EE168 Final Report

Winter 2016

Nattapoom Asavareongchai, Min Cheol Kim, Chayakorn Pongsiri, Vivian Wang

Introduction

In our animation, a player participates in an escape room game with a Pokémon theme. The story leads the viewer through the first person perspective of a player trying to battle through five rooms along a hallway before finally reaching the battle round against Titan. Each of the rooms contains a Pokémon enemy that needs to be defeated in order to escape the room. The player eventually wins, using clever image processing techniques along the way to fight the different Pokémon opponents. From the production end, a variety of image effects were implemented to recreate animated scenes with lively action and engaging visuals.

Implementation

At a high level, each room was developed in MATLAB through separate scripts in order to manage the computational complexity involved with making a video containing hundreds of image frames. The video segments generated through these scripts were eventually concatenated and combined with varying background music audio effects to create a full animation experience. In this section, we describe the storyboard plot and image processing techniques used to implement each room in the animation, with sample illustrative stills from the animation.

Opening

Plot Sequence

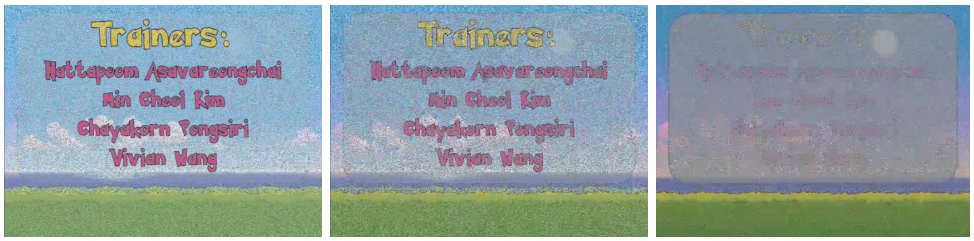
We begin by introducing the story's context with title, author, and background story frames with fading and dissolving transitions in between. We finally fade out of the story frame and into the room where the player wakes up and opens his eyes. Disoriented, he attempts to focus on his surroundings, which blur and sharpen, before turning on the lights and exiting the bedroom. Upon entering the hallway, a player selection frame pops up, displaying each of the possible Pokémon characters in turn. The player selects Pikachu as his character, and Pikachu's circle flashes a couple of times before Pikachu opens up in a smile of excitement. The scene behind Pikachu blurs out with an increasing motion blur before glowing with a bright red flash. A dark blurry circle closes in on the scene as we exit the hallway and enter the first room.

Image Processing Techniques

1. **Frame Succession**: Frames for each of the accumulating story text lines were pre-saved then loaded and played consecutively with some time delay. For ease, this was also used for the player selection segment as it is cumbersome to draw each frame within MATLAB while generating video.
2. **Fading transition**: We linearly varied the fractional weights of the summed images, such as to produce a weighted average of the initial and final images. The initial image appears to blend into the final image.

3. **Dissolving transition**: We randomly selected an increasing percentage of points in the original image to take on values in the final image. The resulting effect visually somewhat resembles static noise you might see on a television.

Frames illustrating succession of dissolving transition:



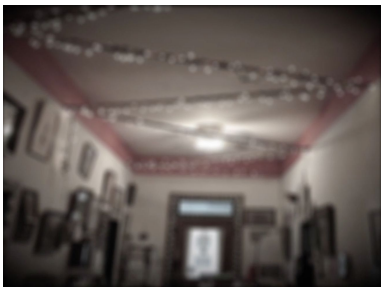
4. **Expanding/contracting vignette**: At the beginning, the inverse of an expanding ellipse is added to the image, with some averaging to give the impression of a semi-transparent overlay. The expanding ellipse, created by linearly increasing the axis length of the ellipse, is used to reveal the scene and generate the appearance that the player has just opened his eyes. The function for the closing vignette at the end is similar to that of the high-pass filter function g(q) used in sharpening, as it has a blurry, circular shape fit for the effect.
5. **Blurring**: Blurring is carried out by convolving the image with disks of increasing and decreasing radii. This serves as a low-pass filter that averages each pixel's neighbors with radially decreasing weights. Note that the circular blurring kernel gives the bright lights in the initial image a bokeh-like effect in the final image.
6. **Sharpening**: Sharpening is carried out by applying a frequency domain filter that enhances frequencies in the higher part of the spectrum; the high-frequency components typically contain the finer details of an image. The Gaussian frequency domain filter is given by 1/g(q), which is given below. This high-pass filter attenuates frequency components near the image center. The animated sharpening effect was created by using successive filters with changing σ values. We find that a = 0.9, b = 0.002 gives a visually appropriate effect.

