Virtual Ping Pong Proposal

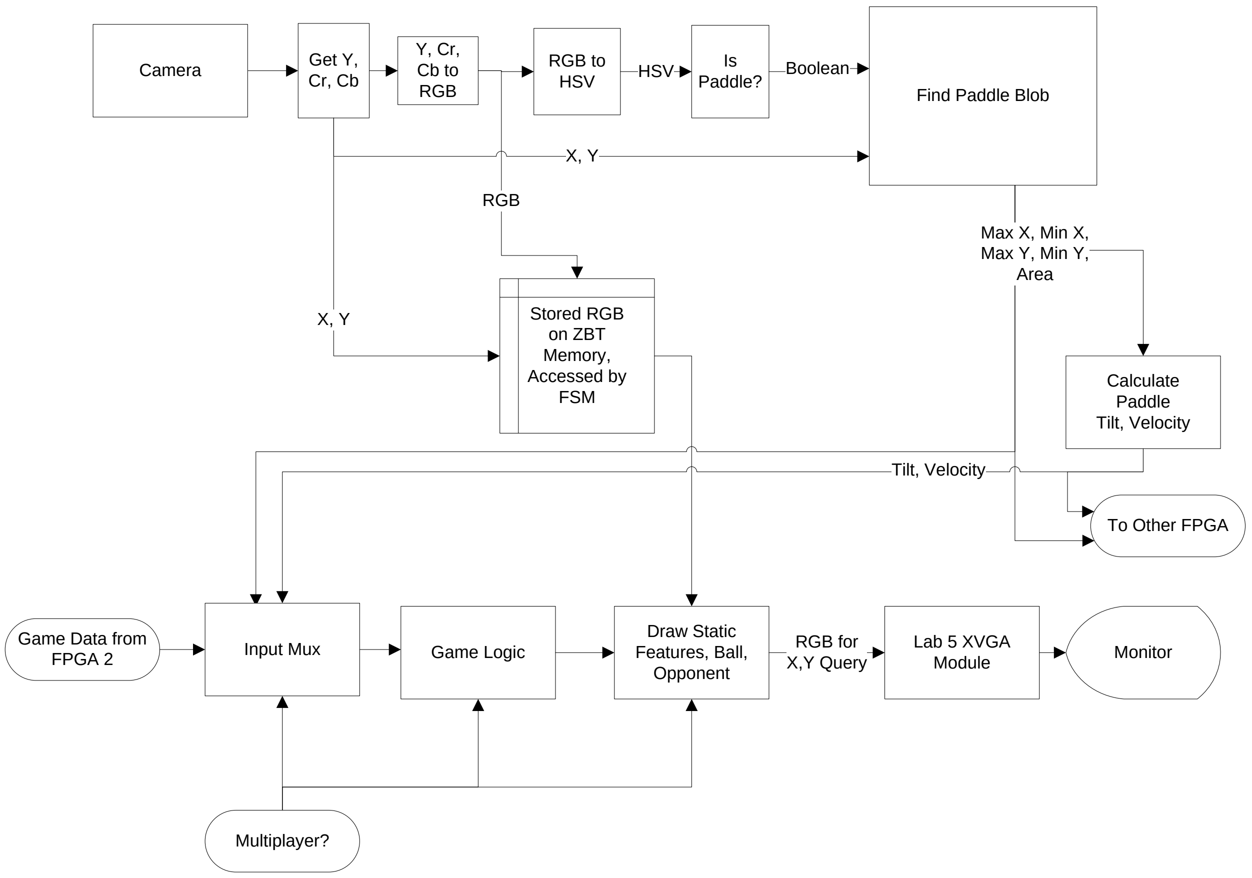
Zach Clifford, Mark Stevens

Overview

Our project will be a virtual ping-pong game that could be played in multi-player or single-player modes. In the single-player mode, the game will resemble virtual squash. The interface will display a three dimensional room populated with a ball. This ball elastically bounces off of walls and is affected by gravity. When the ball reaches the "front" of the screen, the user will have to swing a paddle to hit the ball back into the screen as in squash. The system will be able to detect paddle tilt and speed to make the ball return realistically.

Our project will also have a multi-player component. In this mode the back wall will be replaced by a video image from the camera pointing at the other player. Players will volley the ball back and forth. The first player to miss a return loses the game.

Block Diagram



Modules

Get Y, Cr, Cb

Translate the serial data from the camera into Y, Cr, Cb for each X,Y value. An LA showed us a staff-created module that already does this.

Y, Cr, Cb to RGB

Perform the conversion, which is simple and involves some multipliers and adders.

RGB to HSV

Convert RGB to HSV, which is simple and involves some multipliers and adders.

Is Paddle?

Store an experimentally determined range of H, S, V that correspond to the paddle color. Check to see if the current HSV values are within this range; output true if they are within range and false if they are not.

