Title
3D Interactive Multi-User Drawing Application on Android Mobile Devices

Permalink
https://escholarship.org/uc/item/9sb3b786

Author
Lue, James

Publication Date
2013

Supplemental Material
https://escholarship.org/uc/item/9sb3b786#supplemental

Peer reviewed|Thesis/dissertation
3D Interactive Multi-User Drawing Application on Android Mobile Devices

A Thesis submitted in partial satisfaction of the requirements for the degree Master of Science

in

Computer Science

by

James Lue

Committee in charge:

Jürgen Schulze, Chair
Falko Kuester
David Kriegman

2013
Copyright
James Lue, 2013
All rights reserved
The Thesis of James Lue is approved and it is acceptable in quality and form for publication on microfilm and electronically:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Chair

University of California, San Diego

2013
# Table of Contents

Signature Page ........................................ iii

Table of Contents ...................................... iv

List of Figures ......................................... vi

Abstract ................................................ viii

1 Introduction and Related Work ...................... 1
   1.1 Introduction ....................................... 1
   1.2 Related Work ..................................... 1
   1.3 Contribution ..................................... 3
       1.3.1 Possible usage and extension of sketch application .. 3
   1.4 System Overview .................................. 4

2 3D Scene Rendering, Drawing Tool and User Operations .......... 6
   2.1 3D scene rendering: the scene graph ................ 6
       2.1.1 The Auxiliary Group for User Operations on 3D Objects .... 7
   2.2 Drawing Tools and Users’ Operations ............... 7
       2.2.1 Synchronize Rendering of Geometry Modification Between Users .. 10
   2.3 Limitation of the Scene Graph Structure ............ 10

3 Synchronize User’s Operation: The Networking ............. 12
   3.1 Networking Model ................................ 12
   3.2 Data Exchange .................................... 12

4 Marker Detection and Motion Tracking .................. 15
   4.1 Basic Idea ....................................... 15
### 4.2 Marker Detection

- **4.2.1 The Marker**
- **4.2.2 Marker Recognition**

### 4.3 Marker Decode

### 4.4 Pros and Cons of This Type of Marker

### 5 Enlarge Workspace by Multiple Markers

- **5.1 Multiple Markers AR**
- **5.2 Multiple Markers in OpenCV Marker Detector**
- **5.3 Extend to General Positions**
  - **5.3.1 Triangulation**
  - **5.3.2 Image Information Collecting for Marker Reconstruction**
  - **5.3.3 Search for Good Reconstruction**
  - **5.3.4 Reduce Discontinuity between Markers**
- **5.4 Further Extending the Workspace When No Marker is in Sight**

### 6 Results and Accuracy Testing

- **6.1 Number of Images to be used for Reconstruction**
- **6.2 Tracking Accuracy and Stability of Markers from Different Ages**
  - **6.2.1 Setting for measurement**
  - **6.2.2 Error Result**
  - **6.2.3 Sources of Errors of the Measurement**
- **6.3 Results**
  - **6.3.1 Screen shots**
  - **6.3.2 Image Processing Time and Frame rate**
  - **6.3.3 Camera position consistency problems**
  - **6.3.4 A Case Study for Scene Design**

### 7 Conclusion and future work

- **7.1 Conclusion**
- **7.2 Future work**

**Bibliography**
List of Figures

1.1 Overview of our system architecture ........................................ 5
2.1 The scene graph ................................................................. 8
2.2 The scene graph under user operating ..................................... 9
3.1 The data exchange process ................................................... 14
4.1 The marker and its components: the top left black square is the direction indicator, the 9 white squares with numbers are for marker encoding to distinguish different markers ................................................................. 16
4.2 Coordinate system of the marker ............................................ 17
6.1 The setting for reconstruction accuracy test ............................... 26
6.2 Reconstruction error for different number of images ..................... 27
6.3 Reconstruction error for different number of images, cont’d ............ 28
6.4 Reconstruction error for different number of images, cont’d ............ 29
6.5 The setting for tracking accuracy test ...................................... 30
6.6 The setup when measuring, the sponge box is used to hold the phone steadily, the marker’s center and the camera’s center are approximately at the same height (Y=0) ................................................................. 30
6.7 The setup of tracker which is used to find the ground truth position of the phone ................................................................. 31
6.8 The X and Z directions of the measurement positions .................. 32
6.9 AVG. and STD of absolute value of tracking error for marker with origin for resolution of 640×480 ................................................................. 33
6.10 AVG. and STD of absolute value of tracking error for marker with origin for resolution of 320×240 ................................................................. 34
6.11 Box for tracking, markers are placed in 3D position ..................... 36
6.12 Create a box in the space and both phones can view it .................. 36
6.13 Drawing an "OK" in space ................................................. 37
6.14 The screen shot of the "OK" .................................................. 37
6.15 A box after transformation .................................................... 38
6.16 A screen shot from the phone. It shows the house we made from the modeling
    functions ........................................................................ 39
6.17 Another viewing direction for the house from the phone ......................... 39
6.18 Showing the house we made in PC version OSG .................................. 40
6.19 Another viewing direction for the house we made. This figure shows that some trans-
    formation of the geometries are not very accurate ............................... 40
6.20 The model we reproduced by 3D Studio Max ................................... 41
6.21 Another viewing direction for the model we made by 3D Studio Max .......... 41
The main purpose of collaborative virtual reality is to allow users to share information and interact with one another. Notably, most collaborative virtual reality systems rely on specially designed equipment, e.g., a head tracking system, which is often not available to average users. This work attempts to construct a prototype for collaborative virtual reality based on Android mobile devices, a more accessible and widespread technology available within the general public domain.