$$g(q) = \frac{e^{\frac{-q^2}{2\sigma^2}}}{\sigma(a\sqrt{\pi} + b)}$$

7. **Wobbling**: A wobbling effect was simulated by rapidly translating the image through alternating up-down, left-right shifts. We used small pixel lengths to create a disorienting effect as the player hypothetically lifts his head. Note the translated image is cropped to remain within the fixed frame size.
8. **Saturation increase**: In order to improve animation efficiency, the transition of the room image from partial saturation to full color saturation was implemented with a standard fading transition. The initial de-saturated room image was created by separately converting the all image pixels from RGB to HSV-space, uniformly decreasing the saturation values, and converting the image back to RGB values. Therefore, this image was saved and pre-loaded to be used into the video code.
9. **Brightness increase**: We adjusted the histogram of the image, such that the pixel values take on higher, i.e. brighter or lighter, values. The mean of each color channel histogram was

re-normalized to have a higher mean and constant standard deviation. Enhancing the image brightness has an especially strong effect on the room's ceiling light, giving the impression that the room is lighting up.

Frames of the room photo showing (a) blurring and darkened corners from the vignette, (b) sharpening, (c) fully saturated image, (d) brightened image:

(a) 

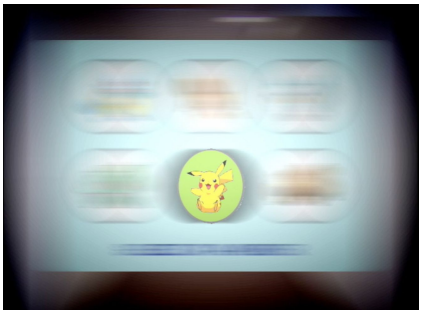
(b) 

(c) 

(d) 

10. **Red flash**: This is created by increasing then decreasing the red channel weight.
11. **Motion blur**: We convolved the image with a motion blur kernel, whose length increases with time. The kernel is a 1D array where each element has value (1/array length). This creates a horizontal motion blur effect, where the "speed" of the motion increases as the kernel length increases. Note that the motion blur is applied only to the background of the selected Pikachu character; a pre-defined mask (selected using interactive user input through *roipoly* command) was created to eliminate Pikachu from the effect.

Frame showing motion blur of the entire image except for Pikachu's circle:



Room 1

Plot Sequence

The scene of the first room is first introduced before the battle takes place. The player enters a haunted forest and looks up and around the forest background in a rotating pan before stepping

into the fighting arena that fades in. The player's character, Pikachu, moves into the scene through a horizontal translation. Zorua fades into appearance and evolves into Zoroark, the opponent. Zoroark morphs between different battle stances before assuming his final fight position diagonally behind from Pikachu. The scene darkens to provide a ominous feel.

In the first fight sequence, Zoroark uses his illusion ability to trick Pikachu in thinking that Zoroarks have multiplied across the forest. We then cross-dissolve into the correlated color image in order to pinpoint the locations of the Zoroark copies, shooting Pikachu's signature lightning bolt at each one. The correlation image wobbles and fades away.

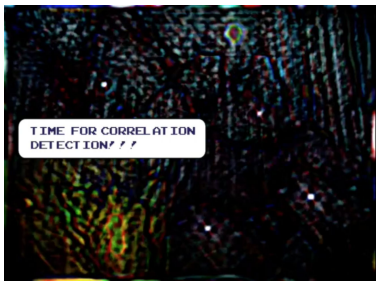
In the second fight sequence, the forest takes on a more reddish-tone, foreshadowing the attack to follow. Zoroark's tail pulses with a red glow before its whole body starts shaking and a green bubble expands around its body, which now exhibits a bright red aura. Pikachu then morphs into attack position and moves to attack: its image shears, distorts, and scales as it jumps backwards and moves forward towards Zoroark. Zoroark then uses its Night Daze ability to emit radially symmetric, dark ripples that pulsate back and forth at Pikachu. Pikachu throws a berry along a projectile path towards Zoroark in response. A lightning bolt spins and zooms into view at Zoroark's position. The image flashes yellow and Zoroark disappears in defeat. The red-tinted image fades back to its original atmosphere, or histogram.

