OPTIC MIXTAPE

The Audio-visual Projection Experience

COS426 Final Project, Spring 2019

Eric Hayes (*eshayes*)

Charmaine Chan (*cwkchan*)

Nina He (*ninahe*)

Introduction

Goal

We designed and programmed an audiovisual projection experience for our final project, in which various inputs of music determined how preset visuals were to be distorted. These visuals were then projected onto a 3-dimensional structure in a way that wrapped the videos onto the individual faces of the structure. The primary goal was to create a working assemblage of pieces: video distortion, music inputs, and projections. Our elevated goal was to create something visually interesting, complex, and dynamic. This type of audiovisual projection could be used to visualize music at concerts, create outdoor experiences by projecting onto buildings or other large surfaces, and this work could benefit artists looking to explore new ways of creating content.

Previous Work

For work related to projection of a 2D image or video into 3D space, this can be seen in amusement parks or art museums. For example, in Disney World there is a light/firework/laser show that involves projections onto the castle. Using projectors, lasers, and lights, they are able to project certain videos and animations onto 3D surfaces so that it looks like there are patterns or displays on the 3D castle. Because the projector projects from a certain point, transformations need to be done to the video so that when projected onto a non-parallel or flat surface, the video doesn't look morphed. As for work related to using music to transform images, there are a number of audio visualization works, often using particle movement and simple imagery that responds to musical cues. However, we generally have not encountered anything that combines all of these components- video distortion, audio inputs, and projection- as we have done with this project.

Approach + Methodology

Our general approach was to split up the music processing, the image distortion for projection, and the video filters into modular pieces that we could later assemble. This also allowed us to divide the work between the three of us. For the purposes of processing music in real-time and manipulating video with many frames per second, we wanted a solution that processed the data efficiently and with relative ease. We chose Max/MSP/Jitter as our toolset for signal processing and JavaScript for image warping and filtering in a similar way to the assignments. Max allowed us to load, process, play, and manipulate music efficiently and generate parameters that we could use as inputs in our image filters. It also provided us with a convenient way to play our videos and pass the frames to our JavaScript image filters one at a time. In JavaScript, we used the Jitter API and leveraged the concepts we learned in class to create a projection warp filter and creative image processing filters.

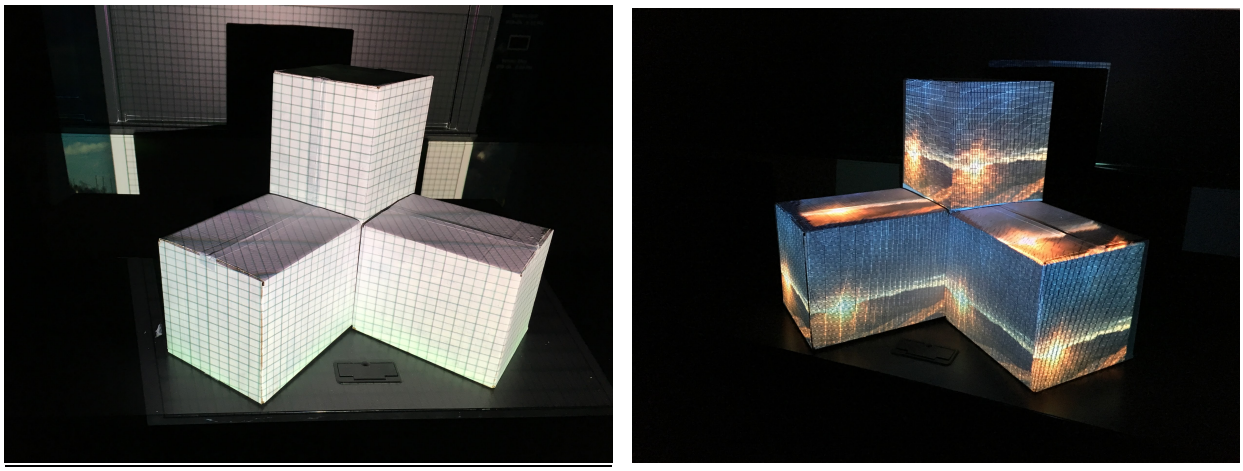
<u>Projection</u>

For the transformation of the 2D video for 3D projection, we had to create a function that could take in an image, coordinates of the rectangle or square within that image that was the "source" image of the projection, and the destination coordinates that the source image should be warped to.

