CVIT-Honors Project

Topic

User perspective rendering for Handheld Augmented Reality

Project Members

Kamineni Vikas
Ashwin Pathak

Introduction

User Perspective Rendering

In AR, the most important feature is to create an illusion to the users that the virtual objects actually exist in the real world. By this, the users can easily recognize or differentiate the virtual objects which are in the real scene with minimum error. This illusion can be achieved by satisfying geometrical, photometric and temporal consistencies. In most of the AR systems, the images are rendered by the exact perspective projection at the optical center of the camera. This type of rendering is called *Device Perspective Rendering(DPR).* In DPR, there are Real scene and Real image inconsistencies. By this, there will be mismatches in users recognizing the virtual objects in the real scene thereby resulting in decrease of some sort of visibility or task performance. This misalignment is kind of important problem when it comes to AR, as the virtual content inside the device screen must be visually connected with the real objects outside the screen. This misalignment affects the performance during the interactions with the AR environment.

In order to create images without this inconsistencies, the AR systems should be able to map the real 3D scene and generate images according to the user's viewpoint. This is called *User Perspective Rendering(UPR).* UPR should actually create an impression as if the screen of the device is transparent.

Our aim in this project is to develop a *User Perspective system where we address the problem of geometric consistency between the displayed and real scenes in augmented*

reality using a video see-through handheld display(mobile device with front and rear camera).

Approach

The approach we followed is *Approximated user perspective rendering using Homography*. In our method, we generate each user perspective image by the homography transformation of captured image. The transformation is done so that each 3D point measured using the rear camera is transformed into a point which is measured from user's point of view.

We divided our work flow into different parts.

1. We have augmented a virtual object over real scene using some marker. This is the initial attempt where we aim to update the pose of the augmented object with respect to the user's viewpoint. So, we first augmented a virtual object on the real scene using Vuforia stones marker placed on some plane.

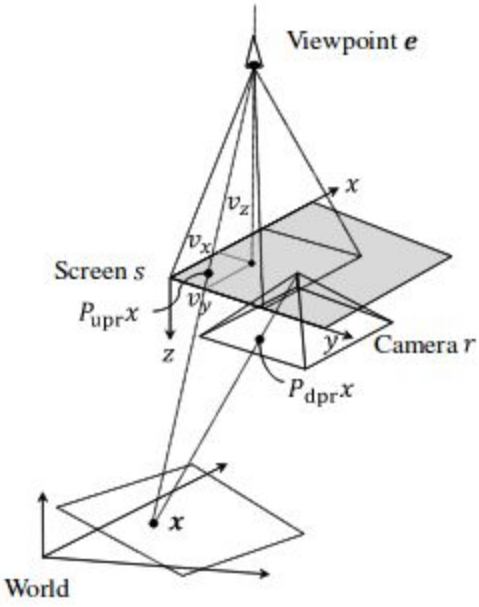
2. Then we tried to track user's eye position as we have to transform the augmented object w.r.t user's viewpoint. In order to find the user's eye position, we tracked the user's face using the front camera. For this, we used Android 2D face tracker, which returns the 2D coordinates of the face on the device screen.
3. Now, we have to convert the 2D coordinate of face on the device screen into real 3D point. For this, we propose a method for estimating the distance between the camera and the human head in 2D images from a calibrated camera. The 3D coordinates of the face can be represented by any coordinate system. We assign

3D positions of each point on face from standard data of faces. But, this way requires that the feature points in the standard data are fortunately the same points of the face tracker's output. Using the above method, we find the z coordinate the user's eye position. (x,y) coordinates are taken to be same as the 2D coordinate we obtained using face detection(previous step).

4. Now, the challenge we faced is that we have to track the real scene from the rear camera and the face from the front camera simultaneously. Unfortunately, all devices does not support this. As, we can only create one instance of camera at a point of time. We found that mobile devices having **Snapdragon 801** processor support simultaneous access of both front and rear camera. We used HTC One M8 which uses Snapdragon 801 processor for our project. Now, we are able to track the real scene using rear camera and the user's head position using the front camera simultaneously.
5. As we have got the simultaneous access of both rear and front camera, we now have to obtain the homography matrix which transforms DPR to UPR.

$$\boldsymbol{P}_{\text{upr}} = \boldsymbol{P}_e \boldsymbol{M}_{r \to s} \boldsymbol{M}_r$$

Where, P_e is the perspective projection matrix of the view frustum formed by the viewpoint $\boldsymbol{e} = [\, e_x , e_y , e_z \,]^T$ and the screen surface as shown below.



