Dodge 'Em

CSEE 4840 Final Project Design - Spring 2011

Jaiseung Bang jb2861@columbia.edu, Vincent Liao vl2187@columbia.edu,
Arunagiri Venkatesan av2294@columbia.edu, David Yang qy2114@columbia.edu

March 22, 2011

Abstract

This document presents the design details of a controller-free, object-avoidance game. The system uses video processing to identify the human body.

1 Introduction

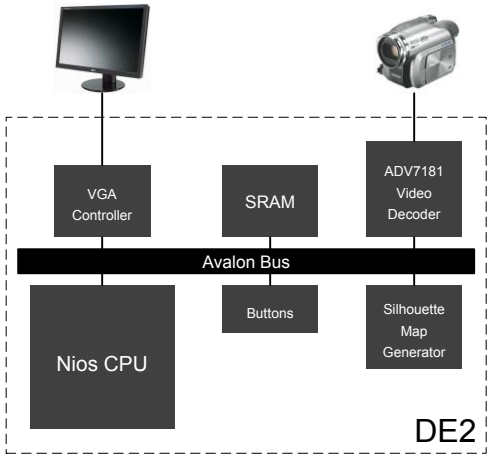
Figure 1: Mock Game Screen

The game takes in input from a digital camera, and runs through an algorithm to detect and track the human body. A silhouette of the person would be projected onto the screen. The player would have to move in a way that makes sure that the projected silhouette dodges incoming projectiles/obstacles.

Our implementation would encompass both software and hardware components. The camera would be able to locate and track a person. The stream would get fed into the video decoder in the DE2 board. This would then go to the Silhouette Map Generator, where the CPU figures out and creates a silhouette where the player is located in the camera. The frame of the silhouette is stored in the SRAM. The software would figure out the location of the incoming projectiles/obstacles and also keep track of score (the number of projectiles/obstacles dodged).

2 Hardware Overview

Figure 2: Hardware Block Diagram



2.1 Video Decoder

The ADV7181 video decoder on the DE2 board will be used to convert the NTSC signal from the camera to 4:2:2 YCrCb component video data. The video decoder will take the DE2s 27MHz global clock as clock input.

2.2 SRAM

The ISSI IS61LV25616, on the Altera DE2 FPGA board, is a high-speed, 4,194,304-bit static RAM organized as

262,144 words by 16 bits. The pin descriptions for the SRAM is listed in figure 3.

Figure 3: SRAM Pin Descriptions

Pin	Description
0-A17	Address Inputs
I/00- I/015	Data Inputs/Outputs
CE	Chip Enable Input
OE	Output Enable Input
WE	Write Enable Input
LB	Lower-byte Control (I/00-I/07)
UB	Upper-byte Control (I/08-I/015)
NC	No Connection
VCC	Power
GND	Ground

The SRAM would be used for several purposes. If we decide to implement body detection through before/after pictures, we will store a heavily averaged pixel data of the background so that we can refer to it to keep track of the player once he/she steps in the frame. This may be of a size of 160X120 at 16 bits/pixel. SRAM would also store the location of the obstacle/projectile and the score (how many obstacle/projectile the player dodged). As discussed in software section, the resolution that we chose should be suitable for tracking peoples movement and have basic hit/miss detection. We may have an option for having different shapes/obstacles so we may need to store the shape in the SRAM as well (we may have a preset number of different shapes that corresponds to different numbers).

One of the things that we need to take into consideration is that the SRAM can not be read and written to at the same time. This is a non- issue because the only the C program would be writing to the SRAM, to update the location of the obstacle/projectiles and maybe to specified the shape.

2.3 Silhouette Map Generator

The YCrCb video data from the video decoder will be compressed into a silhouette map which describes where the image deviates significantly from the initial background image stored in the SRAM. This computation will be done in a hardware peripheral communicating on the Avalon bus.

2.4 VGA Controller

The project will project the game interface onto a VGA display. A VGA controller will be implemented as an DE2 peripheral communicating on the Avalon bus. This controller will take a 25MHz clock input (half the DE2s 50MHz global clock) in accordance with the VGA pixel clock frequency.

2.5 Pushbutton Controller

The pushbuttons on the DE2 will be used to trigger game functions. These functions include, but are not limited to, background image capture and start game. A pushbutton controller will be implemented as a DE2 peripheral communicating on the Avalon bus.

3 Software Overview

The software section of our project is broken up into 3 main parts. The first part is taking in the video feed and detecting where the players body is. The second part is generating the obstacles and their trajectory paths. Finally, the last part would be detecting the collision between the person from part one and the object generated from part two. The software will be written in a combination of VHDL and C.

