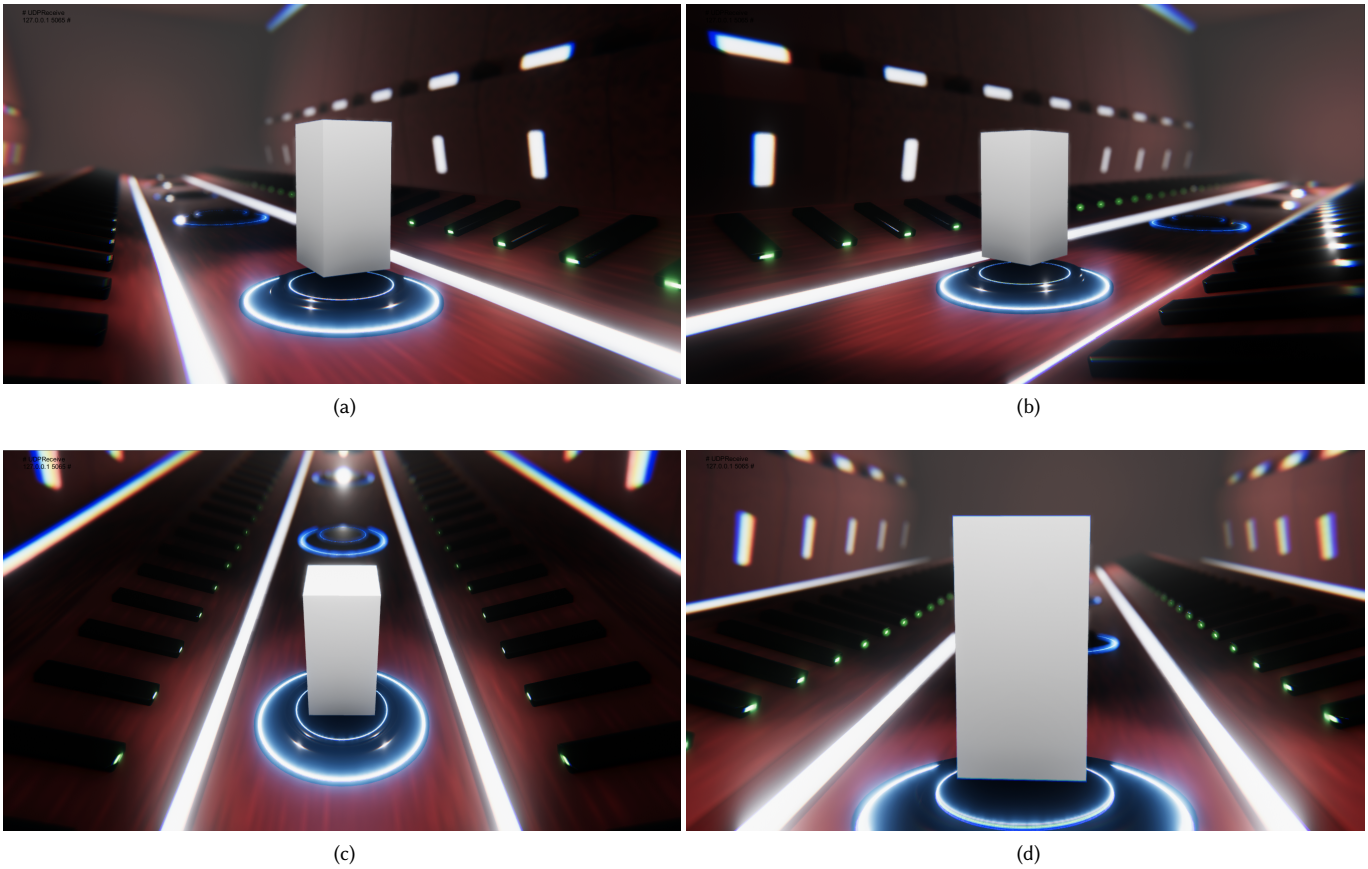
Mapping head motion to a camera in a 3D scene

Source code: https://github.com/J0ey1iu/enhanced-screen

HAITONG LI*, JIAYU LIU*, and YUQI YANG*, University of California, Davis



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

Fig. 1. (a) When a user tries to see the left side of the 3D model by moving his head to the left. (b) When a user tries to see the right side of the 3D model by moving his head to the right. (c) When a user is tries to see the top side of the 3D model by raising the altitude of his head. (d) When a user is tries to get a closer look at the 3D model by moving his head toward the screen(where the webcam is at).