Image Processing Techniques

1. **Rotating pan**: We rotated the image while keeping the viewpoint at the image center, giving the impression that the player is looking up at the trees while rotating his head. We change the angular speed of the panning by changing the step size of the angle chances.
2. **Image morphing**: We apply this solely to the Zoroark image, and superimpose each morphed frame on the background image, by only displaying the non-zero values of the morphed Zoroark sub-image on top of the background. We chose to use an image morph instead of a blend/fade effect in order to animate more realistically the physical movement of Zoroark, who changes between stances. We morph Zoroark through intermediate frames of Zoroark in transitioning stances, until morphing him into the final battle stance. Warp points for each morph are user-specified through ginput, pre-saved, and then loaded into the video generation script in order to aid efficiency.
3. **Image darkening**: We lowered the mean of the image histogram.
4. **Correlation**: The correlation image was formed by performing a cross-correlation of the Zoroark image across the entire image to locate all of Zoroark's locations. Bright white spots in the correlated image represent these locations. We perform correlation on each channel independently, then recombine the channels into the final composite color image. This simultaneously yields an artistically interesting effect resembling blurry nighttime viewing.
5. **High-pass filtering**: We filter the correlated image with a high-pass filter of decreasing σ to enhance the bright correlation peak spots.

Frames illustrating (a) cross-correlated image and (b) high-pass filtered image:

(a) 

(b) 

6. **Wobbling bouncing**: Wobbling is implemented by translating the image up and down with exponentially decaying values, in order to simulate a somewhat realistic springy bouncing-like effect.
7. **Color pulsing**: Weight of the red channel was linearly increased and decreased to yield a pulsing effect. This effect was only applied to Zoroark's tail by using a mask whose boundary points were pre-selected around the tail through user input.
8. **Character aura**: Zoroark was convolved with a Gaussian kernel with increasing size. The enlarging green "bubble" around Zoroark results from numerical edge effects around Zoroark's border (effective image endpoints, given that Zoroark's sub-image is placed against a black = 0 background) from convolution.

Frames illustrating (a) image morph and (b) red aura/teal bubble as well as reddish forest:

(a) 

(b) 

9. **Shaking**: The Zoroark image was scaled by a slight amount (alternating between 0.99 and 0.98) to generate shaking.
10. **Projective mapping**: A series of matrix transformations are applied to warp and project Pikachu into different perspective positions along a jump-and-shoot-forward path. These projection equations are derived from the concept of image homography used in 2D planar perspective mapping. This was necessary to generate interesting shearing distortions of the Pikachu image to simulate jumping.

Mathematical Background:

The homography matrix *H* for perspective transformation with 8 degrees of freedom is:

$$\begin{bmatrix} x' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (1)$$

From this, we see that (x_i', y_i') are given by:

$$x_i' = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + 1} \tag{2}$$

$$y_i' = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + 1} \tag{3}$$

which is equivalent to the following matrix equation:

$$\begin{bmatrix} x_{p,i} & y_{p,i} & 1 & 0 & 0 & 0 & -x_{p,i}x'_{p,i} & -y_{p,i}x'_{p,i} \\ 0 & 0 & 0 & x_{p,i} & y_{p,i} & 1 & -x_{p,i}y'_{p,i} & -y_{p,i}y'_{p,i} \end{bmatrix} \mathbf{h} = \begin{bmatrix} x'_{p,i} \\ y'_{p,i} \end{bmatrix} \tag{4}$$

where $(x_{p,i}, y_{p,i})$ is the i^{th} selected point in the original image, and $(x_{p,i}', y_{p,i}')$ is the point in the new canvas to which the point should map. Our actual matrices will have $2n$ rows for each of n corresponding point pairs used in the planar mapping. We can find h_{ij} by solving the matrix equation in MATLAB with the backslash operator. We apply the H matrix to the whole image as a projective transform in MATLAB. Physically, this is equivalent to viewing the image in the same plane but from a different angle. For different movement trajectories, we linearly vary one of the particular, experimentally determined values of the H matrix used in mapping.

A summary of the algorithm (in our animation, we use $n = 4$):

1. Save n points on the original image plane $(x_{p,i}, y_{p,i})^T$, such as by using *ginput*.
2. Save n desired corresponding points on the new image plane $(x_{p,i}', y_{p,i}')^T$.
3. Compute the homography matrix H from each $(x_{p,i}, y_{p,i})^T \leftrightarrow (x_{p,i}', y_{p,i}')^T$ relation in eqn. (4).
4. Coordinates of the image points are related through $(x_i', y_i', 1)^T = H(x_i, y_i, 1)^T$. We apply the inverse transform in order to avoid holes.

