Friday, October 25, 2012
6.111 Final Project Proposal

Augmented Reality Fruit Ninja™

Nathan Monroe, Drew Dennison, Isaac Evans

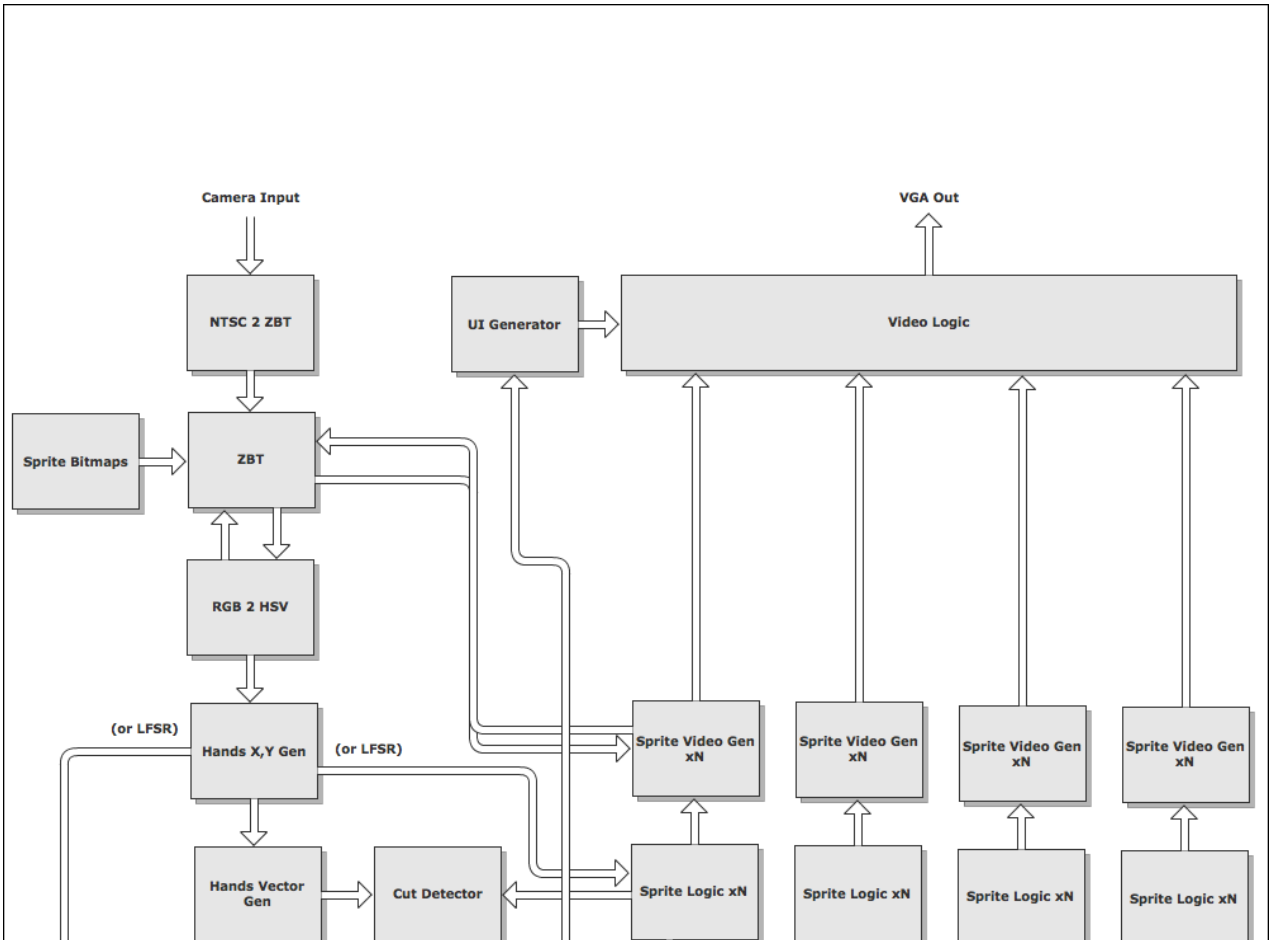
6.111 Final Project Proposal:

Augmented Reality Fruit Ninja

Overview

This project is inspired by Fruit Ninja™, a video game in which fruit appears on a touchscreen and a user must swipe across the fruit to cut it in half before it hits the bottom of the screen. We propose to implement augmented-reality Fruit Ninja™ on the FPGA labkit, using a VGA camera to track the user's gloved hands which will act as the controller for the game. We will display a dimmed and slightly blurred version of the the live input feed underneath our game overlay, which includes pieces of "fruit" that the users must "cut" by waving their hands. The users will wear colored gloves to facilitate hand tracking. As time allows, we will implement more advanced graphics, sound and physics for the game to create a more immersive user experience. This project will require the standard 6.111 labkit, as well as a VGA camera and colored gloves. The basic functionality is that the FPGA calculates whether a fruit is "cut" by comparing the locations of the users' hands to that of the flying fruit, and updating the game according to whether the fruit is cut or not.

