Assignment-2 Report CS7034: Augmented Reality

Design and implement an augmented reality game that uses vision and one other type of interaction mechanism (e.g. wireless mouse, Wii Remote, mobile phone, physical switch, etc). Can be implemented on any platform but must include live video and live graphics

Submitted to:

Peter Redmond (predmond@tcd.ie), Mr.
School of Computer Science and Statistics,
University of Dublin,
Trinity College

Prepared by:

Pankaj Gupta (guptapg@tcd.ie), 11263065
MSc. Interactive Entertainment Technology,
University of Dublin,
Trinity College

13th April 2012

Abstract: This report covers my assignment to design and implement an augmented reality game that uses Computer Vision along with some mechanism for interaction as part of game-play. Assignment specifications allow exploring any hardware or technology to be used as long as it allows having 3D interaction in augmented scene.

In this report I present a 3D game involving human body, its pose and gesture recognition, a basic skeleton representation of human body to assist pose and gesture recognition and to represent it as a player while interacting with the augmented 3D scene.

1.0 Introduction:

The game is a single player 3D game where in the player have to first pose and follow certain body gesture in a sequence to obtain a weapon (a sword in this case) in his/her hand and then use this weapon to burst/break the objects (a beach ball in this case) flying past in the augmented scene (in right to left direction with respect to actual player) to earn rewards. Un-burst objects crossing off the view will again be spawned at their starting position and will again fly right to left direction holding lesser reward. Objects follow random directions in the 3D space and hence player would have to move in 3D space to burst the object after which the object would disappear from the scene.

Live video stream and scene depth information is captured using Microsoft Kinect RGB* camera and infrared sensors respectively to detect a human body (if any), its basic skeleton information and the joint orientations. This information is obtained using Microsoft Kinect for Windows SDK**. "It provides reliable 3D joint orientation and location map which fully describes a human's pose in Cartesian space"[7]. According to a recent paper published by Microsoft [8], it uses something called as decision forest (collection of decision tree) trained using thousands of sample datasets. This result in advantage like not requiring calibration poses prior to use.

To augment the real world scene captured above, I have used Microsoft XNA4.0 framework to model the augmentation in the scene. Also being used is BeTheController library [5] for pose and gesture recognition/matching (which is not a very mature library as yet).

*RGB: RGB stands for Red, Green and Blue channels of a quantized image representation
**SDK: Stands for Software Development Kit. In our context, it's a software framework provided by Microsoft in early Feb. 2012

2.0 Motivation

Augmented reality is bustling with new research backed up by newer hardware and with new hardware, new dimension of applications and research forks. Considering the generous specifications for this assignment where we are encouraged to explore whatever we can, and after substantial literature review, I decided to devote my efforts in Microsoft Kinect. Reason being it allows having marker-less solution, no overhead of calibration before use and that because of lesser restricted environment it has opened up new possibilities of future applications leading to attract substantial research efforts in it. Referring to [8], it is established that Kinect is capable of real-time 3D tracking of an entity with the degree of robustness that was never achieved before.

These facts make me believe that Microsoft Kinect is going to help us find solutions of the problems of the date. Moreover, it could replace several of our existing relatively complex solutions.

3.0 Literature Survey

Though relatively new, a lot of active research is going on in technologies for augmented reality. With addition of new hardware almost every year into our toolset, research is volatile to some extent (ie. It is changing).

All techniques could be broadly divided into two categories – Marker based and Marker-less technique.

Marker based techniques rely on a marker which acts as a unique code for the application to make some sense out of it. They provide a good solution but at the same time they are restrictive as well. The biggest issue is markers themselves as they are not very intuitive to use in all conditions. The number of markers is restricted depending upon the size of the

marker matrix chosen (there are ways to increase the number, but there is always some limit). Moreover it might not work as expected in darker lighting. There are several marker based frameworks I assessed as part of my survey – ARDeskTop, ARToolKit, ArUco, GoblinXNA, Glyph Recognition And Tracking Framework (GRATF) and Parallel Tracking And Multiple Mapping (PTAMM) are the ones that I managed to setup and run successfully. Among these I found GRATF, PTAMM, and GoblinXNA to be marginally good implementations and having good documentation available.

Marker-less technique does not rely on any kind of markers to be present in the scene and hence are less restrictive on their environment. Marker-less solutions need to know about the depth information in some way, it could be through stereo-vision or any other techniques like infra-red sensing. The frameworks I assessed are - Vuzix_Wrap_920AR stereo-vision based solution and Kinect for Windows framework. Vuzix_Wrap_920AR is a wearable head-mounted stereo-camera which needs calibration before use and is not easy to use and carry. Kinect for Windows is equipped with a good quality camera, infrared depth sensor and array of audio sensors. Out of these, depth sensor is the most interesting as it allows detecting position and orientation of object in front of the sensor and is also used to track the skeleton of human game players. two the latter one is much promising as it provides more features and comparatively accurate results, and easy design to work with.