The matrix P_e can be represented as follows,

$$\boldsymbol{P}_e = \begin{bmatrix} -e_z & 0 & e_x & 0 \\ 0 & -e_z & e_y & 0 \\ 0 & 0 & 1 & -e_z \end{bmatrix}$$

And M_r is the Extrinsic Parameters which we obtain using SolvePnP in opencv,

$M_{r \to s}$ is the relative pose of the rear camera *r* to the screen *s.*

6. Next task is we have to track real scene for calculating the pose of the camera(M_r). Pose of the camera is defined by their Extrinsic and Intrinsic parameters. Actually, in the step where we have done the augmentation of virtual object, **Vuforia** must have calculated the camera pose in order to augment the object. But, we are not able to extract it out from Vuforia. So, we have done the following to find the camera pose.
    a. To calculate ***Intrinsic Parameters***, we used normal chessboard as marker and followed the standard approach of camera calibration using checkerboard.
    b. To calculate ***Extrinsic Parameters***(R and T), we used ***SolvePnP*** method in opencv which returns Rotation and Translation vectors. We used a marker in the real scene to track frame to frame. We first detect feature points in the *Reference image*(initial frame). Then, we match these feature points in next frame(*Query Image*). If there are sufficient number of matches, we construct a ***Homography matrix*** using the matches between current and previous frame. Then, we find inlier points using the Homography matrix by applying threshold. Now we have, SourceInlierPoints which are inliers in the Reference image and QueryInlierPoints which are the inliers in the Query Image.

       These inlier points are on 2D device screen. So, we have to find 3D SourceInlierPoints. To obtain the 3D points, we project *SourceInlierPoints onto a plane floor(75 units) away from the device camera*. Now, these 3D points are used to *find the pose of the camera* in the next frame. We use inbuilt ***Solvepnp method*** in opencv,

       SolvePnP(Source3Dpoints,QueryInlierPoints,IntrinsicMatrix,T,R)

       Here, this function returns *Rotation and Translation vectors* denoted by T and R. The extrinsic parameters are Translation and Rotation vectors of the image plane to the device camera.
7. $M_{r \to s}$ is calculated as follows,

   $$\mathbf{M_{r \to s} = M_{f \to s} * M_{r \to f}}$$

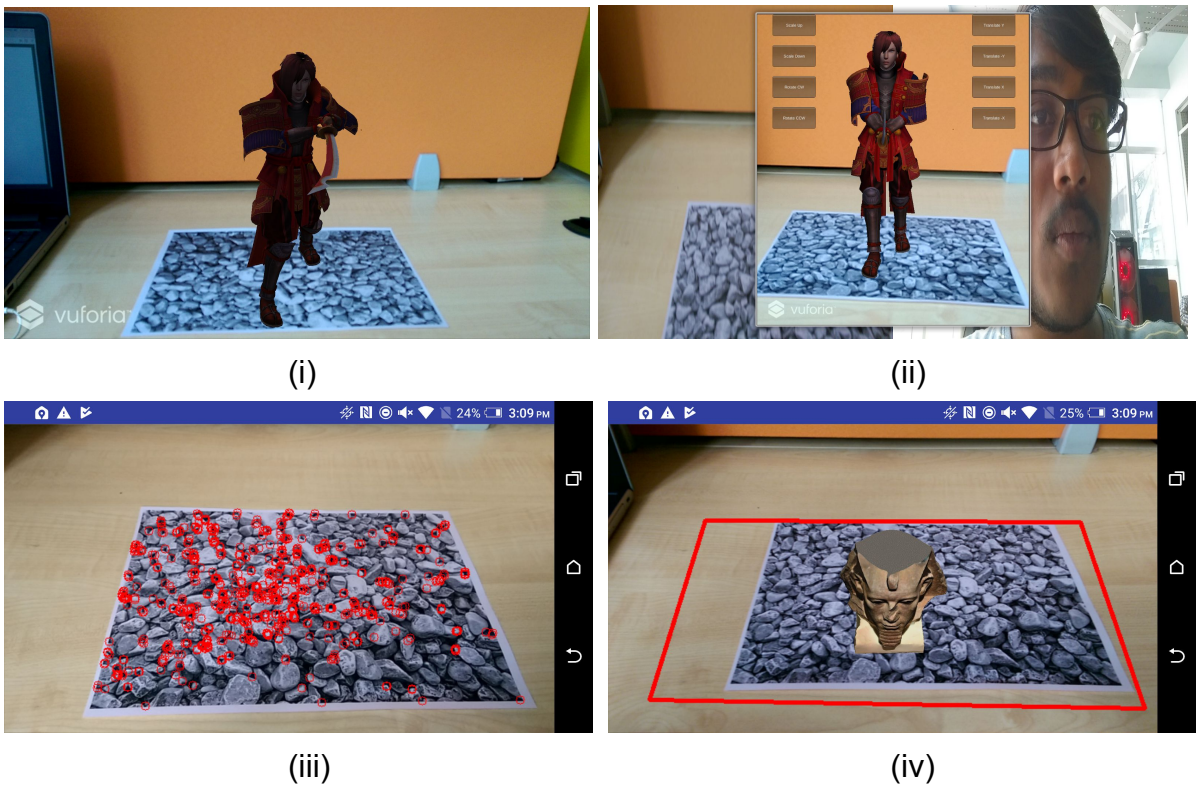
   Where, $M_{f \to s}$ is the relative pose of front camera to the screen s and $M_{r \to f}$ is the relative pose of rear camera to the front camera.
8. Now, solving the three obtained Matrices, we get P_{upr}. Next task is we have to find the Homography matrix which converts P_{DPR} to P_{UPR}. For this, we propose a