In this paper, we implement a game project taking face movements as input and map them to a camera in a 3D scene. The goal of the project is to offer users a creative and interactive way to look at a 3D model. We manipulate the input to control the position and focal length of the camera so that users are able to move their head to adjust the perspective and how close they look at the model. Linear interpolation is used to smoothen the movement of the camera to offer a better experience. We also present a test case and an evaluation of our approach in this paper.

CCS Concepts: • **Computer Animation → Animation Control**.

Additional Key Words and Phrases: game control, face tracking, linear interpolation, camera control

1 INTRODUCTION

Head tracking [5] often used to simulate the experience of freely looking around in virtual (VR) [3] or augmented reality (AR), allowing the user to experience an immersive and natural way to look

*All authors contributed equally to this research. The order follows the alphabetic order of the last names.

Authors' address: Haitong Li; Jiayu Liu; Yuqi Yang, University of California, Davis, Davis, California.

XXXX-XXXX/2020/N/A-ARTECS279 $15.00
https://doi.org/why-cannot-i-delete-this

around in virtual environments. It supports and enhances human-computer[9] interaction. The concept of head tracking is commonly used in games where the player's head movements [1] are tracked and changes are carried out in game controls [6] as per the head movements.

The drive of implementing this project is to go through the process of how to project our head movement into the computer and make interaction [2] accordingly from scratch. To have a better sense of the pros and cons of this technology.

Our goal of this assignment is to successfully implement the basic functions of enabling an application to recognize and identify a user's head movements and present a freely perspective of the 3D scene [8] based on head movement. Besides, optimizing[4] this application to track the head movement smoothly and accurately.

This paper is written in the same order as we implemented this assignment. Firstly, it introduces the methods of face detection and data processing on the back-end. Then it dealt with the process of data sending and data receiving between the back-end and front-end. After that is how to simulate front-end via unity. Finally, analyzing the result and delivering the conclusion.

2 IMPLEMENTATION

In this section, we introduce our approach to implement the project in detail.

2.1 Face detection and tracking

To detect the user's head, we choose to do face detection since when a user is using our system, he needs to look at the screen. And typically, the webcam of a computer, or laptop, is at the top of the screen. So we can capture the user's face using the built-in webcam in most cases.

2.1.1 Choice of detecting approach. Our ultimate goal is to map the movement of the head to a camera in a 3D scene. So we have to use a face detection method and a tracking method that can work well in a very short amount of time. There are quite a few choices out there for this task. A classic approach is the haar-cascade classifier, which can run in real-time on CPUs. OpenCV offers a well trained classifier for detecting faces. But the problem of this approach is that it provides too little information about a face, and since it can only generally detect faces, it can only provide a vague region where lies a face. This works well when detecting faces on a picture. But when it comes to continuous video frames, it can only provide jittering position movements, which is not ideal when we are trying to map the movements to a camera. OpenCV also provides a deep learning based face detector. While it is robust and runs fast on CPUs, it provides, again, too little information about a face.

In order to get accurate position of the face, we eventually choose to use face landmark detector provided by dlib library. It can detect 68 landmarks on a face with good accuracy and speed.