Frames illustrating shear projective transforms:

11. **Rippling**: Inversely decaying, circularly symmetric, 2D cosinusoidal ripples were overlaid on the image to approximate ripples. The phase of the cosine was adjusted to give the impression that the ripples pulsate back and forth.
12. **Projectile flyby**: The berry is translated along a projectile trajectory, which is quadratic in nature (linearly varying *x* position, quadratically varying *y* position). The berry also scales down in size as it travels towards Zoroark, who is situated farther in the background relative to Pikachu's frontal position.
13. **Spinning**: The scale of the bolt was increased as the angle of rotation of the lightning bolt increased from 30º to 360º. This gives the impression of rapid outward spinning.
14. **Color modification**: A reddish forest image, which foreshadows the incoming Zoroark attack, is created by gradually enhancing the red channel of the image. The final yellow flash of the image after Pikachu wins is generated by adjusting the image histogram such that the mean of

the red and green channels are weighted higher than that of the blue channel (in light-space, red + green = yellow).

Room 2

Plot Sequence

The second room is an underwater room with a sea Pokémon enemy (a Seadra). The main plot of this room is that the player enters an empty room. The room then starts to fill up with water, turning it into an underwater room. This transition is mainly done using translation of images and linear combinations of images added together. In other words, we place subset images on top of one another and move them over the course of the animation. The Pokémon enemy is tiny and hides within the water. The enemy then chooses to attack us using a cosine 2-D ring wave as well as a rotation. The player finds the Pokémon by zooming into the origin of the attack. Zooming is done using a first-order-hold zoom technique. After finding the enemy, the player attacks using another Pokémon (Pikachu). The attack is accomplished through linear translation of a lightning image. Different RGB color channels are alternately displayed to create flickering light to show the attack effect. Text narrations within the rooms are part of the images. We added them using an external program and created a image from it to load into our MATLAB script.

Image Processing Techniques

1. **Image subset**: This is not difficult with simple images where the purpose is to blend two images into one. This is done by applying a linear combination of the two images (e.g. 0.5*img1 + 0.5*img2). This effect is great to use in this room to show a room filled with water, not just an empty room or just an underwater scene.

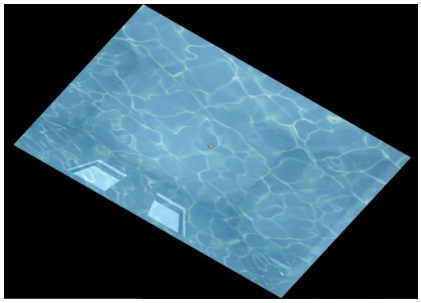
   Subset images become harder when you want to add a smaller image on top of the large background image frame without blending them together. A technique that is used in Room 2 is to first set the background color of the smaller image to black or an intensity of 0 for every RGB values. For the pixel locations that we want the smaller image to be situated, we set the pixel of the background image to the pixel of the smaller image, as long as the pixel is not black for the smaller image. (i.e. if smaller image pixel != 0 for all RGB values, set the background pixels to the smaller image pixels). This is quick and easily done. However, a downside to the image is that if there is a black color image we want to add on, it doesn't do so since we ignored the black pixels in the if statement. This can be solved by setting the black pixels we want to add to gray or not completely black so the if statement doesn't ignore it (by setting the RGB values to a very small number). Note this technique is also used in the previous and subsequent rooms.

2. **Translation**: Only linear translation is used in Room 2 which is simple to implement. In each frame, the location of the image to be translated is changed and calculated within a *for* loop. The *x* and *y* coordinates are modified in every loop and a frame is created in each.
3. **Rotation**: Rotation used in this room involves rotation of the whole background image. We wrote a rotation function to do this. This function takes in a background canvas, an image to be

rotated that is smaller than the canvas, and a rotation angle in degrees. The background image is shrunk so the rotation of the whole image is seen.