Find Paddle Blob

Take the Boolean description of whether or not the current pixel is paddle-colored and generate a description of the paddle-colored blob once per frame. This block is fairly complex and will consist of several components:

- Reset conditions – set area to 0, max values to 0, and min values to the width and height of the input.
- End of frame – output the calculated max, min, and area, and reset the other values using the rules for reset.
- Low Pass Filter – ignore any spurious paddle-colored pixels, and focus only on groups of pixels that are probably part of the paddle.
- Update max, min values: See if the current pixel is a more extreme top left, top right, bottom left, or bottom right than that stored. If so, update the appropriate value. The specific signals output may alternatively include a center point or just a top left and bottom right, depending on what is most useful for the game logic block.
- Update area by adding 1 if the pixel is paddle-colored

Calculate Paddle Tilt, Velocity

Use the max/min data from the paddle blob block to calculate how tilted the paddle is at any time. This can be done by comparing the width of the top of the paddle to the width of the bottom, or by comparing the height of the blob to the width of the blob.

Calculating the paddle velocity will require comparing how the area of the blob changes over time (for velocity towards or away from the screen. Other lateral velocity may be added later; this can be calculated based on the change in X, Y coordinates over time.

ZBT Memory Interface

The ZBT memory will be used to buffer camera input for display in the multiplayer version of the game. As pixels come in from the camera, they will be converted to RGB and stored in the ZBT memory. The state machine there will negotiate this exchange. In a previous term 6.111 provided a module to accomplish this, so we plan to use that. It is designed as a framebuffer, so it should serve our purposes for this project.

Game Logic

The Virtual Ping-Pong game logic will be mostly controlled by the Game Logic block. This block will determine the X, Y, Z coordinates of the ping-pong ball inside the three dimensional space of the game. This information will be sent to the Output block for rendering to the screen. The inputs to this module will include the current position of the player's paddle, the paddle's tilt, and the paddle's velocity as determined by the Find Paddle Blob module. The position will be represented as a rectangle determined by the upper left corner and the lower right corner.

The Game Logic block will take this information to update the location of the ball every update period. While the ball is in the virtual space, the ball will move without user input. It will bounce elastically off the side walls and the back wall. The ball will also be affected by gravity that will constantly bend its velocity towards the bottom wall.

The inputs to the module from the user's paddle will only be used when the ball reaches the "front" of the play field. The module will first make sure that the ball was returned successfully, i.e. the center of the ball lies within the area identified to be the user's paddle. If that was not the case, the game will end and wait for a reset. If it was returned, the Game Logic will then alter the velocity of the ball in response to the tilt and paddle velocity. This will cause the ball to continue back into the field to start the process over again.

This module will have the capability to serve in a multiplayer capacity. By connecting two labkits together the "back wall" could instead be an opponent. The loser of the game would then be the player who failed to return the ball. The Game Logic in one of the labkits would become the "master" that does all of the calculation related to the physical ball. The other would be a "slave" that just passes through X,Y,Z coordinates from the master. The Input Mux would allow the master to pick which paddle information it is interested in (i.e. which player is about to be required to return the ball).

Drawing

The "Draw Static Features, Ball, Opponent" (Video Output) module will be responsible for creating the play field for display to the XVGA monitor. It will always draw a centered rectangle with lines extending from each corner to the corner of the screen. This will produce a fake three dimensional effect. It will have a round "Ball" sprite to draw the ball where it belongs based on the X,Y,Z coordinates given from the Game Logic module. It will perform a transform on those coordinates to determine the size and X,Y coordinates on the monitor that should be drawn. The ball sprite will then be able to take a pixel as input and determine whether that pixel should be shaded or not. The module will also draw a rectangle where it believes the paddle is located.

In multiplayer mode, this module will be responsible for drawing the opponent instead of a plain "back wall". The labkit camera will be pointing at the opponent, so the input from that camera can be drawn into the rectangle at the center of the screen. A different block will be responsible for buffering this data to the ZBT memory to make a framebuffer. The Video Output module will simply display this framebuffer to the screen. A small FSM will negotiate the reads from the ZBT memory for this task.

Output to Screen

The final module will be the actual Lab 5 XVGA module. This will be copied from Lab 5 almost verbatim. It manages the timing of output to the video display. It outputs the pixel it needs information about. It also outputs information about whether the screen is blanking and vsync/hsync data. It expects RGB input about the pixel it is requesting.

Division of Labor

Mark will handle the first half of the modules – doing the vision processing – while Zach will create the second half, the game logic and display.

Testing

Input modules can be tested incrementally as they are created. Once the rgb module is created, this data can be output to the screen and checked for accuracy. Likewise, the hsv data can be sent to the screen and checked later on. The pixels detected to be part of the paddle can be checked by printing out a white overlay on top of the hsv data; the values for paddle detection can also be checked using hsv conversion in Java (using Maslab code).

The Find Paddle block can be tested by coloring the calculated paddle corners and displaying the area – probably on the FPGA hex display. The Tilt and velocity module can also be tested by printing to this display.

The Output and Game Logic modules will be tested first with a closed box. The Game Logic will initially ignore the paddle inputs and assume that the ball is always returned. The ball can then bounce randomly within the cube to see that the display works and that the physics of the ball bouncing looks realistic. Once this is done, adding the paddle location should be relatively simple.