Motion Capture System on an FPGA

6.111 Final Project Proposal

Lauren Gresko
Elliott Williams
November 14, 2013

Overview

Motion capture, or recording and animating an object or person, is a commonly researched technology with a variety of applications. Motion capture is often employed in the fields of computer animation, video games, films, music, medicine, sports, robotics, and defense. For our final project, we would like to design and implement our own ‘mocap’ system that will capture a user's movements and animate them on the monitor. Our mocap system will capture position data accurately and display a simple animation in real-time. This simplified mocap system offers a robust solution to a commonly studied problem of replicating human motion. Our mocap system covers the basics of motion capture, while also offering the challenge of implementing a complex system on the 6.111 lab kit.

To create a 3D motion capture system, we will use two cameras that will face the user at 90 degree angles from each other. Using either colored patches or illuminated bracelets, we will track the user’s joints and generate a list of 3D coordinates that can then be used to reconstruct a skeletal model of the user. We will then use this model to generate a 3D model of the user’s movements on a computer screen. In the final product, the user will be able to move around their arms, and observe as the 3D image on the monitor mimics their motions.

For the minimal design we hope to track a user's arms and torso. This will require the tracking of four separate points. The generated 3D model will be a simple collection of rectangular prisms to represent the user's arms and torso. If we are able to quickly achieve the minimal design, we will improve upon it by tracking all of the user's body parts, for a total of eleven points (if we are clever, it might not be necessary to track 11 colors (an impossible task)). In addition we could improve the complexity of the displayed animation.

Design Overview

This document describes the proposed design of our motion capture system. The project is partitioned into two sections: the video system module and the graphics system module. The video system will process the video received from our two cameras into a useable format for our 3D graphics system. The 3D graphics system will then create a 3D model based on the information it received from the video system. Figure 1 depicts the overall block diagram of our motion capture system.