3.1 Body Detection

There are two proposed body detection algorithms that our group decided to look into: Background Subtraction and Green Screen. Background subtraction is when at the start, we take a photo of the setting without the user being in it and use it as a base, then when the user steps in, the computer will be able to compare the current picture and the base to figure out which pixels are different. The different pixels are the ones where the user is. Green screen uses a similar idea except that the background will all be green and the computer will detect anything is not green as the user. We decided that the image taken in from the camera will be compressed before any processing on it is done. Compressing it will save memory, processing speed, and data transfer which results in smoother outputs on the screen. We decided to compress the image by 16 times by averaging 4 pixel by 4 pixel squares.

We decided that background subtraction algorithm will be our main algorithm and the Green Screen will be the backup. Background subtraction is more flexible in that we do not need to prepare a green screen and that it should work in any environment. However, there are some complications with background subtraction that we need to account for in order for it to work.

The major complication of background subtraction is making sure that the background does not change from the original base background. For example, after a person steps into the frame, not only will his/her body change the camera image, but the shadows will as well. Even if people are not in the frame, their shadows might be or we are close to a window and the sun moves, all these will affect the current image.

Another thing that might cause the difference is white balance. If the camera auto adjusts, when a person steps into the frame, the overall range of colors will also change. That means the camera will auto adjust to a different white balance which changes the color of every original background pixel.

To resolve these complications, we will need to create a good physical setting to minimize the problems as well as using software code to account for them. For the physical setting, we will need to have good lighting from the front so that the shadows created by the user will not be at an angle but perpendicular to him or her. The lighting will also flood the setting so any change of lighting such as the sun will not affect the overall frame of the picture as much.

When comparing the color components of the current pixel to the base pixels, we need to define a level of tolerance. A proper tolerance can be determined manually after testing out different levels.

The algorithm is as follows:

1. Split the initial background image into squares of 4 pixels by 4 pixels.

2. Generate a compressed image by averaging the color components of each 4 by 4 square.

3. Split the current image into squares of 4 pixels by 4 pixels.

4. Generate a compressed image by averaging the color components of each 4 by 4 square.

5. For each square in the compressed image, if any of the color components differs from the initial background images color components by more than the tolerance, it is flagged as in the silhouette. Else, it is flagged as part of the background.

3.2 Obstacle Generation

Another software program will be the generation of obstacles and projectiles that the user will dodge in the game. These objects will be generated randomly and based on the type of game we are playing; the trajectory will be different as well. The generation will be similar to the lab we did in class where the C program will randomly generate the coordinates and shape of the object and the VHDL will draw it accordingly. For example, if the C program randomly chooses to draw a ball, it will pass in to VHDL the coordinates for the center as well as that it is a ball. The VHDL will know to draw the ball with a given radius and coordinates as the center. If the C program passes in the coordinates for a square, the VHDL will know to how to draw the square using a different set of code. For now, we will generally stick with just generating circles first and once everything works, implementing other shapes will be pretty fast.

One consideration that we need to take into account in obstacle generation is to make sure that we do not have overlapping obstacles. Because the C program only knows the one coordinate for the shape, it does not necessarily know how big the shape is. For example, lets take an extreme case; the C program drew a circle that takes up the whole screen. The C program only knows the coordinates for center of the circle, it might generate another set of coordinates for another circle or square which will obviously overlap with the large circle because it takes up the whole screen. Thus, there must be a storage space of some sort that the C program will keep track of all things that were made and about how big they are so that if a random object is generated and overlaps with a already made object, it will know to generate a new one.

3.3 Collision Detection

Collision detection will be done mainly in VHDL. Similar to how the VHDL knows when to draw a pixel in that the circle variable was on, each shape and the user silhouette will have its own variable and the VHDL will

know when to draw the pixel if any of the shape variables is on. Collision detection is a matter seeing that if both the silhouette and another shape are on at the same pixel, then there is a collision at that place. We just have to make sure that collision is only detected between the silhouette and another shape, not between two or more shapes.

4 Milestones

Milestone 1 (March 29)

Prototype an algorithm that can separate the player from the background using a high level language (C, Java, Python, Matlab). This can be done through one of many methods, such as taking a before and after picture and getting the difference or by using a solid background and tracking what is not colored the same as the background color.

Milestone 2 (April 12)

Implement the silhouette tracking and display. The camera will be used to keep track of the players movement and display his/her silhouette.

Milestone 3 (April 28)

Implement basic game logic and collision detection. We should have a rudimentary running game that has incoming obstacles and have some sort of interaction between the silhouette and the obstacle/projectile (the obstacle/projectile could freeze when a hit is detected, for example). Scoring and final version of the game will be implemented after Milestone 3 before the final report.

5 References

- `http://www.cs.columbia.edu/~sedwards/classes/2011/4840/Analog-Devices-ADV7181-video-decoder.pdf`
- `http://www.cs.columbia.edu/~sedwards/classes/2011/4840/ISSI-IS61LV25616-SRAM.pdf`