SHADOW GAME-IMPLEMENTATION

Jan CINERT [1], **Patrik ROŠTÍK** [2]

1 INTRODUCTION

After we've done a lot of research, we decided to create a dodge based game, where a player must not only move around to avoid balls, but try to reach and pop life balls as well. Another functionality might be added. The main goal for us, as the game developers is to create interaction that is simple and easy to understand, encourage expressive and creative movement of entire body. The principle is rather simple: A player is positioned between a projector and a screen and is facing the screen on which the game is projected (see Fig. 1) The player is directing his moves according to his silhouette. A camera is reading the scene picture at (approx.) 20 fps. Game is written in C++ using the OpenCV library.

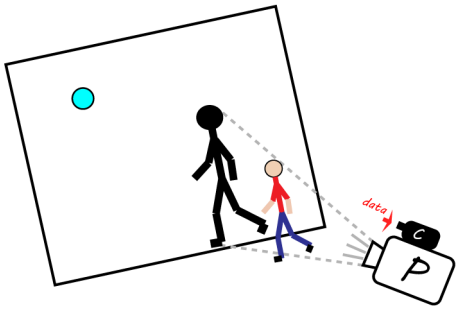

Fig. 1: Figure demonstration

The image captured by camera (or web cam) needs to be processed in order to find position of the player. We're using a simple background subtraction method. Output of this method is binary image, where zero values represents pixels of the camera image with no difference to according pixel values in background image. To label a pixel a foreground pixel (i.e. players silhouette) we demand the difference to be at least equal or higher than a threshold. This is a technique to get rid of image noise, minor changes in illumination, etc. Advantage of this implementation is a good performance and simplicity. On the other hand, a background picture needs to be recorded before the game is stated.
An image captured by the camera isn't always the same as the one we project. In fact, only in an ideal case they are both identically same. Otherwise the camera captures the projection area and little more of a space surrounding it. That makes a difference if we search for ball and the player's silhouette interaction. To do this we compare ball position in the projected image and the player's silhouette found in a picture from the camera. In this point is necessary to capture exactly the area of the projection (nothing more, nothing less), else the ball popping won't be precise, or even won't work at all. The solution for

[1] Jan Cinert, student of the master study programme Applied Sciences and Informatics, specialization Cybernetics, e-mail: cinax@students.zcu.cz
[2] Patrik Roštík, student of the master study programme Applied Sciences and Informatics, specialization Cybernetics, e-mail: rostpa@students.zcu.cz

this problem is precise camera zooming, which may be problematic (especially using web camera). Not mentioning, that in this case we need to position the camera in the axis of projection to reduce captured image deformation. One better solution is to make a camera calibration. Easy way to do it is to project at least 2 objects (e.g. 2 circles) - so we know their coordinates in the projected picture. The next step is to find those objects in the captured image and compare the coordinates in both frames. Doing that, we know the change of scale and transition of any object we project comparing to his image in the grabbed frame from the camera. That guaranteed us correct calculation ball player's silhouette collisions.
Supposing we know where the player is located and we have coordinates of all existing balls, we check for any ball interaction with the players silhouette. An effective algorithm is to look for an foreground pixel in the background subtraction output image at the location of every ball. To be precise we are looking for a foreground pixel interfering in all the area a ball covers.
Positioning of the balls is done randomly. Each ball speed is determined with respect to the place where it appears. For example it is undesirable for the ball to go straight down, when it appears in the bottom of the game screen. The ball is the faster the higher the level is. To improve the fun factor it was necessary to set all parameters to values, which make the game playable by beginners in the first couple of levels and also quite hard to play in the higher levels. By the word parameters I mean: ball positioning locations, ball speed, ball speed increasing through the levels up, number of balls, increasing rate of numbers through the levels up, etc. It was also important to optimize the program to get high enough and stable FPS.

2 CONCLUSION

The Shadow game was originally developed in Processing using Java OpenCV libraries. Due to the amount of data, which need to be processed, we reprogrammed the game using the OpenCV C libraries and C++. Java OpenCV libraries are limited and picture processing operations are computational expensive. One of future improvements would be adding sound effect. It would provide the player more in game feel. It also would give the game a rhythm and it'd be definitely funnier to play.

REFERENCES

Bradski G., Kaehler A., 2008. *Learning OpenCV.*, O'Reilly Media, Inc., Sebastopol, USA

Šonka M., Hlaváč V., Boyle R., 2007. *Image Processing, Analysis, and Machine Vision.*, 3rd Edition, Thomson Engineering, Toronto, Canada