KinectBalls: An Interactive Tool for Ball Throwing Games

Jonathan Schoreels, Romuald Deshayes, and Tom Mens

Software Engineering Lab, NUMEDIART Research Insititute
University of Mons – UMONS, Belgium
`firstname.name@umons.ac.be`

Abstract. We present a tool that was developed in the context of the first author's masters project. The tool implements an interactive computer game combining the real and the virtual world in a seamless way. The player interacts with the game by throwing balls towards a wall on which a virtual 3D scene is projected. Using the *Kinect* 3D sensor, we compute and predict the trajectory, speed and position of the ball. Upon impact with the screen, a virtual ball continues its trajectory in the virtual scene, and interacts with the objects around it using a physical and a graphical 3D engine *Bullet*, and *Ogre3D*. The prototype game has been successfully tested on a large number of people of varying ages.

Keywords: Kinect, HCI, virtual reality, object tracking.

1 Introduction

Creating new games and entertainment applications using affordable state-of-the-art devices has gained a lot of recent interest thanks to the emergence of new HCI techniques and a trend towards the use of natural interaction. The first major step that revolutionized the gaming industry was Nintendo's Wii console, allowing humans to play games with body gestures. Microsoft responded with the Kinect sensor capable of seeing and reacting to the world in 3D. Since its release, an important number of applications using this sensor have been published on the internet [5,2,13,1].

While Kinect's main strength is its ability to track a user's body, it can be used in other ways to serve different goals. We exploited the raw information provided by Kinect's 3D sensor to track a moving ball. We integrated this in a prototype interactive game *KinectBalls* that bridges the gap between the real and virtual world. The aim of the game is to bring down a pile of virtual boxes by throwing a real ball towards them (see Figure 1). Videos of a live tool demonstration carried out with an audience of high school students and small children during a science fair at the University of Mons, can be found at `http://youtu.be/v02BcA-EPrI`. Others have recently developed a similar tool to simulate a pétanque game, using two high-speed PSEye camera's for ball tracking, and a webcam for face detection and tracking [3].

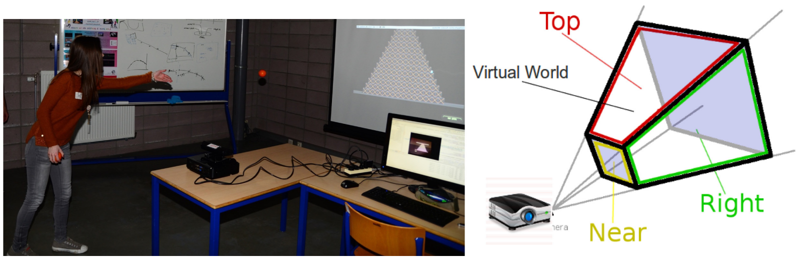

Fig. 1. Left: Live demonstration of the *KinectBalls* prototype. **Right:** The projection matrix (frustum) of the 3D virtual world.

The concept of the application is fairly simple. A beamer projects a virtual 3D world on a screen or wall. The player then throws a ball towards the projected scene. Using the Kinect, the tool tracks the ball's trajectory and speed and predicts the point and time of impact with the wall. At the predicted time and position of impact, a virtual ball is created with the same parameters as the real ball, and it continues its trajectory in the virtual world to interact with other virtual objects. Although there are some technical limitations in the current prototype implementation (low framerate, low resolution) they can easily be addressed by using other input devices than the Kinect (see e.g. [3]). A wide variety of ball throwing games could be implemented in a similar way (such as basket ball, penalty shots, bowling, petanque...) These games can for instance be used in small rooms to train the motor skills of young children by aiming accurately at objects in a virtual scene.

To implement our tool, we deliberately constrained ourselves to an as affordable solution as possible using inexpensive yet state-of-the-art devices capable of 3D vision, together with open source libraries for physical 3D rendering.