4.0 Implementation

The Kinect sensor projects a grid of infra-red dots onto the scene in front of it. By working out the displacement of each dot as it is reflected the sensor can work out the distance to that point in the scene. This is also used to isolate a human body shape from a scene and directly interpret human movement and gesture.

The `KinectSensor` class of Kinect SDK provides a link between the application and the Kinect device. The camera, depth camera and skeleton tracking behaviors could be enabled using this class which can generate events when they have new data. In my case, I have enabled all of these behaviors and provided a target function to be called whenever there is new data available.

```
myKinect.AllFramesReady += new
EventHandler<AllFramesReadyEventArgs>(myKinect_AllFramesReady);
```

Whenever color data (ie. Camera data) is available; it is converted to array of color values which is used to define a texture to be displayed as live video.

The Kinect depth sensor uses an infra-red projector and infra-red camera. The projector draws an array of dots over the scene in front of the sensor. The camera views this scene. Processing elements in the sensor bar then work out the distance from the sensor to each dot based on the displacement of the reflection. The effective range of the sensor is 80cm to around 4 meters. This depth data is also used to track people in front of the sensors. It can track maximum six people in total – two of them in detail (by their skeleton) and rest four by position only. A Kinect skeleton is made up of 19 bones and 20 joints. Each joint is positioned in 3D space relative to the Kinect sensor. At any time, we only store one skeleton in the scene (first fully scanned skeleton). We are using the joint locations to draw small sphere [10] to represent the each joint – together they form visual representation of player's skeleton as shown in Figure 5.

For pose and gesture recognition I am using BeTheController [5] library which acts as a broker between our program and Kinect sensor data. The library is part of the whole package

called BeTheController which also contains an excellent tool which allows creating a list of poses and gestures desired to be recognized in our application. The tool calculates all the relations relative to the player's body so the position of the user in the environment is not important for the recognition of a pose as far as the Kinect controller can track the user's skeleton.

Figure 1: Pose for left or right hand are independent of each other. The pose shown for left or right hand is also the starting of our gesture which ends as shown in Figure 2.

Figure 2: End of right hand gesture that starts with the right hand pose shown in Figure 1. Left and right hands have their own (but similar) gestures recorded so they function independently (refer demonstration video)

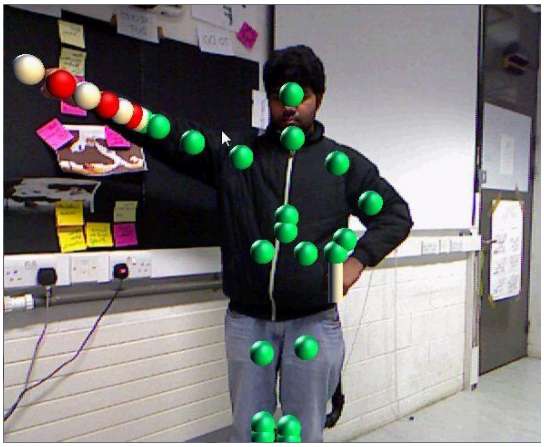

Figure 3: End of left hand gesture that starts with the left hand pose shown in Figure 1. We display the red and white balls at the end of recognized left hand gesture. The red and white balls are just for display and not for collision with any other objects in the scene.

Figure 4: Left and right hand gestures can be performed simultaneously.

Figure 5: Simple skeleton of a detected human in the scene.

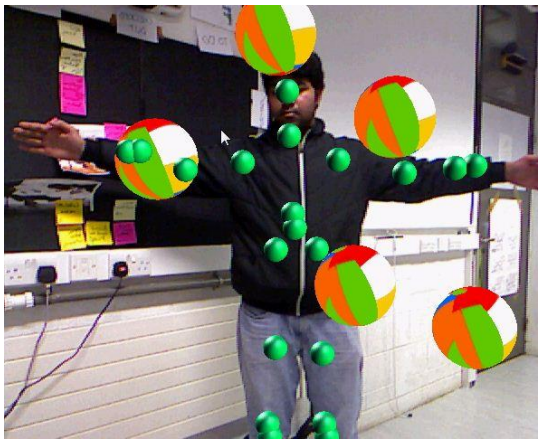

Figure 6: Simple skeleton of human body augmented in the live scene along with the augmented beach-balls

Again we setup target functions to be called for events when a pose/gesture match is recognized. When the pose (shown in Figure 1) is recognized, a primitive cylinder [10] is drawn at the hand joint location. This also marks start pose of our gesture which ends like as shown in Figure 2. When the right hand gesture is recognized, we display the weapon (the sword mentioned in section 1.0) at the right hand joint location. Gesture for left hand is similar but on the other side of the body as shown in Figure 3, after recognizing it we display an array of red and white primitive balls. Both left hand and right hand gestures could be detected simultaneously as shown in Figure 4. You will lose the weapon and the array of primitive spheres when their respective starting poses are detected. Otherwise you can use these sword and array of primitive spheres in all other orientations and locations in the scene.

