Coursework 2

Audio-Visual Computing Final Project

Title of Work: Virtualoso

Names:
- Amrish Parmar, *Student No: 33338726*
- Charlie Ringer, *Student No: 3338136301*

EXECUTIVE SUMMARY

We have designed an interactive instrument application that allows the user to create sounds by interacting with on-screen elements. There are two main parts to the application: a drum kit and a theremin. Using colour tracking techniques behind the scenes, the user can use any pair of objects to "hit" on-screen drums and cymbals which will play the corresponding sound associated with them. In the theremin section the colour trackers can be used to independently control the amplitude and frequency of synthesised sound.

FEATURES AND USER INTERACTIONS

We were interested in writing software that incorporated tracking techniques via a webcam and then applying it in a useful way. We therefore came up with the concept of writing an application that allows you to play virtual instruments. We chose a drum kit because it has a very physical method of playing which translates to tracking via a camera very well. Looking to expand our application and thinking about what other instruments have a physical method of playing and we realised that a theremin is perfect as you can mimic the movements of a playing a real theremin accurately with a camera. When looking at tracking options we settled on colour tracking over motion tracking because we needed to track individual points on the screen rather than an overall motion. For example, we did not want head movement to interfere with the program.

This application is split into three main areas. When you first load the application you are greeted with a main menu screen. On this screen you are presented with the option to select either the theremin or drum kit application. Once one of the options is selected ("Play the Drums" or "Play the Theremin") the user's webcam output is displayed. At this point the user is instructed to set up their colour tracking object by following the on-screen instructions. Once they have completed the setup process they are free to use the chosen instrument.

If the user has chosen to use the drum kit, the endpoints of their colour tracked objects will function as an analogue of the tips of a pair of drumsticks. There are a number of different drum- and cymbal-shaped objects drawn on screen. When the colour tracked object passes over one of them it will produce the corresponding noise, e.g. crash cymbal, snare drum, hi-hat, etc.

If they have chosen to use the theremin, then the two colours we are tracking will act as amplitude and frequency controls. The user can then move whatever objects they are tracking around the screen to change these values. If the user moves the tracker corresponding to frequency up or down, the program will raise or lower the pitch of the note being played. If the user moves the tracker corresponding to amplitude left or right, the program will raise or lower the volume of the note being played.

At any time whilst in one of the instrument sections, the user has the option of clicking on the "Back" button which is displayed in the top-left corner. This will take the user back to the main menu, where they can then choose to open the other instrument if they wish.

CODEBASE

The application uses two states in order to work out what instrument we are playing, the drum or theremin. It also uses two states to determine if we are in the main menu or currently playing one of the instruments. We use final ints to define what each state corresponds to and then use a tree of branching if/else statements on the `state` and `app` variables to determine what functions and methods from the relevant classes should be called.

The `ColourTracker` class is used for the program objects that track the user's real-world objects so we know where they are on screen. It has a colour variable for the tracked colour, a PVector for storing the current position on screen (as well as one for storing it's previous position), a char that stores the key to use to update the colour for the tracker and a boolean to store whether or not it is set up. The main function of the class is the `updatePosition()` method. This is used for working out if the tracker has moved and to where it has moved. In this function we loop through every pixel of each frame of the camera's output and extract the pixel's RGB components. We compare these to the RGB components of the colour we are tracking. If they are close, we store a PVector for the location in an ArrayList. Once we have worked out all of the matches, we average the location of these vectors then set this as the new position. Averaging the position in this way prevents the tracker from jumping around the screen when we have a large single colour object. The colour tracker code also has a method for drawing a marker to the screen (an ellipse) as a visual indicator for the user as to the tracker's current position on screen. Finally, we have a method for updating the colour when the appropriate key is pressed (based on the mouse location at that time).

The `DrumKit` class contains a set of drums and a set of cymbals. In the constructor we create these as arrays of objects. There is a method to draw the drum kit which loops through each of the drums and cymbals and calls their draw methods. The last method, `checkForHit()`, loops through the drum and cymbals arrays and calls the hit detection method for each object. There is an `AttackSurface` class which is extended to create the `Drum` and `Cymbal` classes. The `AttackSurface` class handles collision detection for seeing if the colour tracker has hit a drum/cymbal. This works by getting a line across the drum and a line that tracks where the tracker moved (using it's position and it's previous position). We then perform a line intersection calculation on them to work out where the lines meet and whether that point is along both lines; if it is we have a hit the drum/cymbal and we play the sound associated with it. The drum and cymbal classes have their own methods for drawing the objects as the shapes are different. These are created using the built in Processing functions for drawing geometric shapes to the screen.

The code for the theremin portion of the program is mostly contained within the `Theremin` class. We make use of the `UGen` elements of the Minim library for generating audio. We first generate a sine wave by instantiating a sine wave `Oscil` object with an initial frequency of 440 Hz and an amplitude of 0.5 (which represents volume mapped between 0 and 1) and then patch the audio to Minim's output. The class contains one method, `updateSound()`, which is called every frame that changes the frequency and amplitude of the output audio based on the position of the two colour trackers (which are passed in as function parameters). The y position of the frequency tracker is mapped from the height of the window to 0 into a range of 200 to 1200. The x position of the amplitude tracker is mapped from 0 to the width of the window into a range of 0 to 1. Using the setter methods from the `Oscil` class we then update the sound's frequency and amplitude. As there is the possibility that one of the colour trackers can return a position of `NaN` if a colour is temporarily not found, there is some error checking code at the start of the method that ensures that we only use the returned value if it is greater than 0, otherwise we use a stored value from when the position was last valid.

REFERENCES

Processing video library and capture code,
https://processing.org/reference/libraries/video/Capture.html

Colour tracking sample code,
http://www.learningprocessing.com/examples/chapter-16/example-16-11/

Minim audio library and audio generation sample code,
http://code.compartmental.net/minim/audiooutput_class_audiooutput.html

Menu background image,
http://pixabay.com/en/music-note-twisted-staff-stave-306008/

Menu font,
http://www.dafont.com/chopin-script.font