes. Mode 0 is the baseline Haar cascade tracking. Modes 1-4 correspond to the different physical models described in Section 3.2.4. Mode 1 is the Haar cascade tracking with a two-dimensional Kalman filter and Modes 2-4 is the Haar cascade tracking with three-dimensional Kalman filter with different physical models.

The results of this evaluation are shown in Figure 7. The results show that the best performing mode is Mode 2, which is to model face as a 3D object with constant velocity. In our annotated video, the head moves in a constant velocity most of the time, and it also moves slightly along the z-axis. Therefore, the modeling method of Mode 2 fits the features of the video. We notice that Mode 0 (tracking without smoothing) only has a slightly worse performance than Mode 2. However, the actual user experience varies a

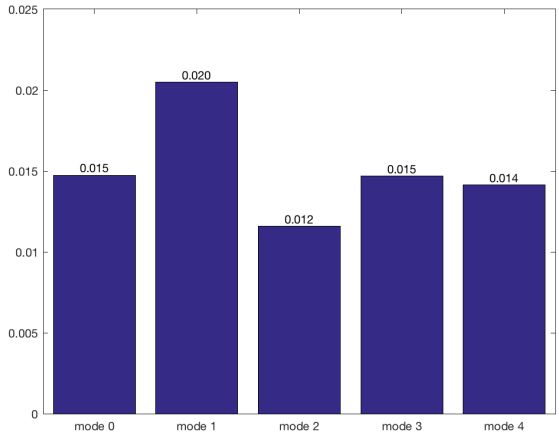


Figure 7: Head tracking RMSE of various tracking modes. RMSE is normalized to the width of the full 1280 x 1024 resolution. Mode 0 is the baseline Haar cascade tracking with no filtering. Modes 1-4 perform Kalman filtering after detection using various physical models of the face. Mode 1 models the face as an object in 2D with constant velocity and a constant width and height. Modes 2-4 perform filtering after 3D position estimation. Mode 2 models the face as an object in 3D with constant velocity. Mode 3 models the face as an object in 3D with constant acceleration. Mode 4 models the face as an object with constant acceleration in the x and y dimensions and constant velocity in the z dimension.

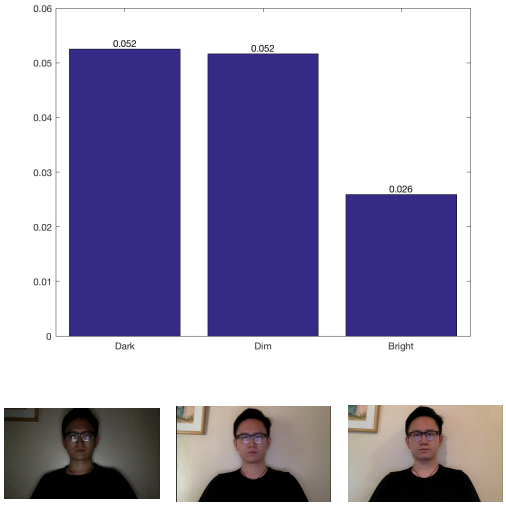


Figure 8: Head tracking RMSE under different lighting conditions at full 1280 x 1024 resolution. RMSE is normalized to the frame width.

lot for Mode 0 and Mode 2 (see Section 4.1.1). The scene rendering for our system using Mode 2 is smooth while that for Mode 0 is shaky.

The results for different tracking modes also show that choosing the best modeling method depends on the characteristics of the input frames. Specifically, in most use cases where the user sits in front of the camera and makes head movements that are not too fast, then the best tracking to use is to model the face as a 3D object with constant velocity.

We also evaluate the lighting conditions in Figure 8. We compare three different lighting conditions: dark, dim and bright. We use the same system with tracking mode 2 and full resolution to test three videos with varying lighting conditions. The results show that the error of dark and dim input videos is significantly higher than the bright one. Therefore, lighting conditions can impact the performance of Haar cascade tracking, and the input video should be reasonably bright. Furthermore, we notice that the error of dark input and that of dim input are almost the same. We believe there is a lighting threshold below which the performance of the Haar cascade classifier will be somewhat equivalently poor.

