3D brain visualization using augmented reality

Team #25: Adrien Hadj-Chaib, Raphael Nacache, and Hoang Nguyen

1 Introduction

Brain visualization in augmented reality has for aim to enable someone to see a 3D model of someone's brain in real time, on the person itself. That is, see an overlay 3D brain model where it is supposed to be as a 3D object embedded in the person's field of view. 3D brain visualization is applicable to many fields such as tumor removal surgery. Before such procedure, a patient undergoes multiple X Ray, MRIs and other imaging techniques to enable the surgeon to image the patient's brain. This data is then used to identify the tumor as well as to visualize the blood vessels and skull of the patient. Those images can then be used to recreate a 3D model of the patient's brain and enable the surgeon to have a better understanding of the situation. However, such data is not very practical as it is all on a side screen and the surgeon would have to keep looking at an external side screen to operate. Moreover, current brain surgery heavily relies on X Ray imaging with a contrast agent to visualize brain and blood vessels of the patient, which is damaging to the patient's brain.

Multiple approaches have been taken in order to tackle this issue. One initial approach was to use the Neuronavigator [1]. However, in addition to cost more than $1,000,000 , this machine is not portable and can only be used in the operating room.

Doctor Naoki Kaneko [2], devised a much lighter approach by using some mono augmented reality glasses to visualize the patient's brain. He would first reconstruct the scans in 3D and place them in the correct place relative to one another. He then placed some virtual markers on this 3D model as well as real markers on the patient's head. The glasses would then map the real markers with the virtual ones and would allow professor Kaneko to see the 3D model embedded in the patient's head. The problem of this approach is that is uses some physical markers to visualize the brain, which can be cumbersome in a surgery setting.

One last attempt to solve this issue had been done by NovaRad using Microsoft's Hololens [3]. The Hololens are some stereo augmented reality glasses, which are able to compute the distance from the glasses to the object seen without physical markers. NovaRad gained FDA approval thus enabling doctors to see a 3D representation of the patient's brain in their fields of view [4]. Another strength of this approach is that it is lightweight and also enables doctors to go through the patient's MRI scans and see them placed where they are supposed to be. One downside of NovaRad's product is the price of the Hololens glasses, which is around $3,500.

The aim of this project is to make an inexpensive and lightweight product using a mono camera system. The reason NovaRad uses a stereo camera is because it enables them to extract the depth component of the visualized image, which enables them to replace the brain model directly in the 3D space and project it on the viewing screen. Mono camera systems are limited by a 2D image and it has historically been near impossible to extract depth from an image, thus requiring some work-around like using a physical marker. Fortunately, recent advances of Machine Learning produced algorithms capable of predicting the distance between the camera and an object based on a 2D image along with the camera's lens specifications [5],[6].

Our product will thus combine Machine Learning for image analysis and Computer Graphics for the 3D brain model rendering. We will first gather the camera 2D image and run some Machine Learning algorithms on them to extract facial features [5],[6]. Such features such as eyes, ear, nose or mouth will enable us to be able to position, scale and rotate the brain correctly as we know where those specific points are supposed to be on the the corresponding 3D model. However, only having the 2D coordinates of the facial features only allows you to put the 3D model on an axis coming from the camera, going through the image and extending in the third dimension. Thus, thanks to advances in Machine Learning, we are then able to place the 3D model correctly in the 3D space, at the right depth along the axis previously described.

Finally, once the brain has been positioned in the 3D space, we can render a sample 3D brain model stored as an STL file, which is common use in hospitals. The rendering will be done in OpenGL and we will project the 3D model back on the same plane as the camera image since our viewing medium will be a classic 2D screen. The resulting image will be the original picture taken from the camera overlapped with a transparent 3D brain model where it is supposed to be, thus giving the impression of 3D brain visualization. This approach can also be generalized to a 3D viewing medium such as mono augmented reality glasses, as we will know the exact coordinates of the brain in the three dimensions. In that case, projecting the brain model back on the screen will not be necessary.