2.1.2 Tracking. After some detecting and tracking test, we realize that the point with index 33 in figure 2 is very accurate in case of indicating the position of the face no matter how the user changes his facial expressions. So we use the position of point 33 to define the position of the face in a frame. While there are a few real-time point tracking and object tracking approaches, such as KLT feature tracker[7] and mean-shift tracker[2], we didn't use any of those. Since our project is designed for a single-user scenario and the face landmark detection is providing reliable result, we simply connect each position detected in each frame to construct the trajectory. In order to allow users to control the zoom-in level by moving their head closer or further from the screen, we also need to capture the information for z axis. Since webcams don't have the hardware support for detecting depth information, we detect how far the user's face is from the screen by calculating the area that his face covers. We multiply the distance[1] between point 0 and point 16 by the distance between point 27 and point 8 to define the area of the face. This way we can partially eliminate the influence of face rotation.

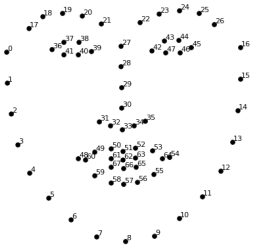


Fig. 2. 68 face landmarks that can be detected by dlib library

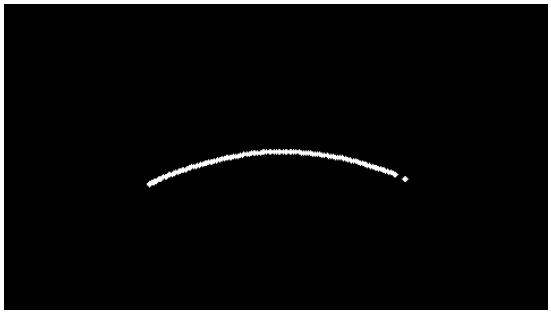


Fig. 3. Trajectory created by drawing point 33's position in each frame when the user's head moves from left to right. The gap at the end of the trajectory is caused by failing to detect a face in a few frames.

2.1.3 Data transmission. In our project, we use python to do face detection and tracking, but we built and display the scene in unity. So we need a way to transfer the tracking data from python to unity, which uses C sharp scripts. We do this by sending the tracking data in the form of a string to a local port using UDP protocol. Then download the data using C sharp and parse the string to the data types we need. The process is shown in Figure 4.

2.2 Movement mapping

In the project, we tested out two ways to map the movements. To offer users more controls, we bind the arrow keys to control the position of the camera. We map the change of the position of the

[1] Euler distance

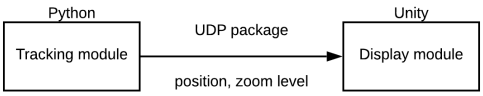


Fig. 4. The data flow of the project.

head in 2D to the change of the position of the camera in a 2D plane. And we map the change of face area to the change of focal length of the camera. The mapping process can be generalized as Algorithm 1.

2.2.1 Relative mapping. The first approach that we come up with is to map the position offset vector between two webcam captured frames to the 3D camera, and we call this relative mapping. It is relative because users can start moving at any position in the frame and stop at any position in the frame, only the offset vectors are recorded. The exact position of the user's head does not matter. The movement of the camera is controlled by the offset vectors, so the key input can work together with the head position input.

2.2.2 Absolute mapping. Another way that we come up with, also the way that we eventually adopted, is to map the exact position of the head in a frame to the camera, and we call this absolute mapping. In this approach, the position of the user's head in a frame does matter. We first store the position of the user's head in the first frame captured, then always calculate the offset vector between the original position and current position and use this vector to control the movement of the camera. This approach seems to give users less freedom to move around, but a major reason to adopt this approach is that it is easier to smoothen the camera movements. Additionally, since the status of the camera is controlled by the position of the user's head, arrow keys can't work together with this approach.