method which finds the optimal Homography transformation closest to the exact User Perspective rendering.

For a particular set of 3D points x, we minimize the following error function *E* to obtain the optimal *H.*

$$E(\boldsymbol{H}) = \sum_{\boldsymbol{x} \in F} w_{\boldsymbol{x}} d(\boldsymbol{P}_{\mathrm{upr}} \boldsymbol{x}, \boldsymbol{H} \boldsymbol{P}_{\mathrm{dpr}} \boldsymbol{x})$$

By this, we get an Optimal Homography transformation which converts the 3D points in the real scene from Device Perspective to User Perspective.



(i) (ii)

(iii) (iv)

Figure: Results at some of the above steps. a) augmented a virtual object into a real scene using stones image as a marker in Vuforia. b) Simultaneous tracking of front and rear camera preview and the front camera returns the 2D face coordinates by face detection and the rear camera tracks the real scene. These experiments are done on HTC One M8. c) Calculating the pose of the camera by extracting feature points from the scene and tracking them to calculate the pose. d) Pose of the camera is calculated, and we have augmented a virtual object onto the scene using the intrinsic and extrinsic parameters of the camera.

Related Works and Analysis

Most of the applications uses device-perspective approach for rendering the Augmented Object, however, that approach is different from what we wanted as our approach should be responsive to the user's motion. Thus, our approach required us to move to the user-perspective rendering part as required for our project.

Baricevic valuated the effects of display size on UPR and found that UPR out-performed DPR for the selection task. These types of tasks are very much similar to what we want in our project and hence, shifting to UPR was required. They initially used kinect and Wiimote for head tracking and reconstruction purposes, however that was slower and not that efficient.

Tomioka and Hill then later proposed a better algorithm using homographies to transform the back-facing camera with respect to front-facing camera's detection of the face using 3-D head tracking.

There were however, initially various versions of UPR suggested and they were properly benchmarked henceforth. FUPR was one of them. Fixed Used Perspective Rendering involves, instead of tracking the users head pose the authors manually measure the distance of the head to the device once at the beginning of the application, and they assume the user looking perpendicular through the center of device over the entirety of the application which is completely irrelevant in our case as we are interested in the complete movement of the user's head pose. Hence, it fails to compute the large interaction spaces that is there.

We wished to include FUPR with Kanade-Lucas-Tomasi (KLT) tracker. We here use KLT-tracking to estimate the head position in image space, which we use to subsequently decide whether the parameters of FUPR needs to be refreshed.

Along with FUPR and KLT, we were also interested in Approximated user perspective rendering (AUPR) using homography. Using homography we wish to achieve the required transformation from DPR to UPR. Along with homography there is another approach which is related to the geometry, it uses dynamic frustum to use the field of view and orientations in order to achieve the required transformations.