4.2. Gesture Recognition Evaluation

4.2.1 Qualitative Evaluation

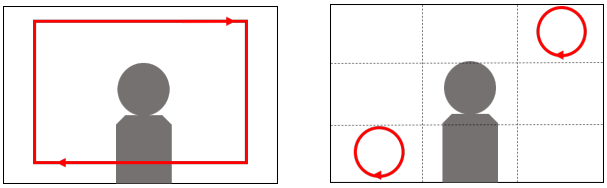
To evaluate our gesture recognition system, we observed the effect of the user's environment on its performance. Although there is less inter-class variation with fists than with faces, a fist shape has significantly greater similarity to daily objects with the same contour and contrasts, so detection may not be as accurate. Features of a fist rely on contrast between folded fingers and the palm, which will be less significant under background lighting or saturated illumination. From our experiments we found that the best fist tracking was achieved with side illumination, which gives good contrast; and a plain background, which gives a clear contour. However, detection errors are frequent when the user puts the fist in front of the body, especially the facial area. This is due to the similarity in the value of the skin tone when transformed to grayscale. The system will not have a clear contrast over the contour of the fist from its background; thus, a Haar cascade classification system will not be able to match certain features in the early stages of classification and will fail to identify the fist.

4.2.2 Quantitative Evaluation

Similar to our head tracking evaluation, we evaluate our gesture recognition pipeline by manually annotating videos and computing the RMSE of our detected hand position. We consider two independent parameters which affect the detection accuracy, both related to the webcam detection environment: background complexity and illumination. Computer webcams tend to adjust exposure according to average frame brightness, which causes faces and fists to be plain black, so backward lighting will not be taken into consideration in our evaluation for its near zero accuracy. Note that although exact numbers are provided for comparison, the testing setup still has variation that is not explicitly accounted for, including fist rotation and shadows. These minor variations do not have a great impact in our comparison but will affect our confidence interval.

We first evaluate the RMSE of our fist detection pipeline as the fist is moved in the frame. We consider four camera frame conditions: a plain background color with sufficient illumination, a plain background color with a complex background pattern, a plain background with insufficient illumination, and a complex background with insufficient illumination. Figure 10 shows the RMSE of fist position measured between the detected fist positions and the manually-annotated positions in four evaluation videos, one for each condition. In each video, the fist was moved around the user's face according to the pattern in Figure 9a. It can be seen from the comparison that a plain background provides good contrast for Haar cascade detection. The importance of background clarity is shown when comparing a complex background with sufficient lighting and a plain background with insufficient lighting. The plain background case has a relatively lower RMSE under the same movement.

We also evaluate the error of our fist detector depending on the position of the fist in the frame. We take the average occurrence within 100 frames of detection failure and sudden out-of-locus jump of over 30 pixels into account. The



(a) (b)

Figure 9: Movement patterns in manually-annotated fist detection evaluation videos. (a) Fist locus for RMSE evaluation under different conditions. (b) Fist locus for spatial error evaluation.

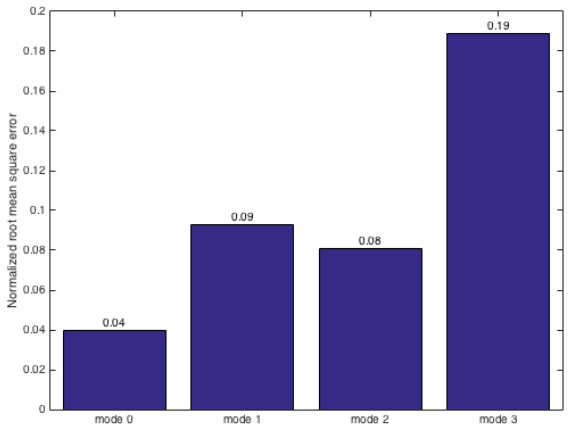


Figure 10: Root mean square error of fist detection in different background conditions according to the movement in Figure 9a, normalized by the frame width. Mode 0: Plain background and sufficient illumination. Mode 1: Complex background with sufficient illumination. Mode 3: Plain background with insufficient illumination. Mode 4: Complex background with insufficient illumination.