We demonstrate information sharing and interaction through a 3D drawing application which allows users to create basic geometries and drawing lines in a 3D space. Basic geometric modification functions are also provided. To achieve pose estimation in 3D space, fiduciary marker-based camera pose estimation technique is used. Pose estimation of this kind allows the processing speed to meet the real time rendering requirements for the 3D scene. The markers can be positioned at any point as long as the position of the markers can be estimated by our marker reconstruction program. The information exchange between users is undertaken through wireless networking, allowing the users’ interactions to be shown in the same 3D space.
Chapter 1  Introduction and Related Work

1.1  Introduction

One common application in virtual reality (VR) or in virtual environment (VE) is 3D data visualization [17]. When displaying data to the audience, the audience may want to manipulate the data or objects in the virtual world to obtain more specific information [3]. Multi-user VR allows users to navigate the virtual world individually and view specific data [44]. Some existing multi-user VR systems or applications have been developed for the PC, where the input devices are simply a keyboard and mouse [20][8]. These kinds of VR are less likely to allow a user to move about the real world since the world is confined to the monitor. Some VR systems, such as EON reality [4], provide immersive display systems in which users are surrounded by display devices, rather than just one monitor in front of the user. This better simulates the real world. However, the VR systems mentioned above require customized and expensive display and input devices to operate, and they are unavailable to the average person in a typical home setting. Mobile devices are another alternative with which users are able to navigate through the virtual world by moving the devices as if they were the user’s eyes. They provide opportunities for users to view the virtual world as they are in the real world, without complicated display systems. Combining the ideas above, we aim to build an application which realizes a prototype of multi-user VR based on mobile devices in which multiple, simultaneous users can view data or models and manipulate these models in the virtual world.

1.2  Related Work

Mobile device virtual reality system has been developing in different ways in order to solve the basic requirement for a VR system, which is the user positioning. A user’s position in 3D space determines the 3D rendering of the virtual world. Traditional virtual reality relies on tracking devices
or simply computing the user’s position by input devices (mouse, joystick, or other designated input devices, etc) [9]. We hope the positioning for the mobile devices can be done without such equipment so that users are able to have an immersive VR application in their home.

In following, we investigate several positioning approaches in single mobile VR applications. Some single user mobile VR applications can estimate the user’s viewing angle by using sensor fusion methods [36] [43]. The orientation and relative rotation with respect to current user pose can be estimated with the help of fused data from a gyroscope, accelerometer and magnetometer (compass) [30][36]. Sensor fusion works well for phone’s rotation and orientation; however, it is difficult to estimate accurate translation from the accelerometer since the noise makes the results of double-integral for displacement estimation from acceleration unacceptable [36]. Besides, the accuracy of the accelerometer at high velocity is poor, as discussed in [37]. Since the user’s translation estimation is required in immersive VR applications, other approaches have been investigated. Some commercial products use global positioning systems (GPS) in the mobile device to calculate the displacement [5], but the precision of GPS is at the level of meters [1], making it impossible for users to run the application in their home.

To estimate translation of mobile phone, one alternative is to use structure from motion (SfM) techniques from computer vision, which estimate both orientation and translation. One common application that utilizes SfM is augmented reality (AR), which uses estimated camera pose to determine where to render the augmented objects in real physical scene. One common way to estimate camera pose in AR is marker-based AR. Marker based AR is suitable for collaborative AR applications [16] since a global coordinate system can be built for all users. A sample game shown in [28] demonstrates the global coordinate system idea for collaborative AR. We apply the global coordinate system building technique to our collaborative virtual reality application on mobile phone. Once the global coordinate system is built, we only need to find out how to make each mobile device communicate to each other so that users can collaborate in the virtual environment. Similar collaborative AR ideas are used in multi-user AR games such as "Invisible Train" [40] and "Rock ’Em Sock ’Em Robots"[7]. The above collaborative AR applications build their global coordinate system with special designed marker patterns and marker position. We hope users can place the markers at the positions they want rather than placing the markers on designed positions. This allows users to put the markers in 3D general position according to the physical workspace, and it is not necessary to put all markers on the same plane.
1.3 Contribution

We built a 3D drawing application as an example of a multi-user collaborative and interactive mobile VR system that users can run in their home. The user position tracking is done through a combination of tracking multi-markers in general position with the help of sensors. The marker tracking is done by using OpenCV Marker Detector [12]. Users can place the markers at a desired location using our marker position estimation application, and are not required to set the markers' positions by hand. Furthermore, only one user needs to compute the markers' positions since the data of marker position can be sent to other users. The communication is done over wireless network, and no additional server is used. Specifically, our system uses OpenSceneGraph for Android to render the virtual scene.

1.3.1 Possible usage and extension of sketch application

The following is a discussion of some possibilities for our system and 3D drawing application. Our system can be used as both a model viewer and a 3D data display system in which users can view the data of interest. Our system also supports basic modeling functions, such as drawing lines, simple geometry creation and basic transformation functions; this could be extended to basic modeling tools or 3D version of Draw Something [2]. In addition, the system supports showing other users positions; it can be used to develop a game which needs to render other players.

User designed games are able to utilize this type of 3D drawing application. Currently, players' design is an important component in some games. In some cases, players are not just playing that which has been designed for them but designing what they play. Famously, Minecraft [6] makes good use of this idea and allows players to design and build structures using basic materials. Some real-time strategy games allow players to edit maps and scenarios within the game. Our application allows players to create their own items or even characters for the games, as well as to share these modifications with other players. However, freehand sketching may not be suitable for items or model design due to the inaccuracy caused by hand shake. To get better positioning for points on an image, we could use 3D lattices as units in 3D space for our sketch application and, we can limit the drawn points to be located at the center of lattice to reduce the effects of freehand shaking. Without the freehand shaking, the geometry can be better controlled. In addition to player-designed games, the sketch application can also be used for games that apply some basic physics. Some 2D architectural structure design games are developed, such as sketching a
2D bridge that can support the loading on it. With our application, this game could be extended to a 3D version. Other possible usage of our application is that players can sketch a tank where its strength depends on the design. Players can use their tank to fight with other players’ tanks. Players can draw a car for racing games in which the car’s shape can affect the speed or stability of the car. Other possibilities for this application are pure sketch and physics based games. For example, one could sketch a track of roller coaster and test whether the roller coaster can operate safely. The 3D drawing application could also be extended for educational purposes. Generally, the technology can be used to visualize physical phenomena. Math is another area in which the 3D drawing application can be deployed, especially with regard to 3D geometry. Teachers are able to move beyond a 2D black board and draw in a virtual 3D space. Another possible extension of our application is game scene design and editing [10]. The difference between this tool and traditional scene editing tools is that the user can move 3D objects in a physical 3D space. Some commercial products have already implemented this technique [11]. However, such products require design peripherals, and the price is relatively high compared to smart phones. It would be advantageous to use smart phones to achieve similar results without using specialized technology as unnecessary costs can be avoided. To manipulate 3D objects in space correctly, the precision of the 3D position of the cursor is an important factor. As with most computer vision based tracking applications, the 3D position estimated is noisy in our application. As such, our application has room to improve in the future.