For this piece, there were several possible implementations. The first implementation we attempted was by calculating the height and width ratios of the destination x and y coordinate (or pixel) and reverse mapping it to get the source x and y pixel. However, when we implemented this, we found that the warping wasn't accurately creating warped images and the calculations for the height and width ratios for each point was extensive/ confusing.

The other approach which we found online involved creating matrices. In this approach, we found the transform matrix from 4 projected points. Taking a source image, and destination coordinates for the corners, we could create a transform matrix that would help us achieve forward mapping. We could select a pixel from the source image and find the x and y coordinates that the pixel should fill in the destination image. Computing the projective transformation involved creating matrices with the homogenous coordinates (adding one more dimension to the last entry) and scaling the matrix by computed coefficients, for the source coordinates (we can call this matrix A) and the destination coordinates (we can call this matrix B). These matrices help map from basis vectors to source positions and from basis vectors to destination positions. Then we compute the combined matrix C which is the product of B and the inverse of A. To find the corresponding location for a pixel in the source image, we just multiply a vector of the source image coordinates by C. More information about the matrix calculation can be found in our sources.

We chose to use this implementation because it created the most accurate projections. However, the disadvantage of this implementation is that is does forward mapping and therefore occasionally produces empty or black pixels when we don't want it to. It also can make the sampling less efficient than it could be. With more time, we could calculate the matrix for reverse mapping. However, because of time constraints, we decided to focus on other components of this project instead.

