Augmented Reality on Multi-Devices

Jiyue Wang, Shaohan Xu, Wei Xia
Stanford University
450 Serra Mall, Stanford, CA 94305
{jiyue, shao2, wei4}@stanford.edu

Abstract

In this project, we built a multi-device marker-based augmented reality system on iOS. As most of the existing AR apps in App store support only one device, it will be more fun if nearby users can share a view and interact with each other. We implemented a demo version of the system in which users can touch at the screen and change the color of the rendering object and the scene will be synced with connected users.

1. Introduction

We have explored many AR apps in App store and most of them use a printed marker to help register the virtual object on the real world image. This marker-based technique is described in [3]. The idea is to analyze the distortion of the rectangular marker and use this information to overlay virtual object on the captured image. Using different markers, we can identify a huge number of objects. One benefit of using marker is that it doesn't need a 3D sensor to measure the position or orientation of the device. However, it is not always convenient for users to print a marker. On the other hand, non-marker based technique, without extra sensors, may suffer from inaccurate register and high computational power. [2] gives a feasible solution to this problem. In our project, we built a multi-device marker-based AR app on iOS. As far as we know, most current marker-based AR apps only allow one device and it will be more fun if another device can join in.

2. Background and Outline

2.1. Review of previous work

Augmented reality is very popular especially in game developing. With the help of AR technology the surrounding real world of the user becomes interactive and digitally manipulable, which is really cool. Games with augmented reality have been available in App store for a long time. "ARBasketball" established marker based AR system and allows user to play basketball shooting games as long as they have a marker with them. "Toyota 86 AR" is to make users feel and exprience the real car by practicing driving a virtual car around a virtual course. They both obtained really good marker tracking performance. Inspired by this, we decided to add a multi-device mode.

2.2. Contributions

We built a marker-based AR app that allows users to share the same view and interact with each other. This may serve as a demo as well as an inspiration for later game development. Several examples are chess, Tic-tac-toe and card games, etc.

3. Technical Details

3.1. Marker Detection and Tracking

The system first captures the camera view and then sends it to the detection and tracking pipeline. The marker detection is developed based on open source libary "ocv_ar"[1]. We will first discuss this library and then we will talk about things to be improved. Our first try is the bitonal marker as they are cheap to detect and identify. It takes 15ms on average to proecess one frame, which makes it real-time and thus the detection is performed per frame. The marker detection pipeline can be decomposed as following:

Figure 1. Tracking pipeline

The preprocessing includes converting camera frame to grayscale image and computing the gaussian pyramid. The gaussian pyramid is used in the second stage where the system converts the grayscale image to binary image. Instead of using a global threhold to get the binary image, we used an adaptive threholding method to make the system

robust to high dynamic of global luminance change. Adaptive threshold gets threshold values based on pixel values of each block size. The next step in the pipeline is to do marker identification. First of the all, the marker we use is of size 7x7 pixels:

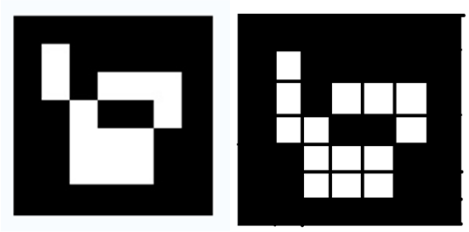

Figure 2. marker(id=111)

With all black on the boundary, markers can be easily identified. The centering 5x5 pixel block is for encoding.For each row of the 5x5 pixel block, it is encoded using 2 bits. All the possible code are 00→10000, 01→10111, 10→01001, 11→01110. Because of the characteristic of the marker, it is attempted to use shape analysis to separate the marker from other objects. OpenCV inbuilt function "approxPolyDP" is helpful at this step to analyze each contour shape and select only convex quadrilateral as candidates for marker.

For each candidates, the system first computes perspective transform and warp the marker candidate to the reference marker image. To filter out the wrong candidates, the system first tests whether the boundary pixel is all black and whether each row has a valid encoding.

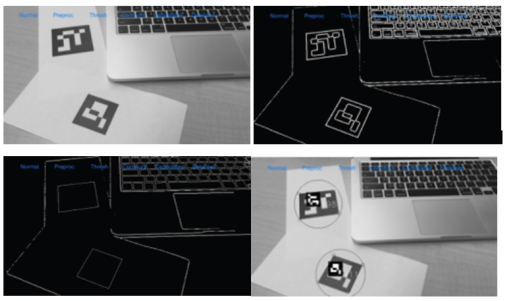

Figure 3. Result of each block

After the marker is identified, we need to estiamte the marker's position. OpenCV has built-in functions "solvepnp" to get the rotation and translation matrix nicely based on the following function:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & & cx \\ & fy & cy \\ & & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Above all this library handles well for the marker detection. However the overall detection time is about 25ms. Also, it does not works well when the iphone is in great motion. We identified that the pre-processing time and adaptive threshold are the main cause for the time consumption. We overcome this by directly scale the input image to smaller resolution instead of using gaussian pyramids. Because in our case, the most coarse pyramid is what we want to use. As to the second factor when the iphone is in motion, the detection is poor. The reason for this is that when the iphone moves, the marker is not as standard as it should be, which cause the shape detection fails. Our improvement for this one is that after we get the binary image, we will scale it down. The rule of scale is that for every 2x2 pixel block, the scaled down pixel will be white as long as there are more than 1 white pixel in the 2x2 block. This operation is similar to "dilate" operation. In this way, we can connect the dotting line and the detection is more stable with respect to motion.

