Live Datamoshing

Garrett Gu, Alex Huang

We have completed the eCIS survey.

We demonstrated our project during Digital Demo Day for extra credit.

Our artifact is available at the following link: https://www.youtube.com/watch?v=Abdq_CCXnCs

Summary

Datamoshing is the process of corrupting files to create art. One popular method of datamoshing is known as I/P, where the compressed frame data of MPEG video files are skipped, repeated, or corrupted. We present a live datamoshing system that is capable of performing a wide variety of I/P effects on live video feeds.

Algorithms and Components

The main centerpiece of this project is a reimplementation of an MPEG-like video compression engine. Essentially, a compressed video is composed of I-frames, P-frames, and B-frames. We implemented the generation of P-frames since these are the only ones necessary for the effects we wanted to explore.

An I-frame is essentially a full frame with no compression. It can be rendered with no previous context.

A P-frame can only be rendered given the previous frame. A P-frame consists of I-blocks and P-blocks. An I-block is simply raw image data and can be rendered with no context. A P-block is essentially a motion vector (dx, dy) which then refers to a block within the previous frame.

Then, generation of a P-frame is done using the following algorithm:

```
for each block:
    search through previous frame for a neighboring block that "looks like" the target block
    if no lookalike block found, render I-block. else
        compute the distance (dx, dy) from this neighboring block to the target block
```

Now, our datamoshing effects work by corrupting the motion vector or collecting I-block and I-frame data from entirely different videos.

Implementation

We implemented the motion estimation algorithm (P-frame generation) and motion reconstruction algorithm (P-frame consumption) inside WebGL shaders. Both accept input video frames as textures and perform the bulk of their work inside fragment shaders. The motion estimation algorithm is rendered onto a frame which is 16x16 times smaller than the input image since we are operating on 16x16 blocks.

The algorithm takes in two separate inputs which we transform into video frames: video files, raw video from the webcam, or audio files. For audio files, we generate audio visualization on the fly based on audio nodes obtained from the WebAudio API to create dynamic patterns.

We also use the MediaRecorder API to allow users to save their performances. When a recording is active, the blobs from the output canvas are saved into a buffer, which is then parsed into a WEBM video that can be downloaded after the data is fully loaded.

We also implement various tuneable transformations which can be applied on top of the motion vectors. These transformations are uniform modifications which produce interesting visual effects when applied in conjunction with the right videos.

As a limitation, the user interface is fairly clunky. In addition, while all the functionality is present and working, switching between modes currently requires manually editing the source code.