COMP3419 Assignment Project
Intelligent Animation

James Koungoulos

May 2017

1 Introduction

This report discusses the implementation of variety of video processing, compositing and 2D animation techniques. These features were implemented using Java and the Processing sketchbook framework.

The program demonstrates several features:

- Motion Capture
- Animated Background
- Marionette Creation
- Intelligent Objects
- Dynamic Sound

The end result is a dynamic video combining a variety of graphics and processed video frames. This program demonstrates a game-like experience, with a marionette character ("One Punch Man") using his fist to fight off moving enemies.

2 Implementation

In this section, the implementation of each aspect of the program will be discussed in detail.

2.1 Motion Capture

In this project, a source video was given as input from which motion was captured. This source video displayed a monkey puppet with five red markers; one for the monkey's chest, and one for each foot and hand. The end goal was to isolate these markers from each frame of the provided video and capture their

motion in a data structure.

First, the image was thresholded to identify the red 'marker' pixels in each frame of the video. This was achieved by comparing a combination of HSB and RGB values from each pixel against a range of values. These values were picked by analyzing the optimal range of values for acceptable red pixels in each frame. In doing this, non-red pixels were excluded.

Next, the thresholded frame had several morphological operations performed on it to eliminate noise and enhance the homogeneity of each marker. These morphological operations were erosion and dilation. The image was eroded twice, then dilated five times to produce an image with clearly segmented markers.

The next step was to classify each pixel as being from one of the following marker groups:

1. Chest
2. Left Hand
3. Right Hand
4. Left Foot
5. Right Foot

Initially, a grassfire algorithm was used to create an undefined number of clusters of red pixels. This algorithm would create a cluster for each group of connected pixels - that is, red pixels that are adjacent to each other in the frame. The 5 largest clusters were used to calculate the limb markers. This technique generally resulted in more accurate marker identification, but struggled overall due to dependence on every frame clearly displaying all of the markers. In addition to this, the grassfire method often merged the two feet markers when they crossed near each other. In order to counter such inaccuracies, data from the previous frame was used as a substitute. However, this led to very rough and inaccurate motion.

Due to its issues, this algorithm was scrapped and replaced with the quadrant algorithm. This algorithm operated by synthesizing the centre marker from the midpoint of all identified red pixels. From there, each red pixel was classified by its position relative to this midpoint.

For example, if a pixel has a lower x-coordinate and higher y-coordinate compared to the centre marker, it is placed in the bottom left quadrant. Each quadrant is represented by an ArrayList data structure containing coordinate Integer arrays.

The nature of each quadrant is intuitive - the top left quadrant contains pixels

correlating to the left hand, the top right quadrant contains pixels correlating to the right hand, the bottom left quadrant contains pixels correlating to the left foot, and the bottom right quadrant contains pixels correlating to the right foot.

After every pixel has been classified, coordinates for each marker are derived by calculating the midpoint of the pixels in each cluster. An issue with this algorithm is that a marker's motion cannot be captured if it crosses over into another marker's quadrant. To correct this, the algorithm checks whether any of the marker data structures are empty. If they are, their coordinates are estimated based off of the quadrant they have crossed into.

This algorithm guarantees five marker coordinates depicting the character's motion for **every** frame - even those in which the chest marker is occluded.

2.2 Animated Background

This feature is very simple. With every draw call, the next frame from a video is loaded as the background image of the program. This video loops until the marionette video is complete.

2.3 Marionette Creation

With the limb coordinates provided by Motion Capture, creating a dynamic marionette graphic is straightforward. Since we already know what limb each coordinate pair corresponds to, we draw the relevant pre-loaded limb image for each coordinate value at each draw step. Then, a yellow line representing the arm/leg of each limb is drawn from the image to a screen position offset from the centre coordinate for optimal cosmetic effect.

A head image is projected above the chest coordinate, and each limb image is loaded at a position offset from the marker coordinates to better represent the distinct limbs of the marionette. In this case, the character "One Punch Man" is created using the motion capture points derived from the discussed video processing techniques.

2.4 Intelligent Objects

Multiple types of intelligent objects are used in this project. The first intelligent object is the character's Fist - this is represented by a graphic. This Fist constantly orbits the marionette, moving in a clockwise fashion.

The Fist's position is dependent on the marionette's centre marker. As well as this, the Fist slowly expands and retracts its arc around the marionette according to a specified range.

The Fist interacts with another type of intelligent object in the scene - the Alien.

Alien objects are periodically spawned at random locations on the screen. Up to 25 Aliens are allowed to occupy the screen at a time. Each Alien moves according to a parametric equation (considering radius of movement and current time passed). This makes it more difficult for the Fist to hit each Alien.

With every draw call, collision between the Fist and any Aliens is calculated. This is performed with simple rectangle collision detection. If the Fist hits an Alien, it is destroyed. Similarly, if an Alien hits the marionette, a stylized hit marker is displayed in response. At the bottom of the screen the current score and character health is displayed. Every time the Fist hits and destroys an Alien, the score is incremented by 50. Every time an Alien hits the marionette, the character's health is decreased by 1.

2.5 Dynamic Sound

The Beads and Sound libraries in Processing were used to implement a variety of sound features.

As the Fist object orbits the marionette, two WavePlayers objects use its x and y coordinates respectively to alter the frequency of an audio stream that plays over the output video. Bead's default SINE buffer wave data is used to mimic a swooping sound - as the Fist's arc grows, the sound grows in volume. Gain is defined to avoid clipping, and a low volume is used so as not to overshadow the music soundtrack.

A background song is played for the duration of the video. This is achieved via Processing's SoundFile object. An mp3 file is loaded from the sketch's folder and looped until the video ends.

Finally, with every collision, a punch sound effect plays. That is, whenever the Fist object strikes an Alien, or when an Alien strikes the marionette, a SoundFile object is played.

3 Conclusion

In combining a variety of video processing, compositing and 2D animation techniques, a robust multimedia project was developed. First, motion was captured from a video file via image segmentation, and represented in the form of five coordinates - one for each limb, and one representing the centre of the marionette.

This data was then used to create a dynamic marionette character, composited with an animated background. As the marionette executes captured movements, intelligent objects interact with it and each other. A Fist object orbits the marionette, destroying Alien objects, which in turn create animated collisions with the marionette. A scoreboard is displayed, keeping track of the

interactions between objects. Sounds are played to express actions - the Fist makes a swooping noise as it orbits, and each collision makes a punching sound.

Despite these achievements, there is much room for improvement. Most pertinently, the motion capture algorithm is still lacking in accuracy and smoothness. Movement transitions are often rough and jagged. A more accurate clustering algorithm would result in a far better approximation of the source marionette's motion. Alternatively, motion could be captured via different strategies like Optical Flow or Boundary Detection. In addition to this, blending techniques could be implemented to improve transitional smoothness even more. Finally, a more cohesive rendering of the marionette would further improve the quality of this project.