1.4 System Overview

The system has three global objects in C++ native code which handle networking, 3D rendering and camera frame processing, respectively. The Java component is chiefly involved in the user interface. The structure is shown in Figure 1.1.
Figure 1.1: Overview of our system architecture
Chapter 2 3D Scene Rendering, Drawing Tool and User Operations

This chapter describes the 3D rendering and functions with regard to user interaction. First, we introduce the 3D graphics engine and explain the structure of the virtual world. Then, the functions for 3D drawing and geometry modification are introduced. Last, we discuss limitations of the structure.

2.1 3D scene rendering: the scene graph

The virtual world is rendered by OpenSceneGraph (OSG), which requires the virtual world to be constructed in a scene graph structure [42]. Different types of nodes in the scene graph play different roles. Here, we briefly introduce the node classes in OSG.

- `osg::Group`: A node which can maintain a list of children nodes
- `osg::Geode`: A leaf node which does not have other nodes as children but contains 3D geometries.
- `osg::MatrixTransform`: A subclass of `osg::Group`, which contains a transformation matrix that affects the transformation of its children nodes.

The scene graph in our application is composed of three main groups (`osg::Group`) and each group has its own functionalities and responsibilities. The first group is used to display the information of other users, i.e., other users current positions and orientations. It can show other users in the scene if necessary, similar to online games where you can see the other players. The second group relates to the main scene of the world; the members of this group are the 3D objects that are displayed, e.g., the lines drawn by users and other 3D objects. The third group is an auxiliary group which aids in object manipulation by the users. Users can use the touch screen to select the object they wish to modify and then select the operations that are applied to the object. The selection is performed by casting a line from the selected screen coordinates into the scene and determining the objects that are intersected with this line. Then, it chooses the 3D object which is closest to the
image plane. We extend the Polytope Intersector class in OSG to perform the line intersection test and object picking. The entire scene graph structure is given in Figure 2.1.

2.1.1 The Auxiliary Group for User Operations on 3D Objects

The auxiliary group is formed by one osg::MatrixTransform node $M$ and its child osg::Geode $G$. Normally $G$ only contains an empty geometry $N$ which does not have vertices array, so nothing shows up for this geometry. $G$ only has geometry that is viewable when the user is manipulating one of the 3D objects in the scene. When a user is manipulating one 3D object $O$, we switch the parent of $N$, which is $G$, and the parent of $O$, say $P$. Therefore, we can just allow the object modification operations to change the transformation matrix in $M$, and the modifications will be applied on $O$. After completing the modifications to the object, the final transformation matrix of $M$ is applied to all vertices of $O$, and the world coordinates of the vertices of $O$ will be changed. After changing the vertices, we switch the parent of $O$ and $N$ again. There are several drawbacks to this design. The first one is that $O$ can only have one osg::Geode as a parent (shown in Figure 2.2). If $O$ is referenced by $N$ osg::Geode, there will be $N$ same geometries that are rendered; in that case, all $N$ geometries would be changed at the same time, so the user may possibly be confused about which object is under their control. Also, difficulties arise in determining which parent of $O$ is to be switched with $N$’s parent. The second drawback is such that users are unable to manipulate each vertex of $O$ using 3D modeling tools, such as 3D Studio Max [14].

2.2 Drawing Tools and Users’ Operations

Users’ operations have two main categories. The first one is data operations. Users can save and load the geometries they have created and share the geometries with other users. The data operations utilize the OSG node serializing functions. The second one is 3D objects creation and modification functions. The objects creation functions include line drawing and cube creation. The drawing function allows users to move their phone and choose the position they want as the vertices of the line. The cube creation is similar to line drawing: users move the phone to the position they want and create the cube. The initial cube size is fixed but can be changed using the modification functions. The modification functions include basic transformations: translation, rotation and scaling. To perform such modifications, users first select the object and then the transformation type and direction; the direction is fixed on XYZ axes. Users are able to change
Figure 2.1: The scene graph
Figure 2.2: The scene graph under user operating
the color of the objects. However, color change can only be applied to the entire geometry of one osg::Geode due to the constraints of OSG on Android.

2.2.1 Synchronize Rendering of Geometry Modification Between Users

The effects of object creation and modification are displayed in real time; since our system supports multiple users, the change in the scene should be shown to all users instantly. If one of the users \( U \) is manipulating one object \( O \), in order to show the transformation on the same 3D object for all users, other users need to know which object \( U \) is being changed. This requires that all users have the same scene graph. For example, when we do the parent switching which is discussed in Section 2.1.1, we need to find the parent osg::Geode of \( O \), say \( P \). However, we can only get the pointer \( p \) that points to \( P \) by using OSG’s API to find parents. It cannot be guaranteed that \( p \) stores the same memory address for all other users. Therefore, we assign an ID for each pointer to each osg::Geode, so that when telling other users which osg::Geode is the parent of \( O \), we can just use the ID to let other users know which osg::Geode is to be switched. When a user knows which 3D object to modify, the modification commands of \( O \) is sent to other users. We do this by sending the transform matrix and update the matrix at each user’s device. When beginning and finishing the object modification, we send a signal to other users’ phones so that they know when to switch parents. Synchronizing the object creation between users does not require finding the parent of the geometry but requires assignment of ID for each new object. This is easier than synchronizing the object modification since we only need to calculate a new ID and send the information about the new object, e.g., the position of the new cube or the new vertex of the line, to other users. The new ID calculation is similar to linear probing hash with the hash table size taken as the maximum of the unsigned integer. It verifies whether the new ID is used; if that is the case, we increase the ID by one and check again.