The scene is also augmented by flying beach-balls through random location and in right to left direction as shown in Figure 6.

After obtaining the sword (refer section 1.0) with right hand gesture, we detect collision between the sword and the flying beach-balls using Bounding-Sphere collision detection method which falls under something known as Broad phase collision detection category of algorithms. After collision with the sword, beach-balls disappear from the scene. Bounding sphere collision detection considers a virtual sphere bounding the object and centered at the center of mass of the object to simplify the computation required to detect collision between complex objects.

5.0 Features

These are some of the salient features of my assignment work-

- 3D game-play
- Pose and gesture recognition.
- Human body skeleton visualization in 3D
- Specifying rotation without using any pointing device

6.0 Algorithms

Following algorithms that I have used are worth mentioning in this section-

- **Randomized decision forests:** Randomized decision trees and forests have proven fast and effective multi-class classifiers for many tasks and can be implemented efficiently on the GPU [8]**.** It is built into the Kinect drivers on there, which is then

provided to games and developers (you can also see its results if you have a development Xbox and go into the Kinect console). It provides an extremely reliable 3D joint orientation and location map which fully describes a human's pose in Cartesian space, for at least one person in the Kinect's view. This allows developers to build gestures and actions support on top of this to allow gamers to use Kinect to interact with the environment. Following figure is taken from [8] and it shows the performance of the algorithm.

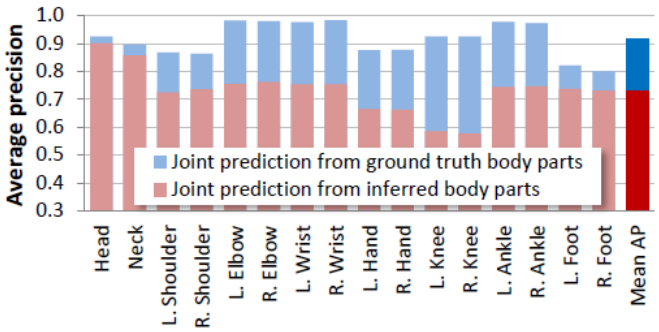

Figure 7: Joint prediction accuracy. Comparison of the actual performance of algorithm (red) with the best achievable result (blue) given the ground truth body part labels. From [8]

- **Holt double exponential smoothing and filtering:** When we are dealing with fluctuating data causing jittery behavior, we prefer filtering the data to attain smoothness. One way is to overage the current data with some of the previous data but it is does not give impressive results. Kinect sensor does not have sufficient resolution to ensure consistent accuracy of the skeleton tracking data over time. This problem manifests itself as the data seeming to vibrate around their positions. However, the Kinect SDK provides an algorithm in form of a class `TransformSmoothParameters` for filtering and smoothing incoming data from the sensor and is based on the Holt double exponential smoothing method, commonly used for statistical analysis of economic data. This algorithm provides smoothing with less latency than many other smoothing filter algorithms. The smoothing algorithm parameters can also be manipulated for desired user experience.

- **Nearest neighbor algorithm:** To track and classify pose and gestures. BeTheController package is a closed source freeware library and is still in the proceedings of being published as a paper. Hence much information is not available about all the algorithms it uses.

- **Bounding sphere collision detection:** Collision detection is a very computationally intensive process. Even for simple 3D objects as we have used in this assignment, it could slow down the whole simulation considerably. And implementing a very robust collision detection is a non trivial task. Mainly depending upon the requirement – one can choose one of the numerous collision detection techniques. For this assignment, I have picked Bounding sphere collision detection as it is one of the simplest and fastest collision detection algorithms as you do not need to check each mesh and curve of your modeled object against each mesh and curve of other object.

7.0 Decision choices

For accomplishing this assignment, I made some decisions and learned from them. This section is dedicated to mention some of them along with reason to choose them

- **Technology selection:** I choose to use Kinect sensor technology to use for this assignment after a thorough research about existing marker and non-marker based technologies for Augmented Reality as explained in section 3.0.

- **Game play objects within game:** My initial idea for the game was to have clouds instead of beach-balls. As the size and shape of cloud is something that is moderately complex for collision detection algorithm to produce good experience. For this reason I decided to choose an regular shape like sphere. Also for the fact that I am using Bounding sphere collision detection – using spherical objects was the best choice for realistic collision detection.