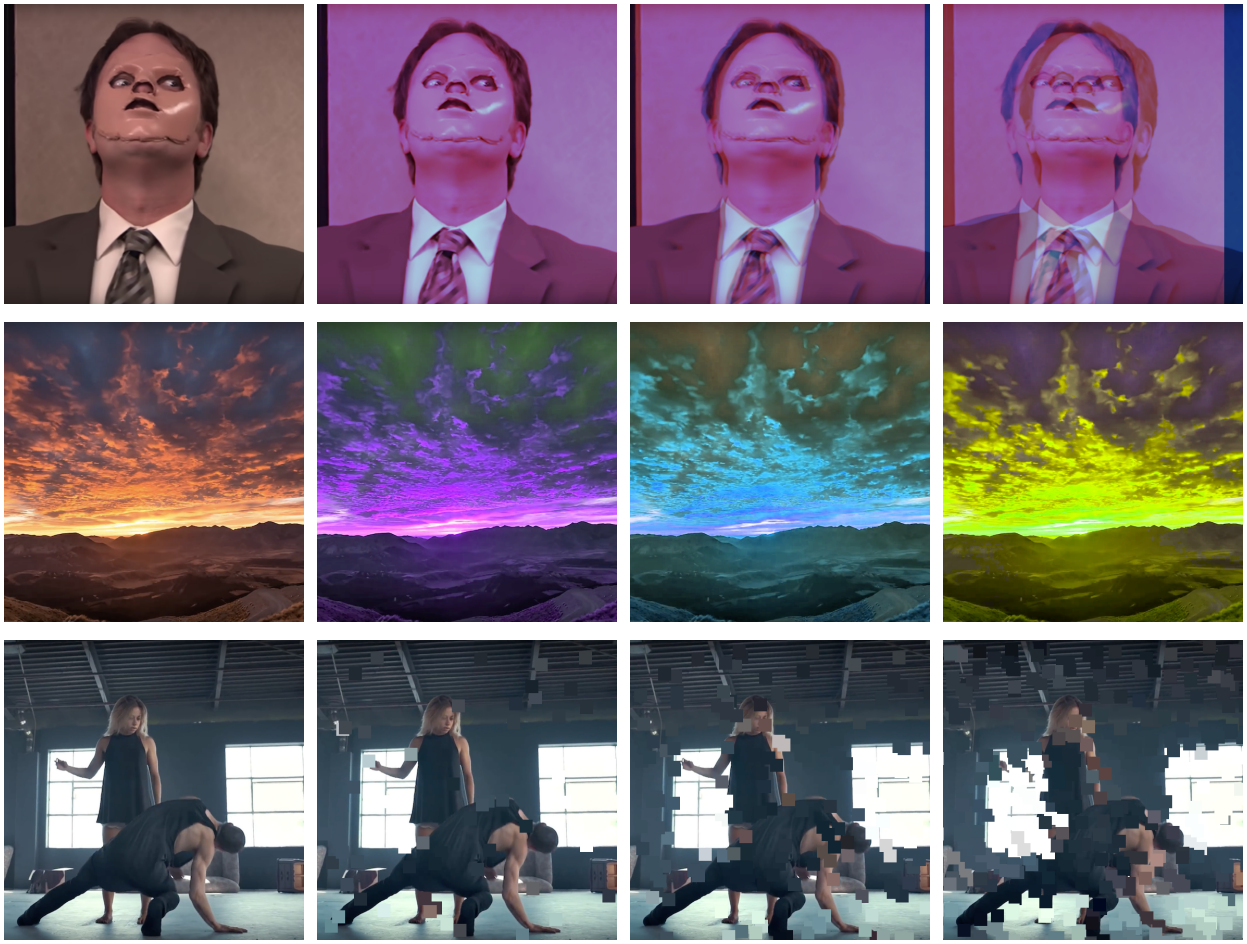


Left: Grid projected normally without any distortion (note the extreme vertical stretch on the top faces). Right: Projection-mapped images.

Image distortion + filters

The implementation of the video distortion and video filters followed the structure of Assignment 0 and Assignment 1 closely; we essentially treated the video as a series of image frames, and each frame was processed through a customFilter. The numerical parameter we passed to our function was taken from the musical component and normalized to be between 0 and 1; this value then dictated the level of visual

distortion shown in the projection. We used JavaScript for the filters and image manipulation techniques very similar to the class assignments, with the exception of some syntactic changes that we had to make to adhere to the Max's available JavaScript libraries. Generally, to keep the performance at a relatively high rate, the filters had to be rather simple. Thus, we were restricted to only using the RGB colorspace and basic pixel manipulation, due to the limitations performance and of the Max software.

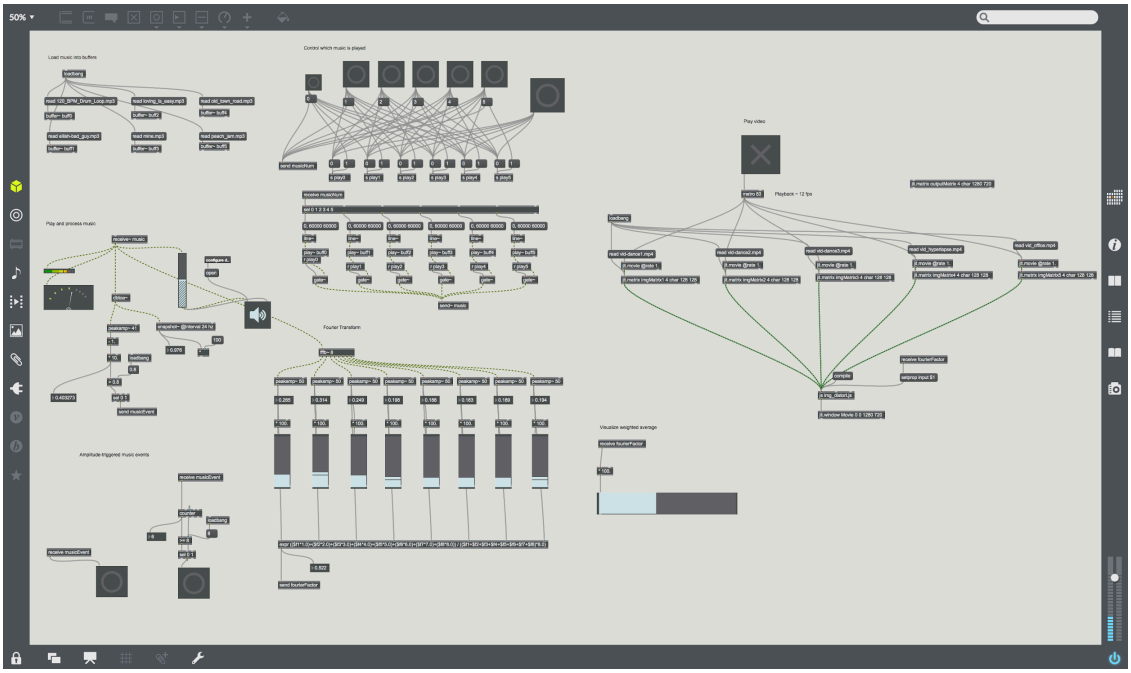


Some examples of filtered still video frames with varying input values.