2.3 Limitation of the Scene Graph Structure

When performing line drawing or geometry transformation, the system cannot allow more than one user to participate these operations due to the scene graph structure. Currently, the scene graph has only one supporting structure for line drawing or geometry modification; therefore, it does not allow two users or more to use the same supporting structure at the same time, which causes race conditions. Since we want to show the process of geometry operations to other users while one
user is doing such operations, it is not the case that we can simply send the final transformation after the modification is finished. The other limitation is that when sending a 3D object to other users, other users should not have any existing object because we need the ID assignment for each osg::Geode to be consistent among users. If other users have their own objects, the ID assignment would be mixed up since there are chances that the same ID is assigned to different objects within the user group. Although it is easy to check whether there are IDs that are assigned and duplicated between users, it is not easy to check if the same ID represents the same 3D geometry, since defining two 3D geometries that are the same is problematic. It is not a viable option to simply check whether the vertices of geometries are similar in determining whether the geometries are the same. Even the same triangle can have a different representation as we can assign any one of the vertices in the triangle as the first vertex.
Chapter 3  Synchronize User's Operation: The Networking

The network allows users to share their actions and information with each other. This is the key point for collaboration among users. This chapter discusses the network model we use and how the information is exchanged.

3.1 Networking Model

The network architecture to exchange information between users is a traditional client-server model. As such, it is easier to implement and is suitable for small scale multiplayer online games [21]. The networking connection is done through TCP to ensure that data will not be lost. The server is also an Android smart phone. Another machine was not used as the server so that users do not need to worry about the setup of a server. However, due to the current computational loading on a phone, we only support two users for testing, making the model resemble peer-to-peer architecture.

3.2 Data Exchange

The model of data exchange between users is similar to data exchange schemes in online games [18] which exchange the states of each user. Based on the state of each user, the 3D scene rendering is done locally on each user's side. We use the same class for both the server and client to store the state. The state includes the following data:

- the position and orientation of the other user.
- flags for transferring geometry.
- flags for geometry drawing (creation of lines or cubes).
- flags for 3D geometry modification, including geometry transformation, deletion and color change.
- flags for transferring other auxiliary data (explained in details later).
The data are exchanged periodically to maintain the same scene state for both users. The Java TimerTask is used to update both users’ states every 0.1 seconds. Based on the flags, each user can determine how to parse the incoming data. For example, when transferring geometry, the data could be large. In that case, we divide the geometry data into small pieces and send the geometry by multiple state updating cycles. Geometry drawing and modification are mentioned in Section 2.2.1; if the flags are regarding drawing lines or boxes, the drawing commands are sent and executed in other user’s phone locally. When the flags are regarding 3D geometric operations, we first send the ID of the geometry then, the $4 \times 4$ transformation matrix for the geometry is sent for every following updating cycle. Other auxiliary data include the marker data (if necessary) and locks for preventing the race condition (the Java TimerTask will not check the state of data transferring, but it may access objects that are under operating) of some data, such as the buffer for sending and receiving data. The data exchange process is summarized in Figure 3.1.
Figure 3.1: The data exchange process
Chapter 4  Marker Detection and Motion Tracking

Tracking is an important part in virtual reality because the scene rendering depends on the user's position and viewing angle. In our application, the tracking is done with the help of fiducial markers. This chapter explains how tracking is accomplished and discusses the pros and cons of this kind of marker.

4.1 Basic Idea

The tracking of the phone position is done by tracking the position of the phone's camera. To compute the camera's position, we use rectangle markers whose positions in a 3D space are known (set by us) and their position in 2D images are taken from the camera. Then, we compute the camera position by the 2D-3D point correspondences [39]. This technique is commonly used in augmented reality (AR). We can use the four corners of a marker to find out the camera position.

We use the OpenCV Marker Detector [12] and slightly modified it (discussed in Section 4.3 to do the marker tracking and recognition).

4.2 Marker Detection

4.2.1 The Marker

The marker is formed by two concentric squares which contain a direction indicator and nine small squares for encoding (to distinguish different markers) in the center. The size of the marker is set by the OpenCV Marker Detector [12] in which each edge is 90mm. Figure 4.1 shows the structure of the marker. We set the units for world coordinates as one millimeter since the marker's size and camera's motion is computed in this unit of measurement. We can assume the origin of the world coordinate system in a 3D space is the center of one designated marker. The world coordinate system is shown in Figure 4.2. This type of coordinate system is the same as in
4.2.2 Marker Recognition

To detect a marker in an image, the OpenCV Marker Detector finds two concentric quadrilaterals in the image and rejects the quadrilaterals whose perimeters are too long or too short. The OpenCV function `cvFindContours` can find contours that are concentric. Then, it just checks number of corners that the contours have; if they have four corners, then they are quadrilaterals. Once the quadrilaterals are found, the pixel coordinates of the four corners can be determined. However, there are ambiguities for the four corners. Since the marker could be rotated, it cannot determine which corner is the top left one, which is bottom left one, and so on. Therefore, a directional indicator is added and used to determine which corner is top left one, and once identified, the direction of the marker can be determined.
To determine the direction of the marker, we first find the homography between the projected marker image and a 90mm × 90mm square. Then, we warp the projected marker image to a rectangle and ascertain where the direction indicator is in the warped image. Since the actual size of indicator is also known, the relative position of the indicator in the warped image can be computed. Once the direction is known, the actual four corners can be computed. With the calibrated camera, we can use the cvFindExtrinsicCameraParams2 function to calculate the camera translation and orientation.