8.0 Demonstration Video

Demonstration of the game can be seen online at youtube.com – http://youtu.be/r8ceIJz9NnQ

9.0 Acquisition and use

The implementation source code is mainly in five files-

`Primitives/CylinderPrimitive.cs`
`Primitives/GeometricPrimitive.cs`
`Primitives/SpherePrimitive.cs`
`Primitives/VertexPositionNormal.cs`

Implementation of primitives (refer [10]) which are being used in the main game play to draw skeleton join locations and tool in the left hand (after left hand gesture

`CloudGame.cs` The main file containing the game logic. It contains class for beach-ball management, sword drawing, Kinect sensor software setup, routines for pose/gesture recognition and handling.

`Program.cs` This file just contains application entry point which by default is Main() function.

The whole project could be compiled using Microsoft Visual Studio 2010 (with XNA4.0 framework installed).

Minimum Hardware requirements: Kinect for Windows hardware, 2GB RAM, 2.6 GHz processor.

Minimum Software requirements: Microsoft Windows XP, Visual Studio 2010, .NET framework version 4.0, Kinect for Windows SDK 1.0, Microsoft XNA4.0, BeTheController library

10.0 Tools

Associated work has been done on Microsoft Windows7 Operating System using *Kinect for Windows SDK*, *Microsoft XNA4.0 Framework*, BeTheController library and Autodesk 3ds Max.

11.0 Limitation

- More than two players could not be supported.
- Camera only works for objects more than 80cm away from sensor (though efforts are being made to bring it down). And resolution of camera decrease as the object goes further away from sensor.
- Kinect hardware runs at 30 frames per second which could be an issue is user experience for some applications.

12.0 Future scope

There are couple of things to be worked-on in future-

- **Camera tracking of the player:** Kinect is capable of rotating the camera with much higher degree of freedom that any other sensing device had before. Once registered in the scene, a player could be tracked by Kinect sensors to orient itself in players direction.

- **Better gesture recognition:** The pose and gesture recognition provided by BeTheController library is much appreciable but at the same time, it lacks robustness as it is greatly affected by depth changes (within the Kinect sensor range).

- **Narrow phase collision detection:** Currently collision detection between sword and beach-ball is by using Bounding sphere collision detection which approximates the body of sword to a bounding sphere and then detects collision. This ends up in beach-balls being busted even before sword reaches them. A Narrow phase collision detection algorithm would detect exact collision of two bodies and would result in a better user experience.

- **Scaling of player skeleton depending on its distance from Kinect:** Player's skeleton as shown in Figure 5 is displayed in 3D, but it does not scale in the same proportion when the player goes away from the sensor.

- **Support two players:** Kinect support full skeleton tracking of maximum two players. Hence it is much logical to extend this solution for two players in the same scene.

13.0 Technology applications

This is an emerging technology which is in very young stage and holds potential to find applications in innumerable applications like (but not limited to) –

- **Automatic translation of sign-language to text.**
- **Free-hand painting.**
- **Patient help system.**
- **Motion capture.**

14.0 Conclusion

Kinect sensors are relatively new in research terms where it's potential is being explored to assist us solving long pending issues. It provides marker-less solutions to augmented reality applications in a way that was never perceived before. On one side it provides a

better platform to build augmented reality applications and at the same time makes it much easier by letting us avoid the careful operations like calibration before using it.

15.0 Acknowledgements

- Many thanks to Dr. Peter Redmond for providing crucial inputs about the technologies, game play and frequent discussions.
- Thanks to my colleague Amy Christel Davidson for helping me in making sword model.

16.0 References

[1] http://www.microsoft.com/en-us/kinectforwindows/

[2] http://rbwhitaker.wikidot.com/3d-tutorials

[3] http://labs.manctl.com/rgbdemo/

[4] http://www.ximplosionx.com/2011/06/19/intro-to-the-kinect-sdkdrawing-joints-in-xna/

[5] http://bethecontroller.com/

[6] http://www.mindtreatstudios.com/our-projects/kinect-3d-view-projection-matrix-rgb-camera/

[7] http://www.developerfusion.com/news/116479/microsoft-paper-reveals-kinect-body-tracking-algorithm

[8] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., et al. (2011). Real-Time Human Pose Recognition in Parts from Single Depth Images. IEEE Conference on Computer Vision and Pattern Recognition (2008), 2(3), 1297-1304. Ieee. doi:10.1109/CVPR.2011.5995316

[9] http://kinectdtw.codeplex.com

[10] http://create.msdn.com/en-US/education/catalog/sample/primitives_3d

[11] http://cm-bloggers.blogspot.com/2011/07/kinect-sdk-smoothing-skeleton-data.html