2 METHODS

2.1 Early implementation using Unity

Our project consists in helping doctors visualize their patients brain. We decided to use augmented reality to achieve this goal. Augmented reality is so amazing because it lets you place 3D objects in your field of view. Today surgeons use many different monitors to view the skull, the skin, and the veins. During a surgery the doctor is always looking back and forth at his patient and the monitors. The doctors field of view is constantly changing, and this increases the chance of having the surgeon make a mistake. Augmented reality keeps all the monitors in the doctors field of view. This is beneficial because the surgeon does not need to move his head anymore. Augmented reality is not new so some people have already tried to use it during surgery. One method was to use a physical marker, and to place a 3D object when the camera detected the marker. Unity [7] and Vuforia [8] made it easy for a user to create its own augmented reality. Vuforia lets you create physical markers which can be made with 2D images or even 3D objects and they are called the target images. After creating the target images, Unity lets you create the 3D object that you want to appear when the camera detects the markers. The software is easy to use, because it takes care of rotations, translations and scaling. But in our project we got rid of the marker and instead used facial features such as eyes, and nose for the camera to detect. This method is called facial detection.

2.2 Facial Feature Extraction with OpenFace

OpenFace is a C++ API that takes in a picture or video containing a human face, and outputs pixel coordinates of the persons facial features [5]. These facial features include facial actions, facial landmarks, and eye gaze direction. Our application uses this API to extract the pixel coordinates of a persons eyes and nose from an image, and to extract the approximate rotation of the head with respect to the the camera. By default, the OpenFace APIs speed was too slow for real-time image processing, so we had to condense the code such it it outputs only the pixel location of the eyes and nose, and the rotation of the head.

The utilities the API provides are data structures that is used to store faces. OpenFace puts these faces inside data structures, based on what the facial feature needs to be extracted. In order to obtain the eye coordinates, we used a data structure that stored all the coordinates of certain parts of the eye. The data structure used 6 points that detect the outer edges of each eye. In order to create a suitable overlay, we needed to get the average over these coordinates. By using the average, we trade off some performance for generality and accuracy.. By having more data points, our model has less variance of coordinates. But doing these averages requires more computation over larger data sets, which, although not as prevalent as a limiting factor of the framerate, can affect the scalability of the application. For example, if we wanted to create a more precise overlay, one that can be used practically by doctors in surgeries involving precise measurements, then this could be a limiting factor. For example, if we wanted to extend this application such that it can be used to overlay organs in the body, the model would require hundreds of data points in order to accurately depict everything onto the body. This large amount of data points would have to be processed at a rate of at minimum 30 frames per second, which may not be feasible. So although the data structures the API produces works for this application, the overall idea could be a limiting factor in the future.

In order to extract only necessary features of the face, we wrote a function that optimizes the creators implementation of image processing. Our implementation is optimized for head rotation, nose location extraction, and eye location extraction from a stream. The primary source of latency in the original code was caused from loading the models for every function call. This increases the amount of disk I/O the program performs. Our function only loads the trained models once. Let N be the number of images the program processes during its execution. Our program reduces the disk I/O reads from being of order O(N) to of order O(1). The other optimization our function performs is the calculation of only necessary facial features. The original code calculated facial actions and eye gaze direction, but our function removes the calculation of these unnecessary features. By removing these 2 feature calculations, we have removed 2N calculations from the program.

Another feature that our application used was a data structure known as a faces bounding boxes. The bounding boxes provides a set of coordinates that form a box around a face. This is primarily used to detect proximity and rotation. From this, we get our rotations as well as some coordinates known in the code as T_x, T_y, T_z. In our implementation, we obtained these features in order to create a suitable boundary around our face, which was used to help determine a set of boundaries for our skull.

2.3 Scene Composition and Rendering

The 3D rendering of the scene was made using OpenGL. We first wrote an STL file loader whose role was to read an STL file specifying vertices positions and normal vectors for a 3D shape. This program was used to load the 3D skull and its data was then loaded in a way so that OpenGL can recreate and display the skull. We then implemented a light source in the scene so that the skull appears illuminated and is self shadowed for a more realistic visualization.