Music & Video Signal Processing in Max

Conveniently, JavaScript code can be run directly in Max with inputs passed to the js node object. This allowed us to leverage the signal processing capabilities of Max alongside the image-manipulation techniques we learned in class.

In our Max project we loaded 6 pieces of music into 6 different audio buffers, which could then be played on command. Similarly, we loaded 5 128x128 videos which were then fed into 5 different Jitter Matrix objects one frame at a time at 12 fps. These image matrices were directly manipulated in our JavaScript code using the Jitter JS API. We considered copying the image data from Max into the Image data structure used in class, but in the interest of efficiency we opted to manipulate the Jitter Matrices directly.



Overview of our project in Max, a node-based tool for audio/video signal processing.

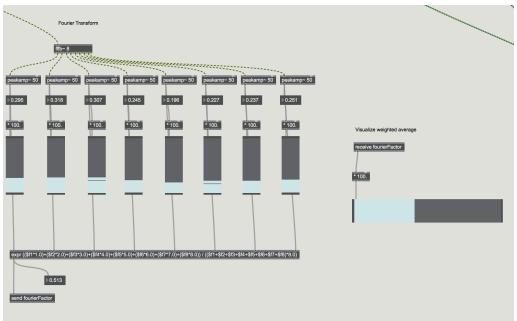
From the audio, we generated 2 main inputs for video processing:

1. Music events

A music event was triggered when the amplitude passed a certain threshold. This was determined by logging the peak amplitude over a 41ms (~24 Hz) period and checking to see if it was above our threshold. Due to the high frequency of events triggered in songs with a high BPM, we chose to only act upon every eighth music event. These events were used to toggle large macroscopic changes, such as which image filter was being applied.

2. Fourier Transform

We generated a more continuous parameter for image processing using the Fourier Transform. Specifically, we filtered the audio into 8 frequency bands (by processing 512 samples at a time) and took a weighted average of the magnitudes of these frequency bands. Then we normalized this weighted average to obtain a float between 0.0 and 1.0, which changes according to the distribution of frequencies in the music.



Fourier Transform allowed us to filter specific frequency bands

Results + Discussion

The success of our results could then further be measured in the quality of our projections, whether that be in regards to the visual aesthetic, the speed of the projections, or the quality of the image mapping/sampling. Our experimentation involved using a variety of video and music types to generate different projections with different visual qualities. We were ultimately able to accomplish the main goal of our project, but the results also indicated the many areas for potential improvement and optimization.

Follow-up work:

- Improve speed and overall performance efficiency. This would allow for a faster frame rate, quicker image processing, etc.
- We measured the points on our projection surface manually. This process could be automated with a camera and computer vision.
- Projecting onto more elaborate surfaces. By mapping a grid projected onto a more complex surface, we could use a technique similar to what we did in the image warp assignment.
- More creative image/video processing filters.
- Add interactivity with different musical inputs.
- The projection function could be adapted to use reverse mapping instead of forward mapping to avoid the black pixelation in the projection.
- Analyze more features of input music to incorporate into parameters for the video processing filters.

Conclusion

We would like to thank the COS 426 course staff for their support and enthusiasm this semester! This final project, along with the other graphics assignments, have helped us to realize the wide range of applications and immense possibilities for computer graphics.

Sources

Projection Resources
https://math.stackexchange.com/questions/296794/finding-the-transform-matrix-from-4-projected-points-with-javascript.

Jitter & JavaScript
https://docs.cycling74.com/max7/maxobject/jit.matrix
https://docs.cycling74.com/max5/tutorials/jit-tut/jitterchapter46.html

Video links
https://www.youtube.com/watch?v=6D-A6CL3Pv8
https://www.youtube.com/watch?v=qk00gbDwGqM
https://www.youtube.com/watch?v=IpVf5-aTgys
https://www.youtube.com/watch?v=4DuNkr3oAEU
https://www.youtube.com/watch?v=5Hqp0tai_qs
https://www.youtube.com/watch?v=Vmb1tqYqyII