frame is divided into nine regions as shown in Figure 9b. For each condition, we computed the failure rate of the system in each of the nine regions. A failure is defined as either the fist being detected in the incorrect region of the frame, the fist experiencing a sudden jump of over 30 pixels with respect to the previous frame, or no fist being detected at all. We computed the failure rate over the course of 100 frames per region. The results of this analysis are shown in Figure 11. The user's body, especially the facial area, is prone to a high failure rate in all four cases. Due to the color similarity of the user's skin tone, the grayscale values fed into the Haar-cascade system will not be as distinguishable as a plain background. The importance of clear contour and high contrast is shown to be critical from the above experiments.

(a) (b) (c) (d)

Figure 11: Heat map of failure rate with respect to local fist movement (Figure 9b) within 100 frames. (a) Plain background with sufficient illumination. (b) Complex background with sufficient illumination. (c) Plain background with insufficient illumination. (d) Complex background with insufficient illumination.

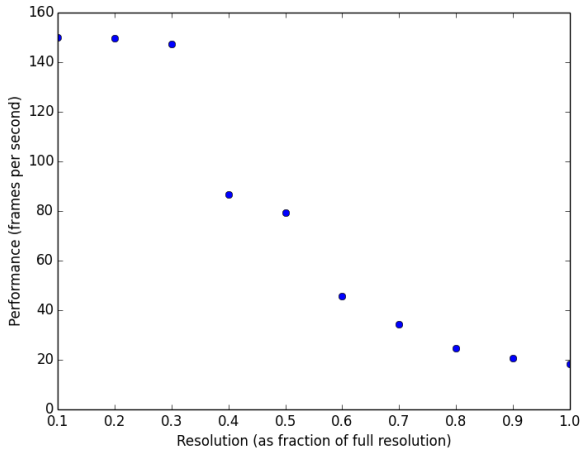


Figure 12: Latency of system on different frame sizes, evaluated on a 2015 MacBook Air with a camera resolution of 1280 x 1024 and a screen resolution of 1440 x 900. The x-axis refers to the resolution of the processed frames. A resolution factor of 0.5 reduces the number of pixels on the screen by a factor of 4.

4.3. Latency Evaluation

We also evaluate the latency of our system. At the full 1280 x 1024 resolution of our webcam, the processing time per frame of our pipeline is longer than the camera's native frame rate, leading to dropped frames and a noticeably slow response time between the user's head movements and the updated scene on the display. Because our system is scale invariant, we can address this issue by reducing the resolution of each frame before processing. The effects on latency

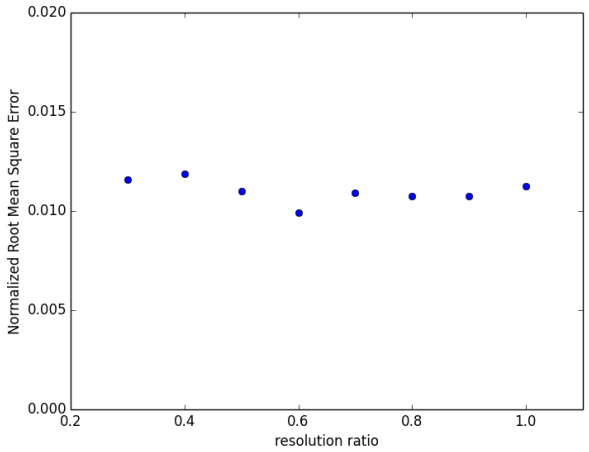


Figure 13: RMSE of head tracking on different frame sizes, normalized by the frame width. The resolution ratio is defined in terms of the fraction of the width of the full-resolution frame (1280 x 1024). A resolution ratio of 0.5 reduces the number of pixels on the screen by a factor of 4.

of downscaling frames are shown in Figure 12. We are able to process frames above the camera's frame rate of 30 fps by reducing the width of the webcam frame by a factor of 0.7, leading to a resolution of 896 x 717. At this frame rate, the latency of the system is fast enough for the user to not notice any delay between sequential updates of the display. The RMSE of the system is also not significantly affected, as seen in Figure 13.