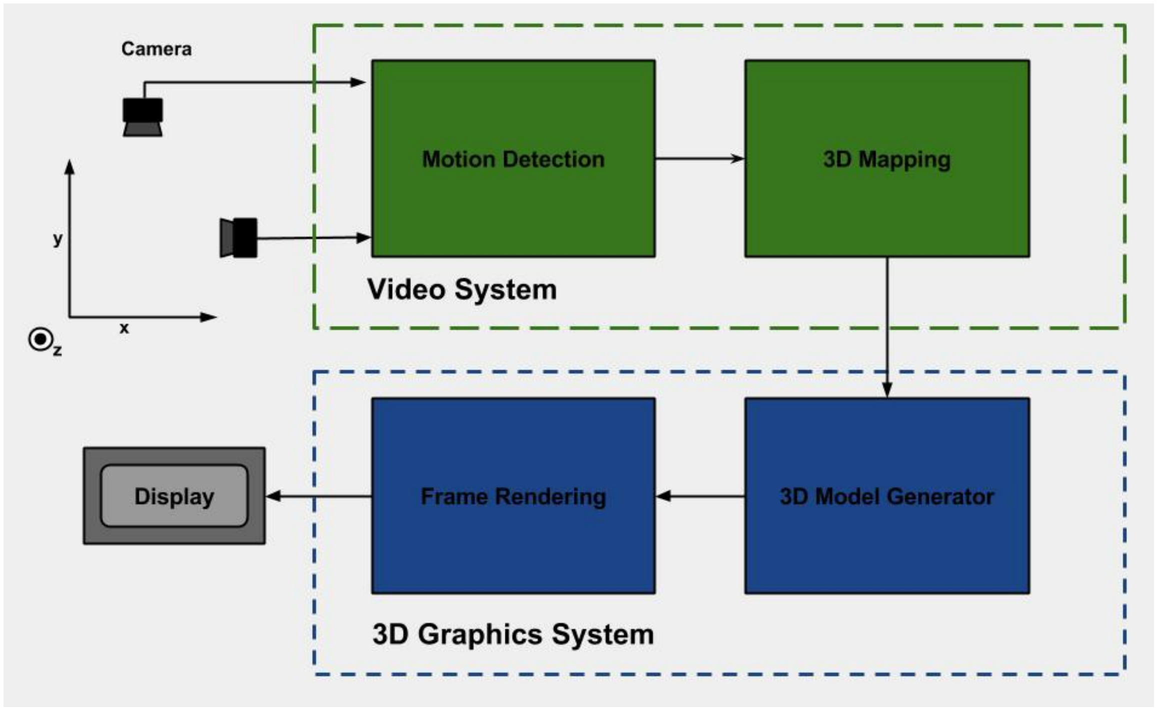

Figure 1: The Overall Block Diagram for the Motion Capture System. Above is a block diagram of the various modules that comprise the motion capture system. The Motion Capture System is composed of two blocks, the Video System (containing the Motion Detection and 3D Mapping subsystems) and the #D Graphics System (containing the 3D Model Generator and the Frame Rendering subsystems). The arrows represent the inputs and outputs of the modules. Additionally, the clock will connect to all of the blocks in the system; however for simplicity, the clock arrow connections have not been displayed.

1. <u>Video System</u>

The video system captures the location of the user's joints, generating a 3D skeleton to be displayed by the 3D graphics system. As shown in Figure 1, the functionality of the video system depends on two main modules, the motion detection module and the 3D mapping module. The motion detection system will track the colors sources on the user, therefore tracking the points and movement of the user’s arms. The 3D mapping module will process the resulting position data to determine the (x,y,z) skeletal coordinates of the user.

a. Motion Detection Module

The motion detection module will decode the video signals from both cameras to generate (x,z) and (y,z) coordinates for each of the four different color sources. As depicted in Figure 2, the user will be wearing four different color sources (terry cloth bands or LED bangle bracelets) at four different points on their body: the right wrist, right elbow, left wrist, and left elbow. The module will first detect the pixels that correspond with each color source. It will then perform a center of mass calculation on these pixels to determine the coordinates of the color source's center. This process will be completed twice, once for each camera, to get both the (x,z) and (y,z) coordinate pairs.