Our next objective was to capture the webcam video feed and display it in the scene. We used OpenCV for the webcam capture and OpenGL for the video feed display. This was achieved by creating a screen whose dimensions were equal those of the webcam frame resolution, using two triangles. We then flipped the webcam frame so that it appears as expected to the user and treated it as a 2D texture. This 2D texture was then applied onto the screen

as shown in Figure 1. By doing this operation repeatedly, the scene was able to display the webcam feed onto the screen, thus giving the illusion of directly displaying the webcam video feed. In order to achieve an acceptable frame rate, the video capture and rendering had to be separated into two different threads. One thread would take care of the rendering loop of the whole scene while the other just acquired webcam frames and flipped them correctly before placing them into a vector. When the scene needed a new frame, it took the last frame from the vector and emptied it. This method allowed to achieve thread synchronization and produced a coherent video feed throughout. Finally, to give the impression of the video feed being full screen, we placed the 3D camera in front of the 2D screen and adjusted the distance for a FOV of 50 so that the screen takes exactly the whole of the cameras field of view, as shown in Figure 2.

We ran the OpenFace facial recognition algorithm once the frame was acquired and before it was displayed. The algorithm provided us with pixel coordinates of the left eye, right eye and nose in the picture space. These coordinates were then translated into world space coordinates easily because the screen was of a size equal to the screen resolution, centered at the origin and aligned in the XY plane. This allowed us to get direction vectors from the camera pointing to the facial features in the screen. The vector from the camera to the left eye was used to align the skull in the cameras Field of View. The skull can be placed at any distance on this axis as the implementation scales the skull so that it always appears at the right size, as seen in Figure 1. The skull can also be in front, into or behind the screen as the skull is always drawn so that it appears in front of the screen. This is achieved by first drawing the screen, clear the depth buffer, and then draw the skull.

Once the left eye of the skull is placed in the right axis, we use the OpenFace head rotation output to compute the head orientation plane, going through the left eye of the skull. We then compute the intersection point between the plane and the line going from the camera through the right eye position on the screen. We then compute the distance between the eyes to acquire the final X scaling of the skull. We repeat the same operation to compute the distance between the mid point between the eyes and the nose to get the final Y scaling of the skull, as shown in Figures 2 and 3. We then apply an X and Y scaling factor so that the distance between the eyes on the loaded model becomes the final distance between the eyes. We do a similar scaling for the mid point between the eyes and nose on the loaded skull model. The Z scaling factor is manually adjusted to adapt to the skull of the user.

Finally, we apply the head rotation to the scaled skull model and translate it so that its left eye goes to the computed final position of the left eye as shown in Figures 3 and 4. This puts the skull in the right position and scales it correctly with respect to how close the skull is placed from the camera and how close the user is from the camera, as shown in Figure 1. For more usefulness in a medical setting, we also implemented an optional transparency of the skull so that it doesnt completely block sight of the patients head while providing a useful overlay as shown in Figures 5 and 6. We then repeat the whole process for every frame that the scene rendering thread wants to display.

3 Results

Our method provides a working product to the user with an acceptable frame-rate and a bit of delay between a user motion and its appearance on the screen. This is due to the added computations necessary to render the scene and run OpenFaces facial landmarks detection and head pose estimation between every frames being displayed. The main thread does not use every frame the webcam provides in order to limit the delay. Instead, the main thread only takes the last frame acquired and flipped by the webcam capture thread. The frame-rate is mainly limited by OpenFaces computations, which we kept to the bare minimum. In comparison, rendering the scene is really fast and doesnt affect the frame-rate significantly.

Our implementation also strived to be depth independent. Indeed, the skull displayed on the screen scales automatically as the user gets closer or further away from the camera as shown in Figure 2. This is done by computing the distance between the eyes and nose in the picture, and scale the loaded 3D model so that it acquires the same dimensions. Hence, the skull also adapts its shape to the one of the user. The other depth independency comes from the fact that the skull will appear the same whether is is rendered in front, into or behind the screen. This means that we can pick any point on the camera-left_eye vector (as long as it is within the view volume) to place the left eye of the model and the rest scales automatically. This results in a bigger skull if the model is rendered further away from the camera and a smaller on if it is rendered closer. By effect of perspective projections, all will appear the same size on the screen as shown in Figure 1.