Software and Hardware Prototypes

Hardware

For the hardware, we have used android based devices with SnapDragon 801 support because simultaneous access of front and rear cameras. Adding to this for the app development, we used HP laptop with intel-i5 processor, 4GB RAM and Linux OS.

Software

For the augmented object rendering, we use Vuforia software which supports AR and is open-source. Next, for app development, we used android studio.

Challenges and Limitations

While implementing we have to convert the obtained 2D face position into 3D world point. The method we proposed requires a particular dataset where there are mappings from each feature point of face in 2D image to its corresponding 3D point on face(fiducials). We use fiducials for some set of face images(sort of training) in order to compute its 3D point on face. Then, we calculate the distance from camera by solving Perspective n Point(PnP) problem. And then, if a new face is given, we use obtained distances and approximate its distance from camera to be the average of the camera distances across all exemplars. So, in order to achieve this we need to have access to that dataset. We got to know that the **Texas 3D Face Recognition Database** is suitable for our requirement. But, we are not able to get access to this database.

Also, we are stuck at implementing the $\mathbf{M}_{r \to s}$ matrix part. As mentioned above,

$$\mathbf{M}_{r \to s} = \mathbf{M}_{f \to s} * \mathbf{M}_{r \to f}$$

Where, $M_{f \to s}$ is the relative pose of front camera to the screen s and $M_{r \to f}$ is the relative pose of rear camera to the front camera. We have faced difficulty in order to find, $\mathbf{M}_{f \to s}$ and $\mathbf{M}_{r \to f}$. As, there was no clear mention of specific implementation details related to finding $\mathbf{M}_{r \to f}$ and $\mathbf{M}_{f \to s}$**.** We also had a brief discussion with the author, he said that initially they implemented it using the inbuilt design parameters of that particular device(phone). But, in the later stages, they calibrated $\mathbf{M}_{f \to s}$ and $\mathbf{M}_{r \to f}$ using opencv SolvePnP. But, a clear explanation regarding its implementation is not given.

Another challenge we faced is that Opencv Camera cannot be used for simultaneous access of both front and back camera. So, we have to change to inbuilt Camera API in android and implement face detection.

Some of the limitations are we did not implement 3D face tracking. We have not considered the case where the user's head is looking at the screen at some angle. We just calculated the eye position along x,y,z direction and updated the frame according to it. Here, x,y are considered to be same as the 2D coordinate we get from face detection. We have not taken the angle of view into account.

Also other limitation is that as we are updating for each frame, the result will be quite slow as it is doing computation for each frame. To overcome this, the approach we propose is that we update the frame when the change is more than some threshold. We do head tracking from frame to frame and compute the difference between images in each frame. Then we reflect the change only when the difference between the images is more than some threshold. Below is a rough explanation of thresholding technique.

```
E ← |PosEyeCalc − PosEyeFlow|
ΔE ← |PosEyeFlow_last − PosEyeFlow|
if E > ε OR (ΔE < ε * 0.1 AND !isPrecise) then
     recalculateFacePose
     isPrecise ← TRUE
Else
     isPrecise ← FALSE
```

References

1. Peter Mohr, Markus Tatzgern, Jens Grubert "*Adaptive User Perspective Rendering for Handheld Augmented Reality*". https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7893336&tag=1
2. Makoto Tomioka, Sei Ikeda, Kosuke Sato "*Approximated User-Perspective Rendering in Tablet-Based Augmented Reality*". https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6671760
3. Ali Samini, Karljohan Lundin "*A Perspective Geometry Approach to User-Perspective Rendering in Hand-Held Video See-Through Augmented Reality*" http://weber.itn.liu.se/~karlu20/papers/samini14perspective.pdf
4. Makoto Tomioka, Sei Ikeda, Kosuke Sato "*Pseudo-transparent tablet based on 3D feature tracking*" https://dl.acm.org/citation.cfm?id=2582103
5. Arturo Flores, Eric Christiansen, David Kriegman, and Serge Belongie "*Camera Distance from Face Images*"

https://vision.cornell.edu/se3/wp-content/uploads/2014/09/camera_distance_from_face_images.pdf

Code Link

GitHub Link: https://github.com/Vikas007Vikas/Honors