3.2. Virtual Object Rendering

After the marker is detected, we can build a coordinate system attached to the marker. The next step is to add 3D objects to the scene.

We used the OpenGL ES to display objects on the scene. With Blender as our modeling tool, we can easily build our own model as well as access the rich online resources. We first tried on some simple geometry like cube and sphere. The obj file of the cube is exported from blender, and then converted to a cpp file that can be directly imported into our code. We also tried some more complicated models downloaded from Internet. So far we successfully displayed the monochromatic geometry of the model on the scene. However when it comes to texture mapping, things are not going on so well. As none of us has experience with OpenGL before, we decided to leave this problem for the future development.

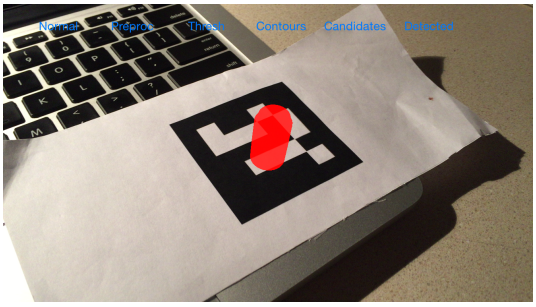

Figure 4. Object rendering

To respond to the user tapping on the screen and find out which object the user tapped on, we have to convert the 2D coordinate on the screen to the 3D coordinate in OpenGL. There are three ways to do that: color coding, selection

mode, and ray picking. Color coding, which we choose, assigns each object with a different color. When the user taps, render the objects on the back buffer, then read back the tapped pixel from the back buffer and check its color. The other two methods have their drawbacks. Selection mode in OpenGL is unfortunately not supported in OpenGL ES. Ray tracing is independent of OpenGL and requires different ray-object interaction algorithms for different geometries. There are still some error in OpenGL regarding this part. We have included the buggy code for reference.

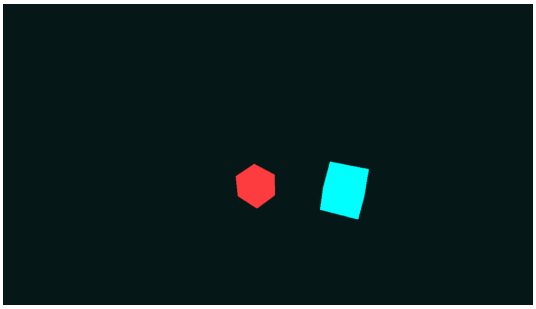

Figure 5. Color coding

3.3. Multi-Device Registration

In this App, users can choose single or multi-player mode to play with nearby users. We choose Multipeer Connectivity framework to communicate with nearby iOS devices through Wi-Fi networks, peer-to-peer Wi-Fi, and Bluetooth personal area networks. Due to band-width limits, we have to determine what kind of data should be synced through the network. Considering that we are using a binary mask, which makes it easy to detect, and we only need to send some state data in the game, it is more reasonable to detect the marker and compute different perspective tranform individually. When user interacts with the device, it then sends necessary data to peers for syncing the view.

In our implementation, a touch listener starts to work ss soon as the user starts the game. It captures the position of tapping on the screen and sends a random color information to the peers if necessary. If some device sends the color data, all the device will render the object in the new color at the next frame, making it look like viewing the same object from different devices.

As there is no host in the current system, there might be concurrency issue; however, it is rare in our scenario.

4. Experiments

We tested the system on varies devices including iPhone 5 and iPhone 6 with iOS 8.3 as deployment target. The system performs real time with about 15ms to process one frame. Here is a link to the demo: `https://youtu.be/r4sogq0fwzA`

5. Conclusion

None of us have experience in iOS (or Android), thus we spent a lot time on debugging iOS and building the system view. OpenGL framework in iOS is more complicated than we thought and its hard to debug. We successfully imported different geometry to the system, but there is still problem with texture mapping.

Currently, our mark-based tracking system is not very stable. Even occluding a small part of the maker can make the tracker lose target, as the square boarder is no longer complete. Moreover, when camera shakes, the blurred images may fail the edge detection. We can use a marker board to deal with the occlusion problem and alleviate the blurring issue.

We didn't use an image-based marker as we thought a binary marker should give faster and accurate result. However, as the binary marker worked worse than we expected due to the blurring issue, we decided to try optical flow tracking from project 2. Optical flow does not detect the marker each frame but tracks the existing marker, thus helping to deal with the occlusion problem. Although tracking fails when the marker is far away, it won't often be the case considering cell phone is a small device. This part is almost finished, but unfortunately we do not have enough time to finish it.

Although marker-based augmented reality may be easier to implement, computationally cheaper, and yields better result, the inconvenience it brings still make us think about a way to get rid of it. There has been previous works about non-marker based AR, we are looking forward to learn from them.

References

[1] ocvar, `https://github.com/htw-inka/ocv_ar`.

[2] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.

[3] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, pages 63–68. IEEE, 1998.