4.3 Marker Decode

The marker is encoded by a 3×3 2D rectangle array, where each rectangle represents one bit. So, in total, there are nine bits. The encoded result is a marker’s ID. If the rectangle is painted black, the bit is on. To check if each bit is on, we first warp the marker’s image to square, and then check if the average image value in the bit is under a certain value (close to black). It should be noted that we modified the encoding method to make the encoding range of the marker larger. The original encoding method is similar to ARTag [25], which uses numerous bitwise XOR within all bits. However, the possibility exists such that different patterns of the bits will have the same encoding result as the original encoding; this reduces the number of markers that can be differentiated. In fact, not all nine bits can be used to encode. The bottom right square is not treated as one bit, in order to avoid the ambiguity of the direction indicator. Therefore we have 9-1=8 bits for the
ID encoding, which means the number of different markers is $2^8 = 256$. Therefore, we choose to encode the marker to calculate the ID as a binary number. In total, we could have up to 256 different markers.

### 4.4 Pros and Cons of This Type of Marker

Compared to the markers of ARToolKit, this type of marker is easier for encoding; one is able to use a pen to draw the eight bits to distinguish different markers and the ID can be easily computed by a human, which is helpful for debugging. ARToolKit's markers have more freedom for encoding; it can use any arbitrary 2D geometry for encoding. However, when creating new markers with new encoding tasks, the camera needs to store the pattern of the new markers; therefore, whenever users create a new marker, they need to generate a pattern description file and store it on the phone. The other benefit of our marker system is the compatibility with OpenCV. Since it uses OpenCV to detect the markers, we could use OpenCV functions to carry out other operations on the markers, for example, highlighting the marker in the camera frame using OpenCV functions.

The main disadvantages of our marker system have to do with marker recognition. There are four main recognition errors discussed in [38], which are:

- **Misrecognize non-marker objects as marker**: this could happen when there are rectangles in the scene.
- **Decoding error**: this could happen when the image of the marker is not clear and the positions of the bits are not well determined.
- **Fail to compute correct ID**: this happens when the direction indicator is not detected or detected erroneously; in this situation, the direction cannot be determined and the marker is treated as invalid.
- **Fail to recognize a marker**: the `cvFindContours` function needs the image to be binarized, so the question of how to set the threshold for binarizing remains troublesome. Specifically, the lighting conditions severely affect binarization. Although we can use the averaged pixel values as the threshold, at some conditions, such as when the markers are in a shadowed area, the pixel values for entire markers still could be lower than the threshold and cannot be detected by the contour finding process.

The first error can be reduced by not using ID's that have similar image for the pattern of bits, but the number of different markers is reduced as is the ARTag marker design [24]. However,
unlike ARTag which has 36 bits for encoding, we only have eight bits. A similar pattern elimination process will make our markers insufficient. The second and third detection errors can be avoided by not viewing the marker at a tilted angle. The last error can be mitigated by placing markers in well-lighted locations.
Chapter 5   Enlarge Workspace by Multiple Markers

Using one marker is insufficient for user position tracking in a large physical space. In order to make the tracking available in a larger physical space, we allow users to place multiple markers in places they desire and measure the markers’ positions. This chapter explains the benefits of using multiple markers to do the tracking and how the marker’s positions are calculated. It also discusses issues with multiple markers and tracking compensation when marker is lost in the camera frame.

5.1 Multiple Markers AR

The drawback of using markers is that if no marker is in the scene, the camera position cannot be calculated. Therefore, the workspace is limited to the space near the marker. In order to increase the workspace, we use multiple markers. There are several benefits of using multiple markers. ArUco [19], which is an OpenCV-based marker detection library, presents the benefits of multiple markers. The benefits are such that there is a reduction in the chance that marker detection fails increasing the robustness of the estimation process. The reasons for marker detection failure are discussed in Section 4.4. Other causes for detection failure could be motion blur and a tilted viewing angle. However, the chance that for all markers fail to be detected is relatively low since it is less likely that all markers are in bad conditions; thus, the tracking robustness is good.

5.2 Multiple Markers in OpenCV Marker Detector

The OpenCV Marker detector is based on one known marker whose center is treated as the world origin. No matter how many markers are in the camera frame, it calculates the orientation and translation for each marker individually by treating each marker’s center as the origin. This means each marker builds its own coordinate system. To get the tracking to be consistent for all markers (not treating every marker as the origin), we can choose one marker as the origin and find the world coordinates of other markers. In fact, we can measure the world positions of the markers
by using a ruler. ArUco uses a marker grid in which the markers form a 2D array. In this case, the world coordinates for other markers can be easily measured as they are on the same plane and regularly arranged. OpenCV marker detector can also achieve such 2D marker array as long as the positions of markers with different IDs are calculated. However, our goal is to place markers in general 3D positions. It is not easy to measure markers’ positions if those markers are in general 3D positions. In next section we discuss our approaches to calculate markers’ positions in 3D case.

5.3 Extend to General Positions

The limitations of the marker grid are such that the camera still needs to look at part of the grid, which means the camera cannot view the space under the surface of grid. To solve this constraint, we put markers in 3D general positions. However, measuring the position of markers is difficult as the markers are on the same plane as the grid. To allow users to easily determine the 3D position of markers, 3D triangulation techniques are used in general positions. To perform the triangulation, several images are required that contain non-reconstructed markers and reconstructed markers. The reconstructed markers are used to determine the camera position. We collect camera positions and the pixel coordinates of the non-reconstructed markers’ corners for each image to perform the marker reconstruction.