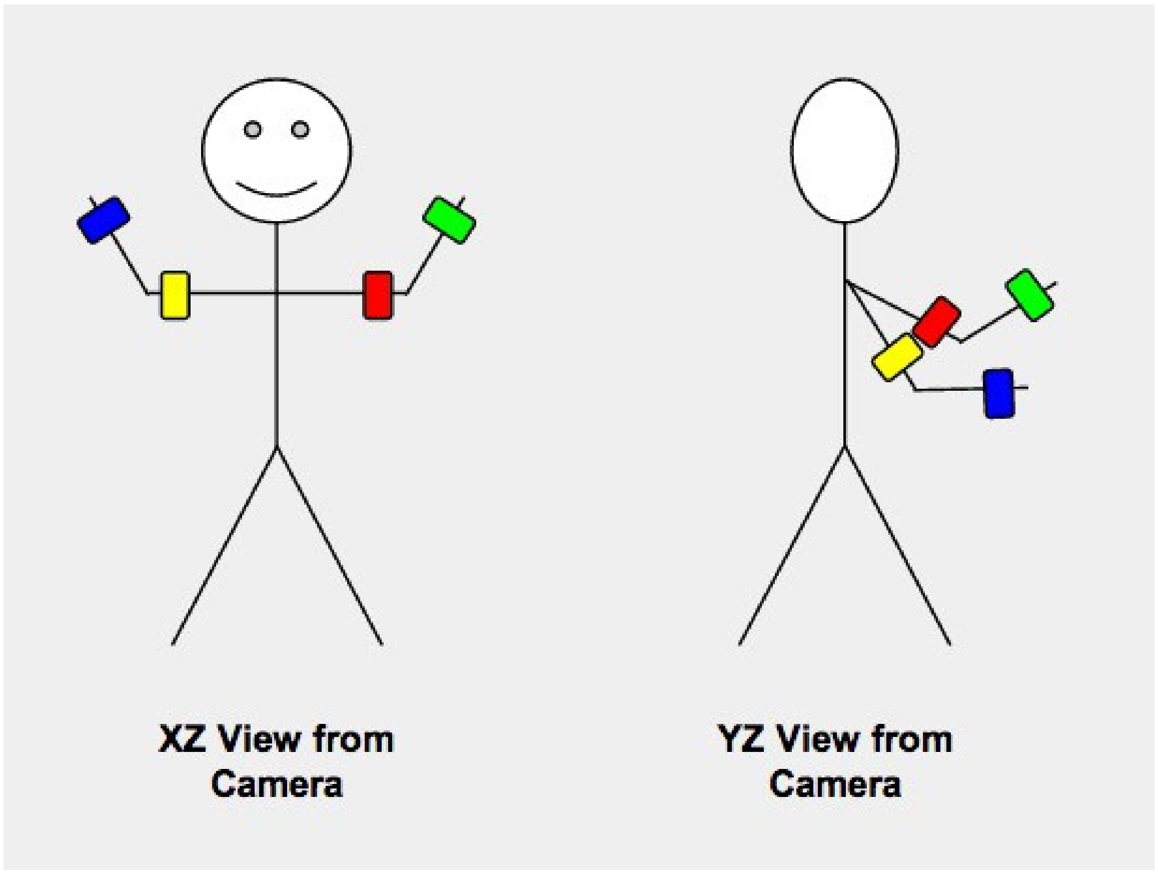

Figure 2: User input: four color sources. This diagram shows the four color sources (blue, yellow, red, and green) that the user of the motion capture system will be wearing, as viewed from the two camera inputs to the system. By placing color sources on the user's wrists and elbows, the motion detection module will track the endpoints of the user's forearm and upper arm in two coordinate planes (x,z) and (y,z).

The motion detection module is made up of four sub modules, the Video Decoder, RGB to HSV, Point Detection, and the Center of Mass Detection sub modules. The following section contains the details and processes for each of the sub modules. All of these modules will be designed by Lauren. These sub modules, along with the rest of the system's sub modules, are depicted in Figure 3 in the appendix.

Video Decoder:
The video decoder converts the NTSC and Ycrcb input video data to RGB video data. This step is necessary to allow further conversion into HSV video data. This module is provided by the 6.111 staff, and will be modified to output color video in RGB rather than the black and white This module will be used twice; it will be used on the (x, z) video data and the (y, z) video data.
Input: clock, video camera stream
Output: RGB bit stream

RGB to HSV:
The RGB to HSV converts video data using the RGB color space protocol into video data using the HSV color space protocol. This conversion is important because the HSV color space provides more space between color values, making subsequent point detection easier. This module will be used twice; it will be used on the (x, z) video data and the (y, z) video data. This module is also provided by the 6.111 staff.
Inputs: clock, RGB bit stream
Outputs: HSV bit stream

Point detection:
This module will take the HSV stream of bits and detect which pixels correspond to the color source, accumulating the correct color locations to enable the center of mass detection module to determine the location of each color source. This detection will be done by comparing the hue of the HSV bit stream to the color sources that the user would be wearing. In this project, this module will instantiated 4 times; a parameter will determine which of the 4 colors the instance is looking to match and therefore which of the 4 points (R wrist, R elbow, L wrist, L elbow). This module will be used twice; it will be used on the (x, z) video data and the (y, z) video data.
Parameter: Source Color
Inputs: clock, HSV bit stream
Outputs: matched pixels