5. Conclusion and future work

We designed and implemented a virtual window display that simulates the illusion of depth by tracking the user's head using a webcam and updating the rendering of a virtual scene accordingly. We quantitatively evaluate our head and fist tracking pipeline by manually annotating videos and computing the RMSE of the system under different scene conditions, confirming that tracking works best when the user's surroundings are well-lit and uncluttered. We compare different physical models for our head tracking Kalman filter and conclude that modeling the head as an object in 3D with a constant velocity provides a lower RMSE, while modeling it as an object in 3D with a constant acceleration provides higher responsiveness to sudden changes in velocity. The latency of our system can be adjusted by downscaling the resolution of the camera frames, enabling it to process frames at the camera's native frame rate without a noticeable drop in accuracy. The illusion of depth is convincing, especially when using a clean, non-reflective display and viewing it with one eye shut. Overall, our virtual reality window meets our initial objective of being useful for viewing 3D models.

Our work can be extended in several ways. One of these is to render the scene as a stereoscopic display, using red-cyan anaglyph glasses to provide a separate signal to each eye, as alluded to in Section 4.1.1. An autostereographic display could also be used instead. Another avenue of research would be to incorporate an explicit depth signal for improved 3D tracking accuracy, such as that provided by a Microsoft Kinect. However, all of these extensions require additional hardware: we feel that our system is close to the limits of what can be achieved with a single display and a camera. Integrating our approach with a modern game engine or a 3D CAD program would be helpful in evaluating its usefulness in real-world applications.

References

[1] Lee, Johnny Chung. *Hacking the Nintendo Wii Remote*. IEEE Pervasive Computing, 2008.

[2] Viola, Paul and Jones, Michael. *Rapid Object Detection using a Boosted Cascade of Simple Features*. CVPR 2001.

[3] M. Cote, P. Payeur, and G. Comeau. *Comparative study of adaptive segmentation techniques for gesture analysis in unconstrained environments.* IEEE Int. Workshop on Imagining Systems and Techniques, pages 2833, 2006.

[4] X. Zabulis, H. Baltzakis, and A. Argyros. *Vision-based hand gesture recognition for human-computer interaction.* The Universal Access Handbook, Human Factors and Ergonomics. Lawrence Erlbaum Associates, Inc. (LEA).

[5] M. Asaari and S. Suandi. *Hand gesture tracking system using adaptive kalman filter.* Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on, 29 2010-dec. 1 2010, pp. 166 171.

[6] H.K. Lee and J.H. Kim. *An HMM-Based Threshold Model Approach for Gesture Recognition.* IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 21, no. 10, pp. 961-973, Oct. 1999.

[7] Chang Yuan. *Using Large-size 2D Displays to Create 3D Hyper-Realistic See-Through Experiences.* iris.usc.edu.

[8] R. Kooima. *Generalized Perspective Projection.* LSU Computer Science and Engineering Division. June 2009.

[9] Sandin, D., Margolis, T., Ge, J., Girado, J., Peterka, T., DeFanti, T. *The Varrier Autostereoscopic Virtual*

Reality Display. ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2005, vol 24, no 3, pp. 894-903.

[10] P. Goorts, S. Maesen, D. Scarlino and P. Bekaert. *Bringing 3D vision to the web: Acquiring motion parallax using commodity cameras and WebGL.* 2013 International Conference on 3D Imaging, pp. 1-6, Dec. 2013.