5.3.1 Triangulation

To estimate the 3D coordinates of a marker’s corners, the iterative linear triangulation method [27]. First of all, the fixed-point iteration method [41] is used to calculate the undistorted corner coordinates of markers in each image from the distortion formulas [32]. After finishing the distortion correction step, linear triangulation (see Multiple View Geometry in Computer Vision [34]) is used to find a coarse initial solution for the corner position in 3D space. Then, the iterative method [27] is applied to determine the 3D coordinates that minimize the reprojection errors. We also tried the Levenberg-Marquardt algorithm (LMA) [35], which is an iterative numerical method for solving nonlinear equations, to solve the error minimization. The 3D corner points are treated as the parameters that we want to optimize in LMA. However, the result shows that the error after convergence is close to the error of the iterative linear solution; therefore, the iterative linear solution is used for reconstruction as it is faster as compared with our LMA implementation.
5.3.2 Image Information Collecting for Marker Reconstruction

Since the camera pose is needed for marker reconstructions, if there are multiple markers in the image, we need to determine which marker is used to compute the camera pose. According to the above discussion, a reconstructed marker would have errors in their positions. If we use reconstructed markers to do camera pose estimation, the error of estimated pose could be larger since the positions of the markers are already incorrect. If we use such incorrect pose to reconstruct new markers, those new markers’ positions would have larger reconstruction errors since errors in pose estimation is also accumulated. Therefore, when we reconstruct markers, we want to use the reconstructed markers that have less accumulated errors to do pose estimation and reconstruct other markers. In order to find out which marker could have less accumulative errors, we define a number for each marker to record the number of positional reconstructions are done before this marker get reconstructed. We call this number as the "age" of the marker. If a marker is older, it means that the marker has fewer reconstructions before it get reconstructed and potentially the marker has less accumulated errors. Therefore, when computing the camera pose, the oldest marker in the scene is used. The oldest marker is the origin, which has an age of 0, the younger markers, intuitively, have larger numbers as their ages. When collecting image information, each non-reconstructed marker stores the pixel coordinates of the four corners, the camera pose and the age of the marker. After the information is sufficient (collected from a certain number of images), triangulation is applied. The age of each reconstructed marker is the average age of the markers that are used to find camera pose.

5.3.3 Search for Good Reconstruction

When undertaking marker reconstruction, we randomly select five images [26] (see Section 6.1) from the images that are collected to perform the triangulation. We randomly select images because some images could introduce high reconstruction errors and we do not want to use those images to perform the triangulation. In our experiences, if we use images that the viewing angle to the marker for camera pose estimation is too tilt for triangulation, there is chance that the triangulation errors could be very large. Since we do not know which images will cause bad triangulation in advance, random selection has chances not using such bad images. After the reconstruction process is complete, verification is required to determine whether the reconstructed markers are adequate and that errors are below a certain threshold (set by us). If a reconstruction result that is
below the threshold is found, this result is accepted and the averaged reprojection error is stored as the new threshold. The new threshold is stored because the system allows users to undertake the triangulation with random selection again and again to see if better reconstructions can be found using different images.

After the reconstruction for every marker is finished, we force the four corners of each marker to be coplanar; this reduces the instability when estimating the camera position in our experiments. To make the corners coplanar, the least square plane fitting method[22] is used to find the plane that allows the sum of distances from the corners to the plane to be minimized. We further adjust the corners to restore the marker’s properties (right angle for each corner and length of 90mm for each edge). We first make the two diagonals in the reconstructed marker perpendicular by rotating the corner around the intersection of the diagonals, say \( C \). The rotation axis is the norm of the fitting plane. After the diagonals are perpendicular, we adjust the length between \( C \) and each corner to be \( 45\sqrt{2}\text{mm} \); this further stabilizes the pose estimation, but may introduce more errors which will be discussed in Section 6.2.2.

5.3.4 Reduce Discontinuity between Markers

When the tracking is switched to another marker, a visual discontinuity in 3D rendering is triggered as the position calculation could not be the same (since the errors in marker reconstruction and position estimation) for different markers in the same camera frame. We do not reduce the pose estimation error by estimating pose from multiple markers as ArUco or ARToolKit as the reconstructed marker positions have errors. Here, there is no guarantee that other reconstructed markers can make the estimation error lower. Therefore, when there is visual discontinuity from tracking marker switching, the pose is computed by interpolating the pose estimation from the previous marker and a new tracking marker. The rotation interpolation is done by spherical linear quaternion interpolation [23]. The interpolation for translation is a simple linear interpolation. The ratio for the interpolation is determined by the distance from the camera to each marker. Suppose the camera detected two markers: \( A \) and \( B \). The camera pose is computed from \( A \) and \( B \), giving us \( q_a, t_a \); \( t_a \) as the camera pose from \( A \) and \( q_b, t_b \) from \( B \), where \( q \) denotes the quaternion that represents the camera’s rotation and \( t \) represents the camera translation. Then, the distances between the camera and each marker is computed. This is done by estimating the camera translation while treating the center of each marker as the origin. Suppose we calculated the distances as \( d_a \) and \( d_b \). The final rotation is computed by the quaternion \( q = r \times q_a + (1 - r) \times q_b \) and the final
translation is: \( t = r \times t_a + (1 - r) \times t_b \), where \( r = \frac{d_2}{d_1 + d_2} \). In this way, the marker that is closer to the marker that got a greater weight in the interpolation.

5.4 Further Extending the Workspace When No Marker is in Sight

As sensor fusion was discussed in [36], the data from the gyroscope in the short term are trustworthy. Therefore, when no marker is in the image frame or the marker tracking is lost, we switch to use a gyroscope to calculate the rotation of the phone. Since the gyroscope calculates the rotation and as the rotation center is the center of the camera, it does not rely on an arbitrary coordinate system as do magnetometers. Thus, we can simply multiply the rotation matrix to the camera matrix for OSG with the axes corrections. However, due to workload constraints, the data updating frequency flag in the Android system is set to SENSOR\_DELAY\_NORMAL, which is about 10 to 20 Hz. The accuracy under this frequency could be poor, but a higher frequency often causes program crash in our experiments.
Chapter 6  Results and Accuracy Testing

This chapter shows the experimental results for the accuracy of marker position calculation and the relation between tracking stability and tracking distance, as well as the results from drawing functions and collaborations among users. We also provide a model example created by the application for the purpose of discussion.