Center of Mass Detection:
This module will determine the center of mass of the user’s arm joints. This center of mass determines the coordinate location of each color source in the (x,y) and (y,z) planes. One way would be averaging the position of all the matching pixels; or for a= x or y or z coordinate, the module could compute a*intensity(a) divided by sum(intensity(a)). Additionally, this module will keep track of the velocity, which will be used in future modules. In order to do this, the module will also keep track of the previous location of the center of mass and compare the change in position for each frame. This module will be used twice to calculate both (x, z, color) and (y, z, color) coordinates.
Inputs: clock, matched pixels
Outputs: (x, z, color) or (y, z, color) which represent the coordinates of the designated body part (the L wrist, L elbow, R wrist, or R elbow).

b. 3D Mapping Module

The 3D mapping module will synthesize the coordinate data generated by the motion detection module into a group of 3D coordinate pairs that comprise the user's joints. The 3D mapping module will first map the (x,z) and (y,z) coordinates of the four color sources together based on the color and z dimension of the points. The module will then match the (x,y,z) points of the right wrist to the (x,y,z) of the right elbow, and the (x,y,z) points of the left wrist to those of the left elbow. This matching will result in a forearm “bone” that will then be passed to the graphics module to generate a real-time 3D model of the user’s arms.

The 3D mapping module is made up of two sub modules, the 3D Coordinate Generator and the Skeleton Generator. The following section contains the details and processes for each of the sub modules. All of these modules will be designed by Lauren. These sub modules, along with the rest of the system's sub modules, are depicted in Figure 3 in the appendix.

3D coordinate Generator:
This module will create the 3-dimensional (x,y,z) coordinates of each point from the (x,z, color) and (y,z, color) coordinates, thus detecting the locations of all of the user's joints. This module also must handle hidden points, i.e. when a color source is hidden from one or both of the cameras. As described below in the Testing section, this module will be first implemented as a software prototype.
Input: clock, (x , z , color) and (y, z, color) coordinates
Output: (x,y,z, color) coordinates

Skeleton Generator:
This module will match the (x,y,z) coordinates of the left wrist with the left elbow and right wrist with the right elbow. These matched coordinates will be outputted as “bones” to be drawn, providing the 3D graphics system with a 3D skeleton of the user's location. This will also be developed originally as a software prototype.
Input: clock, (x,y,z) coordinates
Output: matched elbow and wrist (x, y, z), (x,y,z) coordinates

2. 3D Graphics System

The 3D graphics system generates and displays a 3D model of the user based on the 3D skeleton generated by the video system. As shown in Figure 1, the graphics system is also divided into two parts, the 3D Model Generator and the Frame Rendering module. The 3D Model Generator uses the skeleton generated by the video system to create a 3D model of the user when viewed from a predetermined viewpoint. This viewpoint will be controllable by the user. The frame renderer draws this model to the display.

a. 3D Model Generation Module

The 3D model generation module creates and manages a 3D model of the viewer. It first generates a collection of rectangular prisms, each representing a segment of the user, from the "bones" supplied by the video system. This module then passes the prisms through a series of linear transformations that determine how the prism would appear from a certain viewpoint. The user will be able to use the button inputs on the FPGA (labeled Camera Control Input in Figures 3) to move the camera and view the model, and thus their own movements, from a different angle. To provide a greater sense of depth, the model will be lit by a simple uniform light source and shaded appropriately. This module will be pipelined to reduce the number of necessary signal wires and to enable real-time video playback.

The 3D model generation module is made up of t into six different sub modules, the Rectangular Prism Generator, the Normal Vector Calculator, the Shader, the Camera Controller, the Transform Matrix Generator, and the View and Projection Transformer. The following section contains the details and processes for each of the sub modules. All of these modules will be designed by Elliott. These sub modules, along with the rest of the system's sub modules, are depicted in Figure 3 in the appendix.