2 Object Tracking

One of the biggest problems with traditional ball tracking algorithms is their inability to take into account 3D information. To address this problem, stereo vision has often been used to retrieve information about the 3D position of the ball. The main limitation of this technique is that it requires a relatively complex setup. For example, to track a tennis ball, up to 6 cameras have to be placed and calibrated together before being able to accurately track a moving ball [12].

The problem we are addressing is more simple than tennis ball tracking because the area where the ball needs to be tracked is much smaller and the hand-thrown ball to be tracked moves at a much slower speed (a few m/s). On the other hand, the system has to respond instantly, that is to say, even before the ball hits the wall. This requires us to predict the ball's trajectory and its impact with the wall. We will see in Section 3.2 how this can be achieved.

Over the last decade, a lot of effort has been put in finding robust algorithms to track a moving ball in realtime (e.g., for soccer games, tennis games, golf etc.)

[15]. Using 2D cameras, the main challenge resides in being able to differentiate between the object to be tracked and the rest of the scene. To do so, many techniques exploit chromatic and morphological [7] features of the object to be tracked. Other techniques for object tracking have been developed since. The most widely used one is the so-called *fiducial*-based tracking. This technique uses visible markers placed on the tracked object, thus improving the speed, robustness and accuracy of the tracking algorithm [9,14]. The major drawback of such techniques is that they are invasive as the tracked objects have to carry markers. When objects with strong contours have to be tracked on non-cluttered scenes, a RAPID-like method can be used [8,10]. The main advantage is that this method is quite simple and was also one of the first methods to run in real-time. Many enhancements for this method have been proposed to make it more robust, such as the use of a more complex least-squares curve fitting method [6]. A more detailed overview of the different kinds of tracking techniques can be found in the excellent survey proposed by [11] that discusses most popular model-based 3D tracking methods.

3 Architecture of KinectBalls

We have developed our tool in a modular way, in order to facilitate changes to (1) the characteristics of the moving object, (2) the game application and (3) its 3D rendering. Figure 2 shows the 4 modules of our tool: data acquisition, object detection, trajectory prediction and graphical rendering.

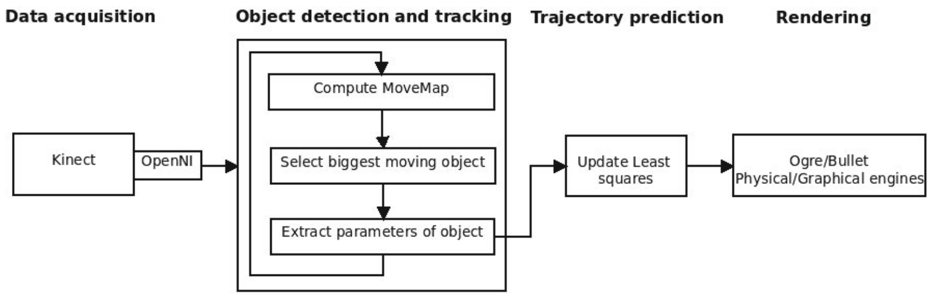
Data acquisition
Object detection and tracking
Trajectory prediction
Rendering
Compute MoveMap
Kinect
OpenNI
Select biggest moving object
Extract parameters of object
Update Least squares
Ogre/Bullet Physical/Graphical engines

Fig. 2. Architecture of the prototype framework

3.1 Object Detection and Tracking

The most widely used affordable 3D sensor today is undoubtedly the Kinect. Its infrared projector and sensor allow to analyze and create a complete depth map of the observed scene in real-time at a framerate of 30Hz. We used this sensor to ease the detection and tracking of moving objects.