2.2.3 Smoothen camera movements. As we can see in figure 3, although the overall trajectory is smooth, jittering effects still exists. When we mapped the raw data to the camera, we noticed some camera shaking effects, much like some camera apps adding effects to your face–they shake in a very small range, but still noticeable. We thought the outcome was not ideal, so we applied linear interpolation to smoothen the mapped camera movements. The reason to choose linear interpolation seems to be counter-intuitive since obviously cubic, or quadric interpolation can better make smooth curves. However, they need a series of existing points to do that, and they take more time than linear interpolation. By using linear interpolation we can only take current status and desired status of the camera into account, without store all status for cubic or quadric interpolation, which will slow down the program since the status data accumulates over time. In order to do linear interpolation, we need the start and end status of the camera. This is where absolute mapping shines. While relative mapping seems to offer users more freedom to move around, we cannot get the exact status of the camera since it is controlled by offsets. However, absolute mapping can provide enough information to do linear interpolation. We can get current camera position and zoom level, as well as desired camera position and zoom level(mapped from current head position and face area in a frame). We do linear interpolation between current status and desired status of the camera and always choose a status that is near current status to be the next status, so that all movements would be smoothen out. But at the same time we introduced a noticeable delay of the movement mapping. We can tweak the time parameter in the interpolation to adjust how much detail or delay we want for the movements. Generally speaking, the higher the delay, the smoother the movements .

2.2.4 Parameters. To some degree, parameters are the key to make this project look good, and indeed they are. We spent quite some time to tweak the parameters to make map the head movement naturally to the camera. Most webcams capture 1080P videos, and that is too large to perform computer vision. For the efficiency of face landmark detection, we resize the resolution to $480 * 270$. All the parameters that we are about to introduce are based on this capture resolution. We noticed that if we map the raw position data to the 3D camera, the movement would be too drastic. So we multiply the raw position offset vector by 0.08 to make it suitable for camera movements. To smoothen out the movement of the camera, we set the time parameter to be 0.125 for linear interpolation, which is pretty close to current camera position. The calculated area of the face in a frame punctuates drastically so we multiply the raw data by 0.001 to make it suitable for camera focal length. We set the time parameter even closer to current focal length with 0.1 to eliminate the shaking caused by jittering data input.

Algorithm 1 Mapping

$posPara = 0.08$
$zoomPara = 0.001$
$p = getPackage()$
$origPos, origZoom = parse(p)$
while $True$ **do**
    $p = getPackage()$
    $curPos, curZoom = parse(p)$
    $rawOffset = curPos - origPos$
    $zoomOffset = curZoom - origZoom$
    $desiredPos = origPos + rawOffset * posPara$
    $nextPos = Lerp(curPos, desiredPos, 0.125)$
    $desiredZoom = origZoom + zoomOffset * zoomPara$
    $nextZoom = Lerp(curZoom, desiredZoom, 0.1)$
end while

2.3 Unity setup

Our project includes a 3D scene built by us to explore our approach of mapping movements. The scene is built in blender and imported to unity. In unity, we create a simple cube as our object to observe. The cube is place on our pre-designed object holder. When writing the C sharp scripts for unity, we ensure that users can assign any GameOject they want via unity interface to make the object changeable. We manipulate the $LookAt()$ function of Unity camera object to ensure that our main camera always focus on the object that we are observing. We mainly used area light to light our scene

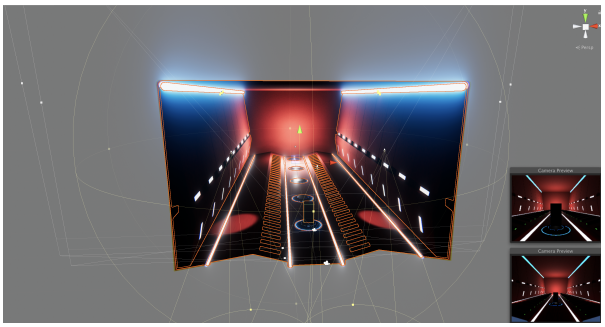


Fig. 5. The scene in Unity

in order to make the scene look better. We baked the lighting maps for static objects in the scene to allow area lights to work. Several real-time point lights are added to light up the object and the background, emphasizing the depth of the scene to magnify the effect caused by the change of the position and focal length of the camera. The post processing package helped us a lot when adding effect to the rendered images, including depth of field, glow and chromatic aberration. We also wrote scripts to make the spot lights point to a editable object to make it easy to change objects.