<u>Rectangular Prism Generator</u>:
This module will generate a set of eight vertexes (x,y,z) that compose a rectangular prism centered around each provided “bone”. These prisms represent the user's arm segments and are the objects on which the rest of the 3D graphics system will operate on. This will be developed originally as a software prototype.
Input: clock, matched elbow and wrist (x, y, z), (x,y,z) coordinates
Output: the eight vertexes (x,y,z) of a rectangular prism in standard coordinates

<u>Camera Controller</u>:
This module controls the location and orientation of the viewing camera by saving a set of coordinates in a state in memory. These coordinates consist of a camera location (x,y,z), the point that the camera is facing (a,b,c), and a unit vector describing which direction is up from the camera's perspective. These values will be able to be updated by using button inputs, thus the camera will be able to be moved, rotated, and rolled. This will be developed originally as a software prototype.
Input: clock, camera control input
Output: camera location (x,y,z), focus point (a,b,c), and orientation vector (i,j,k)

<u>Transform Matrix Generator</u>:
This module uses the state of the camera to create the transformation matrix needed to orient objects with respect to the camera and perform the necessary perspective transformations. This will be developed originally as a software prototype.
Input: clock, camera coordinates
Output: view and projection transform matrix

<u>View and Projection Transformer</u>:
This module performs the matrix multiplication necessary to transform standard coordinates based around the origin into coordinates based around the camera's field of view. This transformation includes perspective effects such as foreshortening. This will be developed originally as a software prototype.
Input: clock, view and projection transform matrix, the eight vertexes (x,y,z) of a rectangular prism in standard coordinates
Output: the eight vertexes (x,y,z) of a rectangular prism in view coordinates

<u>Normal Vector Calculator</u>:

This module calculates the normal vector of each surface on the rectangular prism created by the Rectangular Prism Generation module. These normal vectors are critical in the proper shading of the arm segments. This will be developed originally as a software prototype.
Input: clock, the eight vertexes (x,y,z) of a rectangular prism in standard coordinates
Output: the unit normal vectors of the rectangular prism's six faces

Shader:
This module calculates the appropriate ambient and directed light incident on each face of the rectangular prism. It then uses this light intensity to create the proper shade of color for each face. This shading will create a much greater sense of depth in the final image.
Input: clock, the six unit normal vectors of the rectangular prism's six faces
Output: the shaded colors of the rectangular prism's six faces

b. Frame Renderer Module

The Frame Rendering module takes the 3D model and shaded color values and displays a properly scaled 2D image of the 3D model on the VGA screen. The module uses a double frame buffer in ZBT memory so it can update one frame while displaying another. If the rest of the system is fast enough, this buffer will enable real time graphics. Like the 3D model generator module, this module will be pipelined to reduce the number of necessary signal wires and to enable real-time video playback.

The Frame Renderer module is divided into four sub modules, the View-port Transformer, the Renderer, the Frame Buffer, and the VGA controller. The following section contains the details and processes for each of the sub modules. All of these modules will be designed by Elliott. These sub modules, along with the rest of the system's sub modules, are depicted in Figure 3 in the appendix.

Viewport Transformer:
This module performs the matrix multiplication necessary to transform 3D camera coordinates into 2D pixel coordinates. This process provides the rest of the frame renderer module with the proper polygons to draw to the screen. This will be developed originally as a software prototype.
Input: clock, the eight vertexes (x,y,z) of a rectangular prism in view coordinates
Output: the eight vertexes (x,y,z) of a rectangular prism in pixel coordinates

Renderer:
This module draws the provided prisms into the frame buffer and fills in the prism's sides with the appropriate color. This module also communicates with the frame buffer to determine whether parts of the prism it is drawing is obscured by other prisms already drawn. It does this by comparing the received depth coordinate, z, at location (x,y) with the depth coordinate at that location in memory. This will be developed originally as a software prototype.
Input: clock, the eight vertexes (x,y,z) of a rectangular prism in pixel coordinates, the shaded colors of the rectangular prism's six faces, z_read
Output: location, z_write, color

Frame Buffer:

This module manages the double frame buffer and depth buffer stored in ZBT memory. It allows the renderer to write into one buffer while the VGA controller reads from the other buffer. When both processes are done, the frame buffer switches the two buffers, enabling constant frame drawing and displaying.
Input: clock, read location, color, z read,
Output: write location, z_write, color_write, z_read, read location, color_read

<u>VGA Controller</u>:
This module manages the generation of the proper signals to control the VGA monitor. It communicates with the frame buffer module to read and display the proper pixel values.
Input: clock, color
Output: read location, VGA control signals

Testing

Video Module Testing

To test the Motion Detection module, a debugging module will be created that displays both the video feed and the center of masses of the detected points. Using this module, we will be able to check whether we are successfully converting from HSV to (x,z) and (y,z) coordinates as well as correctly locating the correct center of mass of the color sources.

The 3D Mapping module will first be developed as a software prototype in Python. Once a working algorithm has been developed for matching the (x,z) coordinates to the (y,z) coordinates and then matching the correct side wrist (x,y,z) to elbow (x,y,z) the module will be implemented in Verliog. Also in this software prototype, we hope to figure out the best way to handle "missing coordinates," i.e. when one of the color sources is hidden from one or both of the cameras. Developing and debugging this tricky part of the project in software first will save time, because we will not have to recompile after each adjustment we make.

Graphics Module Testing

The majority of the 3D graphics generation system will first be developed as a software prototype in Python. This will enable the complicated math issues involved with coordinate transforms to be tested in a fast development environment. This prototype will also enable predetermined matrix values (like those needed in the viewport transformation module) to be tweaked without having to recompile everything. This system will be tested first by creating a series of mathematical tests that will ensure the matrix, normalization, and shading math is being calculated properly. Then a series of predetermined skeleton coordinate values can be fed into the graphics module and examined from multiple view angles to catch bugs. The exact same tests can be used for the software prototype and hardware implementation. Modelsim will be used for hardware math testing while the complete graphics test will be performed on the FPGA itself.

The only graphics modules that will not be tested in python are the Renderer, Frame Buffer, and VGA controller modules. These will not be tested in Python because they are highly dependent on the ZBT and VGA interfaces. Instead these modules will be tested in Modelsim using predetermined shapes. The memory stored in the buffer can be examined to ensure that the proper data was written in the proper location and the result could be displayed to the screen once the VGA module is verified. The VGA module will be tested by reading predetermined frames from memory and displaying them to the screen.

Once both major systems are complete, they will be tested together, first with still subjects where all markers are seen, then with subjects with hidden markers, then with subjects performing slow simple motions, and finally with fast moving subjects performing complex motions.

Schedule

10/27-11/1
Final Project Meeting & Proposal
Order Parts

11/3-11/8
Block Diagram Meeting
Edit module from staff (Video Decoder Module and Ycrcb to RGB)
Begin Point Detect and Center of Mass Module
Complete Graphics Software Prototype

11/10-11/15
Design Presentation
Complete Point and Center of Mass Detection Module
Begin 3D Coordinate Generator
Complete 3D Model Generator Module

11/17-11/22
Project Checklist
Complete 3D Coordinate Generator
Begin Skeleton Generator
Complete Frame Rendering Module

11/24-11/27
Project Status Update
Finish Skeleton Generator
Integrate Modules together before leaving for Thanksgiving