The application is limited by how well the OpenFace API detects nose and eyes. Our applications accuracy was measured based on how well it performed on faces that were far away, and how well it performed based on how far away the face was rotated from the cameras view. The result was that the application performs well on scaling the skull based on the faces distance from the camera, and performs poorly on rotated faces. Our application produces good overlays for distances up to approximately 4 feet. Because this application is targeted at doctors who interact with patients up within this range, this is an excellent result. Our application performs well given that the face is rotated +/- 45 degrees from the camera. This is an adequate result, given that this is a limitation of OpenFace. Further research and development of this application would want to improve this range.

Finally, although this paper only showcases the skull of the user, it is also possible to load blood vessels, a brain or a tumor for instance. This can be achieved by loading the corresponding models and align them manually to the

loaded skull. Once this setup is done, applying the skull transformations to all of the objects will place them in the correct place, with the same positioning relative to one another. The skull is always needed as it is the core basis of our implementation, but viewing it is optional as it can be made transparent, only leaving the brain or tumor to be seen as shown in Figures 5 and 6.

4 Discussion

Augmented reality is not new and some surgeons have already used it during surgery [2]. Their method consisted in placing physical markers on the patient. The camera would detect the markers and the 3D object would appear at a determined position relative to the marker. Our methods is different in that we used facial detection in order to get rid of the markers. It is very beneficial to not have markers, because placing the markers on the patient is very challenging and many problems can occur. For example if the patient moves and the markers change position or if the doctors misplaces the makers by a few centimeters then the augmented reality will not work correctly. Some surgeries must be done on the millimeter scale and thus very challenging with markers. Our method increases precision and gets rid of the markers. With the help of facial detection, the computer is able to identify facial features such as eyes, mouth, and nose. Now instead of having a physical marker, the new virtual marker is a facial feature of the patient. If the doctor is operating on the eye then instead of using the eye as the marker we could use the nose.

There are many ways to improve the performance of the project, the main ones being the frame rate, video delay and the rotation of the 3D object. In order to improve the frame rate and video delay, we could use a facial landmark detection tool that is lighter and faster then OpenFace, or rework OpenFace in depth to optimize the code and reduce the unnecessary work even more. OpenFace proved to be extremely useful in that is computes the rotation of the head, which hasn't been implemented by many toolboxes. Keeping the current OpenFace implementation, we could delegate the work of the facial detection to the webcam capture thread or an entirely new thread. However, this implementation will require passing data between threads and will create thread concurrency issues. Improving the 2D and 3D rotations will also involve reworking OpenFace as it often failed to produce correct facial landmarks and head rotation if the head was tilted too much or if the features used for detection were occluded. One can also improve the precision of OpenFace's predictions by doing a calibration of the camera to extract its homogeneity matrix as OpenFaces uses some of its values in the prediction. Hence, a calibrated camera will produce more accurate measurements and will therefore improve the precision of the 3D visualization.

Our project currently has one skull from an anonymous patient uploaded but the goal is to have a 3D model of each patients brain. Indeed, the current implementation scales this sample brain to try to match the one of the user, which is not ideal as skulls have many different shapes. The current implementation is able to scale the skull in the X (eye spacing) and Y (nose-eyes) direction, but lacks Z handling (forehead-back of the head). Another way to improve our project is to add ear detection as a Z-component, depth, to scale the 3D object so that the distance between the eyes and ear scales and matches, which will make it look more realistic. Finally, a limitation of the project is that the user cannot move too fast or rotate its head too much due to the latency and processing time, which makes the experience less interactive, but not unusable by any means.