3 RESULT

The result of our project is shown in figure 1. The user sit in front of the laptop and run the python program to starting tracking his head. The tracking data will be send to a local port, in our case, port 5065. A script in Unity create a thread to continuously download data from the port and parse the data into the information we need for moving the camera. After the configuration that we mentioned in the implementation section, we are able to make the project pretty intuitive. Due to the limits of the face detection approach, there are some cases when the python program cannot find a face in a frame. Camera movement will smoothly stop when this happens, and will move back in corresponding place when the face can be detected by the camera again. Since our project is designed for observation of a 3D model, we present a video[2] to show the result in a more visual way.

When playing around with the project, we surprisingly found that our approach can be used to implement a gyroscope software for laptop. Gyroscopes are now built-in for smartphones and have inspired new ways to control games and look at things(such as skywalker app). For some reason, gyroscopes are not built-in for computers. Obviously we cannot move and rotate our PC to try to use the functions of a gyroscope, but probably we can try that on a laptop. Maybe somebody can figure out an interesting way to manipulate this feature of our project.

4 CONCLUSION AND FUTURE WORK

4.1 Conclusion

This paper was motivated by the need of an efficient and effective way to track head movements and map them into 3D scenes. Our objective is to develop a method that is able to take the user's head movement data as input and adjust the perspective and position from which the user will be observing the 3D object accordingly. At the point of this paper, we have constructed a head tracking system which uses OpenCV and dlib libraries to handle the face detection and tracking process, absolute mapping augmented with linear interpolation to deal with the mapping of movements to the 3D scene, and Unity to render the 3D scene and visualize the real-time changes in the user's viewing perspective. The results show that the proposed head tracking system performs accurate head detection and sensitive motion tracking, and it is able to render the changes in perspective without a noticeable delay. The completed work has demonstrated the feasibility of efficiently mapping head movement into 3D scenes.

4.2 Future Work

In the future, more work can be done to explore more use cases with different scene setups. Using the proposed system in real game scenes with more complex background objects or letting it be a part of a gyroscope software on computers may be an interesting and potentially useful direction to further investigate. Another possible direction is to incorporate the proposed head tracking system into a web application, which allows the user to get real-time head tracking experience on a web page. The web application can also be further improved by enabling backgrounds that can be customized by the user. Some real use cases may be the user recording some specific movements on top of the selected background. The system can also be used for web games for the players to have more realistic visual experience.

REFERENCES

[1] Sumit Basu, Irfan Essa, and Alex Pentland. 1996. Motion regularization for model-based head tracking. In *Proceedings of 13th International Conference on Pattern Recognition*, Vol. 3. IEEE, 611–616.

[2] Gary R Bradski. 1998. Computer vision face tracking for use in a perceptual user interface. (1998).

[3] Sing Bing Kang. 1999. Hands-free interface to a virtual reality environment using head tracking. US Patent 6,009,210.

[4] Marco La Cascia, Stan Sclaroff, and Vassilis Athitsos. 2000. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3D models. *IEEE Transactions on pattern analysis and machine intelligence* 22, 4 (2000), 322–336.

[5] Sourabh Niyogi and William T Freeman. 1996. Example-based head tracking. (1996), 374–378.

[6] George V Paul, Glenn J Beach, Charles J Cohen, and Charles J Jacobus. 2006. Real-time head tracking system for computer games and other applications. US Patent 7,121,946.

[7] Carlo Tomasi and Takeo Kanade. 1991. Detection and tracking of point features. (1991).

[8] Paul J Travers and Ying S Yee. 1994. Head tracking apparatus. US Patent 5,373,857.

[9] Ynjiun P Wang and Timothy P O'hagan. 1998. Head tracking system for a head mounted display system. US Patent 5,742,263.

[2] https://vimeo.com/399046413