12/2-12/6&12/8
Final Week of Debugging

12/9
Final Project Check off

Appendix

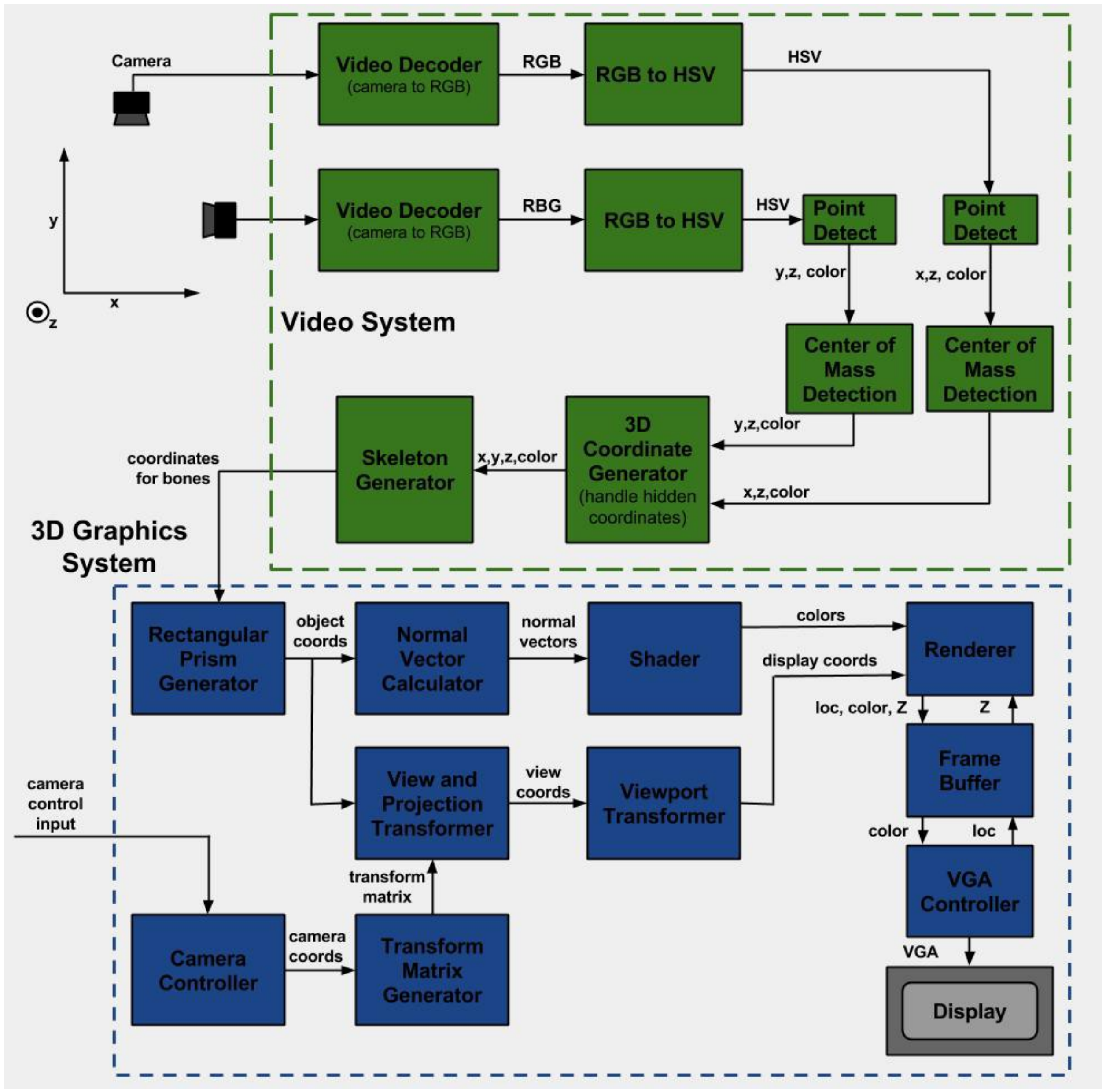

Figure 3: The Modules and Sub modules of the Motion Capture System. Above is a block diagram of the various modules that comprise the motion capture system. The two main systems are the Video system and the 3D Graphics system, and each of these systems has several sub modules represented by the blocks in the diagram. The arrows represent the inputs and outputs of the modules. Additionally, the clock will connect to all of the blocks in the system; however for simplicity, the arrow connections have not been displayed. Modules in the Video System will be implemented by Lauren, while modules in the 3D Graphics System will be implemented by Elliott.