6.1  Number of Images to be used for Reconstruction

To find out how many images are sufficient to reconstruct the marker locations, we reconstruct markers with ages ranging from one to three with different numbers of images. We put markers on a box, as shown in Figure 6.1. We measure the actual position of the markers by ruler and compute the reconstruction error as the distance between actual position and reconstructed position. The possibility for error regarding the measured position exists since we assume the corners of the box are perfect right angles; however, our purpose is to find out how many images are needed.

Figure 6.2, 6.3 and 6.4 show the relation between the number of images used and the reconstruction errors. For each marker, we take 10 images and randomly choose two to eight images to perform the reconstruction. We run the reconstruction 10 times and determine the range of the reconstruction error for each age. The result shows that the error becomes stable when using more than five images. Therefore, we use five images to perform the marker reconstruction.

6.2  Tracking Accuracy and Stability of Markers from Different Ages

This section we want to find out at what distance between the phone and markers the tracking, or pose estimation, of the phone can be effective. Effective tracking should be accurate (estimated pose should be close to actual pose) and stable (the pose estimation error should not
fluctuate too much for each pose), these are the two factors that we want to measure. The following we describe the measurement setting and discuss the results.

6.2.1 Setting for measurement

To measure the tracking accuracy and stability, we use the DTrack tracking system to track the ground truth position of the phone and compare the phone's position to our computations using the markers. We mainly measure the translation error for the phone at different positions relative to the markers. The tracking system and the markers have different coordinate systems. Fortunately, the two coordinate systems are both right handed coordinate systems, and we can rotate and shift one coordinate system to make the two coordinate systems consistent. The markers are placed to be aligned on the X axis, as shown in Figure 6.5; the distance between each marker is about 30cm. We take measurements at 4 different ages. We tested 2 different resolutions which are 640×480 and 320×240. For each age, we measure the accuracy at 28 different positions. We stop the measurements when the errors are too high to be worth measuring (stop after a significant error increase is observed), the marker is outside of camera's field of view and when we reach the physical space restriction (device setup and the working area of the tracking system). Pentenrieder et al. [33] summarized several studies about marker accuracy measurements; the
Figure 6.2: Reconstruction error for different number of images
Figure 6.3: Reconstruction error for different number of images, cont'd
The number of images they used was at least 250. We chose to take 250 images for each measurement position to reduce the time for measurement and battery charging. When measuring, we positioned the camera facing the marker as no relative rotation to the markers as possible. This is because we do not want the relative rotation between the markers and the camera affects the measurements. The effects of the relative rotation is worthy to test, however, we do not have the devices to control the rotation degree of the camera currently. Maintaining the camera on the plane $Y=0$ (there might be small offset in $Y$ direction, but in our observation, the offset is less than 10mm), was chosen to reduce the effects of the viewing angle changed along the $Y$ axis. The setting of measurements is shown in Figure 6.6. The marker's ground truth position is measured by putting the tracker at approximately the same position as the camera (see Figure 6.7).

### 6.2.2 Error Result

**Explanation of Terms**

We calculate the error as the distance between the ground truth position and estimated position. We record the averaged error and standard deviations of the error. The standard deviation of the errors shows how large the error scatters. If the scatter is large, it means that the estimated
Figure 6.5: The setting for tracking accuracy test

Figure 6.6: The setup when measuring, the sponge box is used to hold the phone steadily, the marker’s center and the camera’s center are approximately at the same height (Y=0)
position scatters and that the tracking is not stable. Since the difference between markers with different ages is the marker reconstruction error, which is a constant displacement error. Thus, standard deviation of the error is not changed with age, the age only affects the average of error. Therefore, we only show the error results for marker with age 0 since the errors for other ages are similar to age 0 except a constant shifts caused by marker reconstructions. Figures 6.9 and 6.10 shows the errors we measured. Figure 6.8 visualizes the terms in Figures 6.9 and 6.10. The origin in Figures 6.9 and 6.10 is the center of the measured marker. The red X and the green Z in Figure 6.8 are mapped to "dist. to marker’s center in X" and "dist. to marker’s center in Z" in Figures 6.9 and 6.10 respectively. The positive Z (the green Z) is the normal of the plane that the marker lies on.

**Error Result Discussion**

From the results, we can see that, for both resolutions, the standard deviation of error for Z distances less than 800mm is approximately stable (most values are less than 100mm), but the standard deviation of error for lower resolution is slightly higher when the Z distance is higher than 600mm. This trend of error distribution is similar to [13] but the value of our data is slightly higher.
Those positional shifts from the reconstruction errors for different ages introduce the discontinuity for pose estimation between markers. As such, we use interpolation between markers to reduce the discontinuity. However, plane fitting reduces the instabilities of pose estimation which is reflected in the standard deviation of absolute errors.

6.2.3 Sources of Errors of the Measurement

Sources of the errors in our measurement come from internal and external limitations regarding the camera. The internal errors pertaining to the camera come from the: camera calibration, image processing and numerical error of pose estimation. Specifically, these internal errors are what we want to measure. However, the measurement results contain external errors which mainly come from the measurement devices. The sources of external errors are summarized as follows:

- the marker reconstruction error: the positions of the markers are estimated rather than measured; these are constant errors between ground truth and estimated position as discussed above.
- the error in tracking system: the coordinate system of the tracking system is not matched
Figure 6.9: AVG. and STD of absolute value of tracking error for marker with origin for resolution of $640 \times 480$
Figure 6.10: AVG. and STD of absolute value of tracking error for marker with origin for resolution of $320 \times 240$
perfectly with the coordinate system of the markers. This error can be measured by DTrack system. We use the tracker moving along the marker's axis. Moving the tracker along marker's X axis for about 800mm, the tracker reads less than a 6mm change in the Y and Z directions.

- the error of devices: the devices for setting the position of the phone could also introduce some errors. We measured this error by ruler, which is less than 10mm in each direction.
- the marker: since the marker can be deformed a little (not a perfect plane), this introduces errors for corner detection. Although this error is not able to be measured, we take 10mm as it maximum in each direction.

Only the first item of the external error source is unable to control the measurement setting. The rest of the external sources should affect less than 30mm.