5 Conclusion

In about eight weeks we were able to create an augmented reality software. We started using Unity and Vuforia, but we quickly changed direction because we wanted to be independent and make our own software. Unity is powerful and easy to use, but it forces its users to use physical markers. Surgeons have already used makers so we wanted to do something new and remove the markers. Our method does work but needs a lot of improvement before being used in hospitals and schools. The camera is too slow and the 3D object has trouble rotating correctly. The main challenge was to produce a working product in such a short time. Another challenge was that we used a regular camera from a computer. Our results would have been much better if we had used a 3D camera but it was too expensive. We believe that augmented reality can change the medical industry. Doctors will probably be using augmented reality like we use smartphones today. Our software can help surgeons not only in surgery but also in training schools. For example one could use augmented reality on a dummy, and test their surgery skills. Since our method is not precise on the millimeter scale we believe it can be helpful for training surgeons, but not for real surgeries yet.

References

[1] Products, S. and Products, O. (2019). Cranial Navigation - Brainlab. [online] Brainlab. Available at: https://www.brainlab.com/surgery-products/overview-neurosurgery-products/cranial-navigation/ [Accessed 6 Jun. 2019].

[2] N. Kaneko, K. Maruyama, E. Watanabe, T. Kin, K. Saito, A. Kumakiri, A. Noguchi, M. Nagane, Y. Shiokawa, Smart Glasses for Neurosurgical Navigation by Augmented Reality, Operative Neurosurgery, Volume 15, Issue 5, November 2018, Pages 551556, https://doi.org/10.1093/ons/opx279.

[3] D. Fornell, Two Technologies That Offer a Paradigm Shift in Medicine at HIMSS 2017, Imaging Technology News, 03-Mar-2017. [Online]. Available: https://www.itnonline.com/content/blogs/dave-fornell-itn-editor/two-technologies-offer-paradigm-shift-medicine-himss-2017. [Accessed: 28-Apr-2019].

[4] VIDEO: Augmented Reality for Surgical Planning Example, Imaging Technology News, 30-Nov-2018. [Online]. Available: https://www.itnonline.com/videos/video-augmented-reality-surgical-planning-example. [Accessed: 28-Apr-2019].

[5] OpenFace 2.0: Facial Behavior Analysis Toolkit Tadas Baltruaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency, IEEE International Conference on Automatic Face and Gesture Recognition, 2018.

[6] F.-J. Chang, A. Tran, T. Hassner, I. Masi, R. Nevatia, G. Medioni, "FacePoseNet: Making a Case for Landmark-Free Face Alignment", in 7th IEEE International Workshop on Analysis and Modeling of Faces and Gestures, ICCV Workshops, 2017.
[7] Technologies, U. (2019). Unity - Unity. [online] Unity. Available at: https://unity.com/ [Accessed 6 Jun. 2019].
[8] Ptc.com. (2019). Industrial Augmented Reality, Vuforia, PTC. [online] Available at: https://www.ptc.com/en/products/augmented-reality [Accessed 6 Jun. 2019].

6 APPENDIX

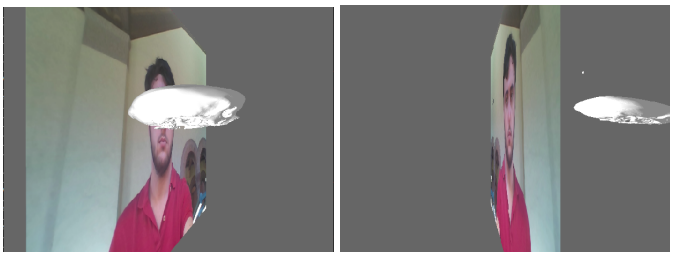

Fig. 1. View of the 2D screen from different places, at an angle different from the usual camera placement . By an effect of perspective and alignment, the skull appears as on the screen, at the right place and size from the initial camera's point of view. In that instance, the left picture has a left eye distance of 400 units away from the camera. On the other hand, the right hand picture has a left eye distance of 200 units.

Fig. 2. View of the skull as the user is closer and further away from the camera. The skull automatically scales to match the user's eye spacing and nose-eyes spacing.

Fig. 3. Skull Visualization when the user translates its head in the XY plane (left) and rotates it around the Y axis (right).

Fig. 4. Skull Visualization when the user rotates its head in the XY plane (left) and rotates it around the X axis.

Fig. 5. Skull with a transparent visualization superposed onto the user's head.

Fig. 6. An optional transparency of the skull can help focus on inside structures such as the opaque red tumor for instance.