An important challenge is to differentiate between the background and the moving object. The infrared camera provides a set of successive frames representing snapshots of the observed scene and constructs a *depth map*, i.e., a 3D image where each pixel has three (x,y,z) values representing the exact position in metric space w.r.t. the camera. By comparing a frame F_n with the previous frame F_{n-1}, we compute the difference in depth (z-axis) for each pixel. If this difference exceeds a certain threshold T (allowing us to filter out noise) and if a sufficient number of adjacent pixels have undergone a similar difference in depth, we conclude that something has moved. We apply this idea to create a matrix

$$MoveMap(i,j) = \begin{cases} 1 & if\ |F_n(i,j) - F_{n-1}(i,j)| > T \\ 0 & otherwise \end{cases}$$

Every 1 in $MoveMap$ corresponds to a moving pixel in terms of depth. The 0's are considered as being part of the background. Using this matrix we can easily track moving objects using the following setup. The camera faces the wall on which a virtual scene is projected to which the ball will be thrown, thus the only moving object that will be detected is the ball (since all other objects will not change position). However, due to imprecision of the 3D sensor, the edges of some of the objects composing the scene might still be detected as moving. Therefore, to improve the robustness of the tracking algorithm, we look for the biggest square of 1's in $MoveMap$. We can assume with a fairly high confidence that the biggest moving thing in the scene is the ball.

The next step of the algorithm is to detect the shape of the moving object. With the technique of the biggest square, we only get an approximation of the moving object's shape. By refining our algorithm, we consider all the adjacent 1's, compute the centroid of this new shape and use this point as the position of the ball to approximate the trajectory.

3.2 Trajectory Prediction

To predict the moving ball's trajectory, we store the centroid computed on each frame F_n. As soon as we have at least 2 positions of the ball (i.e., two frames in which a sufficiently big square was detected), we use a least squares regression model to approximate the trajectory of the ball with 3 second-degree polynomials (1 polynomial for each axis). At each new frame where a moving ball is detected, we update the regression model by taking into account the newly detected position of the ball.

Knowing the exact 3D position of the wall on which the virtual scene is projected, we use the computed regression model to predict the position and the time at which the ball will hit the wall. The speed of the ball is also computed using the derivative of the position. The closer the ball gets to the wall (and the more data points are used), the more precise the trajectory prediction will be. At the predicted time of impact, a virtual ball is created at the predicted position. The virtual ball will continue its route using the regression model parameters provided by the trajectory approximation algorithm.

To transform a position in the real world to a position in the virtual world, we convert the coordinate system of the real world (given by the Kinect) into the coordinate system of the virtual world (computed by the projector's parameters and its position relative to the screen and the Kinect). Thus, we calibrated [4] these two devices by calculating their intrinsic parameters. The extrinsic parameters are estimated using a matrix $M = (R, T)$ containing the rotation R and the translation T to be applied.

To create an immersive virtual world, we modified the projection matrix defined by a perspective frustum (i.e., a pyramid lying between two parallel planes cutting it, see right of figure 1) of the 3D rendering engine to match the projector's intrinsic parameters. We measured the distance between the projector and the screen, the vertical and horizontal size of the screen to set the frustum parameters. This way, we can use the same scale in both worlds, and we can easily create an impression of a virtual box (increasing the level of realism when the ball continues through the virtual world).

4 Lessons Learned and Future Work

We tested our setup with an Intel i5 computer with 4Gb RAM and an ATI 7850 graphical processor during a full day in front of a live audience. Calibration was a challenge, since it depends on the angle of width and position of both the projector and the Kinect relative to the screen. We did not take into account the distortion of the camera and projector because the precision of the Kinect was not sufficiently high to gain any important benefit. For higher resolution devices, distortion should be taken into consideration.

Kids of 5 years and older interacted with the game very enthusiastically and without requiring any explanation. With a supple throw, between 4 to 10 successive positions of the ball were detected. With this amount of points, the precision of the predicted impact of the ball varied between 1 and 5 centimeters. Only when the ball was not long enough in the field of view, or when it was thrown too fast or too straight, the resolution and framerate of the Kinect did not allow to compute the trajectory correctly. Doubling the framerate to 60Hz would mostly solve this problem.