6.3 Results

6.3.1 Screen shots

The functionalities are explained in Chapter 2 and 3, here we show the screen shots from real testing. We use a box with multiple markers attached on each side for camera position tracking (Figure 6.11). Figure 6.12 shows that we have created a virtual and both phones render it. Figure 6.13 and figure 6.14 shows an "OK" we wrote by the line drawing function in 3D space. Figure 6.15 shows the geometry transformation function, the box after transformation became a cuboid and rotates to a certain angle.

6.3.2 Image Processing Time and Frame rate

The image resolution dominates the processing time. In our experiments, for the Samsung Galaxy SII with resolution of 640×480, it takes about 120 to 200 ms to finish the processing (depends on how many markers are processed) for each camera frame (running with 3D rendering and network processing), the resolution of 320×240 takes about 60 to 100 ms. The 3D rendering takes about 15 to 30ms and is independent of the camera resolution.

6.3.3 Camera position consistency problems

The tracking result depends on the calibration of each camera. Unless we have exactly the same camera for each phone, the tracking result cannot be the same. Therefore, the virtual object can only be mapped to the same approximate physical position for each phone.
Figure 6.11: Box for tracking, markers are placed in 3D position

Figure 6.12: Create a box in the space and both phones can view it
Figure 6.13: Drawing an "OK" in space

Figure 6.14: The screen shot of the "OK"
Here, we tried to use our application to build a simple scene with the functions we provide. The purpose is to find out if our application can allow users to design the scene for the game. The scene includes a house and a tree. The house is constructed using transformed cubes and a heart-shaped line. The tree is composed of a transformed box and a spiral green line as the leaves. The screen shots of the scene in the phone are as Figures 6.16 and 6.17. The house model can be loaded from the phone to PC and shown by PC version OSG. Figures 6.18 and 6.19 show the house model which is displayed in PC. We tried to reproduce a similar scene by 3Ds Max (shown in Figures 6.20 and 6.21).

Case Study discussion

We discuss the pros and cons of the modeling function:

Pros:

- If the functions of the game are supported, the game could be started as the model is completed; this concept is common for user design based games. Possible products for the 3D
Figure 6.16: A screen shot from the phone. It shows the house we made from the modeling functions.

Figure 6.17: Another viewing direction for the house from the phone.
Figure 6.18: Showing the house we made in PC version OSG

Figure 6.19: Another viewing direction for the house we made. This figure shows that some transformation of the geometries are not very accurate
Figure 6.20: The model we reproduced by 3D Studio Max

Figure 6.21: Another viewing direction for the model we made by 3D Studio Max
design game can be based on structure design. For example, users will be able to create a building together and see if their building can withstand an earthquake once they finish the design.

- Since free hand drawing in 3D space is supported, the 3D spiral line (the leaves) can be drawn in an intuitive way. This process does not require that the line is drawn on a plane then modified.

Cons:

- Apparently, the precision of transformation cannot be controlled well as a modeling tool as in Figure 6.19, it can only make the transformation roughly because the amount for transformation is set by the movement amount on the touch screen. We do not support users insofar as inputting the transformation amount currently.

- The tracking precision is still a problem, which dramatically affects the line drawing since the point position will be wrong. The problem is not only the pose estimation noise; sometimes, there is marker misdetection. Marker misdetection introduces huge errors in the estimate.

- The total time for modeling by our application is much longer than in using 3Ds Max since the functions for modeling and basic geometry creation are not supported well. For example, geometry copy and paste are supported. This really slows down the speed for creating objects that have symmetrical structures. Also, the box creation must start from a cube and its dimensions must be modified later. Notably, 3Ds Max can determine the dimension of the box while the user is creating it.
Chapter 7  Conclusion and future work

7.1  Conclusion

Our work created a prototype of a multi-user collaborative virtual reality application on Android mobile devices. In this application, users can show the lines they draw in 3D space or 3D geometries to other users. They can also perform some transformations on 3D objects. This system only needs users to build the workspace with markers; no other tools are needed. The markers do not need to be aligned on checker-boards; users can place the markers in any position, as long as they can be reconstructed by older markers. When tested on the Samsung Galaxy SII with a camera preview resolution of $320 \times 240$, it runs in real-time (camera frame processing and rendering need about 70ms to 150ms). With a resolution of $640 \times 480$, the longest processing time is about 230ms. It should be noted that the processing time does vary slightly. The largest Z distance for stable tracking is about 800mm, with a resolution of $640 \times 480$ and $320 \times 240$ and marker age is, at most, 3.

7.2  Future work

Here we list some points as future work. The first one is about the tracking side. The stable tracking range is, as discussed above, about 800mm from the marker. Improving the tracking range and the tracking stability helps our work to be more useful in different applications such as games or educational tools.

The next is that we can try to extend tracking to markerless tracking. If markerless tracking can be done, users can skip the marker calibration step and run the application directly. The PTAM on iPhone 3G [29] shows that markerless augmented reality is possible on smart phones. Notably, we would like to try to get rid of markers. However, some issues remain regarding markerless systems which need to be considered. The first of these issues is regarding how to make the
coordinate systems of each user consistent. PTAM builds the coordinate system by assuming that the first two key frames are taken by moving the camera approximately along the X direction of the image coordinate system and the actual distance between the two key frames is assumed. As such, it is hard to guarantee that other users can build the same coordinate system since it is impossible for each user to have the same key frames. Using some special natural feature could build the same coordinate system for both users [31] but we need to consider how large the workspace could be for users. Making a coordinate system consistent for multi-users is a key point in the development of this type of technology.

The third one is that more measurements can be done. Our measurements only measure a pure translation of the marker; more factors should be tested to get more accurate workspace volume estimation. These factors include: viewing angles, the actual position of reconstructed markers (not just put markers aligned on the X axis) and other camera positions (not moving camera only on the XZ plane).

The last one is about the networking side. It is possible that more users can join in the drawing application. However, which kind of networking architecture (peer to peer connection or centralized server) is suitable is worth considering and needs to be tried out.
Bibliography