Block Diagram



Friday, October 25, 2012
6.111 Final Project Proposal

Modules:

1. **NTSC2ZBT**
   Takes in the NTSC camera data and stores it as a frame in the ZBT RAM. This module should already be available from our TA, Kevin.
   Inputs: NTSC ports built into labkit
   Outputs: A 640x480 image frame in ZBT RAM.

2. **ZBT**
   RAM storage for camera frame data and sprite bitmap data. Sprite bitmap data will be statically loaded at compile-time.

Inputs: NTSC2ZBT, precompiled bitmaps
Outputs: camera frame images, bitmaps

3. **RGB2HSV [x(640x480)]**
   Converts RGB pixel map to HSV for better hand recognition. This module should already be available from our TA, Kevin.
   Inputs: R, G, B (red, green, blue) pixel color
   Outputs: H, S, V (hue, saturation, value) pixel color

4. **HandsXYPos**
   Takes in frame data and generates the hands position as XY coordinates. Initially this module will probably calculate the center of mass for a given hue range; later, more advanced image recognition techniques may be used (blobbing, connected-components, etc.).
   Inputs: One 640x480 camera frame as HSV coordinates
   Outputs: Two (x, y) coordinate pairs corresponding to the user's hands

5. **HandsVectorGenerator**
   Buffers a series of recent hand positions and generates a vector for hand motion based on a function which stores the past several hand coordinates and builds the vector based on weighting the coordinates with an exponentially decaying weight function.
   Inputs: One (x, y) coordinate pair representing a hand position
   Outputs: An (x, y) vector representing the hand's motion vector over the past 250 milliseconds.

6. **CutDetector**
   Inputs vectors from HandsVectorGenerator, Sprite Positions from SpriteLogic, and SpriteStates from the GameLogic to determine if any of the sprites have been 'cut' by the hands. The cut detection is based on a simple bounding box detection for each sprite.
   Inputs: two hand vectors and some arbitrary number of sprite positions and states.
   Outputs: cut boolean for each sprite

7. **GameLogic**
   The game logic records and manipulates the overall state of the game. It will activate fruits on a pseudorandom basis (with either an LFSR or from input from HandsXYGen) and will turn off fruits either when the cut detector tells them they have been cut, or when they reach the bottom of the screen, based on position input from the Sprite Logic. It will also track game-level logic such as the score, number of lives left, level number, etc., to send to the UI generator. The sprite state will be a single boolean: 1 if the sprite is actively on the screen and in the game, or 0 if not.
   Inputs: HandsVectorGenerator, HandsXYPos, CutDetector
   Outputs: game state information to UIGenerator, updated position and state information to each SpriteLogic instance

8. **SpriteLogic**
   This module keeps track of the sprite position based on game physics, and input from the game logic for if that sprite should be on or not. It knows that on a 0→1 transition of the "sprite active" signal, the sprite is entering the game and should be flying up from the bottom of the screen and uses LFSR or input from the HandsXYPosGen for pseudorandomization in placement. The Sprite Logic also knows that on a 1→0 transition of the "sprite active" signal, the fruit is either being cut or it hit the bottom of the screen, so it must show the "cut fruit" animation accordingly. the "xN" in the block diagram indicates that the number of SpriteLogic modules must be equal to the maximum number of sprites which will be on the screen any given time.
   Inputs: sprite[n]_active signal
   Outputs: sprite position and bitmap id to the SpriteVideoGen

9. **SpriteVideoGen**
   Takes in Sprite positions from SpriteLogic and bitmap data from the ZBT and generates video signals
   Inputs: sprinte[n] (x, y) and bitmap data
   Outputs: VGA video signal

10. **UI Generator**
    Generates video signal for UI, such as score, level, number of lives left, etc.
    Inputs: Collection of signals from GameLogic pertaining to user-relevant game state
    Outputs: VGA overlay appropriate for blending on top of the standard VGA output

11. **Video Logic**
    Combines video signals from sprites, UI and camera input. Performs alpha blending to sure that the UI is visible and clear to the user without obstructing gamepla
    Inputs: SpriteVideoGen, UIGenerator VGA signals
    Ouputs: final VGA signals

Project Plan

We will begin our basic project with no UI for the user, assuming the hands are point masses and the fruits are "blobs" which are square and fall down vertically from the top. The collosion will be based on simple overlap, not vectors, and the fruit will simply disappear when cut.

Once we have established this baseline functionality, we will iterate to add the following features:

- Give it a start screen where you have to cut a piece of fruit to start
- Give it a UI with score
- Add sprites rendered from ZBT bitmap
- Make it so it gets harder as the game goes on

- add physics so it simulates actual gravity, and nonlinearly moving sprites
- make the cutting result in fruit halves that obey conservation of momentum
- add music & SFX
- cheat mode with pictures of Gim and TA's
- Make collision based on hand vector motion, not position
- 'Game over' screen
- Possible 'bombs'

External Components

- Two brightly-colored gloves for our user (different colors)
- One NTSC camera (borrow from 6.111 TAs or purchase, perhaps collaboratively with other teams)

Division of Labor

Nathan: Game Logic, Sprite Logic
Isaac: Camera input, Camera to Cut detection
Drew: Sprite Video Generator, UI Generator, Video Logic, VGA Out