Some adult players reported a lack of immersion, because they had difficulties to interpret the 3D virtual world, as it was only projected in 2D on the screen. Using a stereoscopic 3D projector could address this problem. Another way to make the game more immersive is by tracking the position of the player w.r.t. the projector using a second Kinect device. The rendered virtual scene can then be adapted to match the user's relative position to the screen.

The game could be extended into a multiplayer version with multiple balls thrown simultaneously. This would require to identify different balls (e.g. based on their color), and detecting possible collisions between them.

5 Conclusion

We presented *KinectBalls*, an interactive game capable of tracking a moving ball thrown towards a projected virtual scene. The technique used for achieving this requires only one very affordable 3D sensor to track the ball and predict its trajectory and impact on the wall. The developed algorithms are fast enough to run in real time on a standard computer. After calibration, the solution worked fine in all tested indoor situations, but various improvements can be made to increase the level of immersion.

References

1. Bailly, G., Walter, R., Müller, J., Ning, T., Lecolinet, E.: Comparing free hand menu techniques for distant displays using linear, marking and finger-count menus. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part II. LNCS, vol. 6947, pp. 248–262. Springer, Heidelberg (2011)
2. Boulos, M.K.N., Blanchard, B., Walker, J.M.C., Tripathy, R.G.-O.A.: Web GIS in practice X: A Microsoft Kinect natural user interface for Google Earth navigation. Int'l J. Health Geographics 10, 14 (2011)
3. Dalpé, S., Monat-rodier, J., Riendeau, G., Voutsinas, P.: Poly-pétanque. In: Int'l Meeting on Virtual Reality and Converging Technologies, LAVAL VIRTUAL (2013), `http://youtu.be/0Z2VDdaS3rs`
4. Deshayes, R.: Reconstruction algorithmique d'objets 3D. Master's thesis, Faculty of Sciences, University of Mons, Belgium (June 2011)
5. Deshayes, R., Jacquet, C., Hardebolle, C., Boulanger, F., Mens, T.: Heterogeneous modeling of gesture-based 3D applications. In: MoDELS Workshops (2012)
6. Drummond, T., Cipolla, R.: Real-time visual tracking of complex structures. IEEE Trans. Pattern Anal. Mach. Intell. 24(7), 932–946 (2002)
7. Gong, Y., Sin, L.T., Chuan, C.H., Zhang, H., Sakauchi, M.: Automatic parsing of TV soccer programs. In: IEEE Int'l Conf. Multimedia Computing and Systems (ICMCS), pp. 167–174 (1995)
8. Harris, C.: Tracking with rigid objects. MIT Press (1992)
9. Hoff, W.A., Nguyen, K., Lyon, T.: Computer vision-based registration techniques for augmented reality. In: Intelligent Robots and Computer Vision XV, pp. 538–548 (1996)
10. Klein, G., Drummond, T.: Robust visual tracking for non-instrumented augmented reality. In: IEEE/ACM Int'l Symp. Mixed and Augmented Reality (ISMAR), pp. 113–122. IEEE Computer Society (2003)
11. Lepetit, V., Fua, P.: Monocular model-based 3D tracking of rigid objects: A survey. Foundations and Trends in Computer Graphics and Vision 1(1), 91 (2005)
12. Pingali, G.S., Opalach, A., Jean, Y.: Ball tracking and virtual replays for innovative tennis broadcasts. In: Int'l Conf. Pattern Recognition, pp. 4152–4156 (2000)
13. Ren, Z., Meng, J., Yuan, J., Zhang, Z.: Robust hand gesture recognition with Kinect sensor. In: ACM Int'l Conf. Multimedia, pp. 759–760 (2011)
14. State, A., Hirota, G., Chen, D.T., Garrett, W.F., Livingston, M.A.: Superior augmented reality registration by integrating landmark tracking and magnetic tracking. In: SIGGRAPH, pp. 429–438 (1996)
15. Tong, X.-F., Lu, H.-Q., Liu, Q.-S.: An effective and fast soccer ball detection and tracking method. In: Int'l Conf. Pattern Recognition, pp. 795–798 (2004)