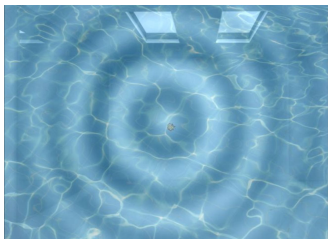
Frames illustrating (a) water in middle of translation, image subsetting, and blending (note the water and room images are averaged to give a transparent effect) and (b) image rotation:

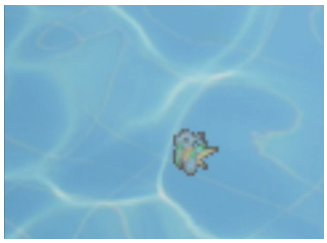
(a) 

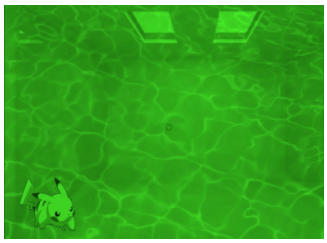
(b) 

4. **2D cosine wave**: This is the same as the rippling effect in Room 1.
5. **First-order-hold zoom**: A first order hold function is written that takes in an image and the zoom factor and returns the zoomed images. A first-order-hold is used because it creates a more pleasing image than zero-order-hold and is simpler computationally than a second-order-hold. The zoom is applied several times with an increasing zoom factor to create an illusion of slowly zooming/moving inwards towards the zoomed object.
6. **Color modification**: The flickering color attack effect using different RGB color channels was accomplished by zeroing out two different channels in each frame (3 variations in total) and displaying the result within short periods of time in an alternating fashion.

Frames illustrating (a) 2D cosine wave ripples, (b) 1st order zoom, (c) color modification:

(a) 

(b) 

(c) 

Room 3

Plot Sequence

Transition into Room 3 is done by blending in the third room into the hallway. The third room is a room filled with trees and flowers with a plant Pokémon (Vileplume) hidden within the background image. However, upon entering the room, the image is filled with smoke/smog which blurs out the image. This smog is just another image of smoke blended into the background image. The intensity of the Pokémon is increased 5 times so image processing techniques can separate it out later on. Still, the Pokémon is well hidden in the picture frame. Small skull and bone images are displayed at different locations in different frames to indicate toxicity of the smoke in the background image. To figure out where the enemy is, the player applies a false color effect on the image, by stretching the histogram of the different RGB channels separately. The blue color

channel is stretched more than others for a different false color image. The enemy is spotted right after. However, the player also applied negative color to display the enemy more clearly. The room then transitions back to the original image with no smoke this time. A morph is used to transform the player's Pokémon (Pikachu) into another Pokémon for a different attack. A fireball attack is then used by the player's Pokémon to defeat the enemy. Fireball images are subset and translated onto the image using the same technique as Room 2. Then the enemy is blended out of the background image.

Image Processing Techniques

1. **Image subset and translation**: Image subset and linear translation techniques in this room are the same as the ones used in Room 2.
2. **Image blending**: We blended from one scene to another to create a smooth transition by changing the fraction of the images intensity each frame. (i.e. *frac*img2 + (1-frac)*img1* where *frac = 0:10* and the image changes from *img1* to *img2* within 11 frames by blending).
3. **False color image**: In order to create a false color image, we calculated the histogram of all RGB channels of the image. The histogram of these RGB channels are then stretched by changing its mean and variance. If a value goes over 255 or under 0, then it is clipped at 255 or 0 respectively. Here, we stretched only one of the RGB channels and left the other two as is. We tried the stretch for all three channels, saw which one is best, and used that as the image in the movie frame.
4. **Negative image**: This is done by switching the high and low intensities of all three RGB channels. For example, 255 would become a 0 and vice versa.
5. **Image morphing**: Coordinates used for morphing are selected in a different MATLAB function, and are chosen only around the pokemon to be morphed.

Frames illustrating (a) original blended image, (b) false color image, (c) negative image:

(a) 

(b) 
(c) 

Room 4

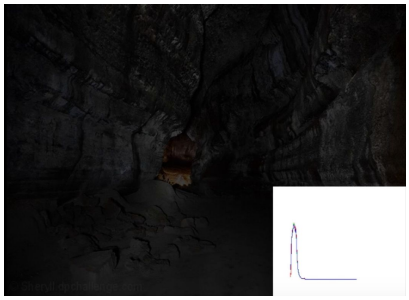
Plot Sequence

The main character enters Room 4 and finds that the room is too dark for him to see anything. He uses a histogram stretching technique to brighten up the image. He then looks around the room to look for the Pokemon and finds a stone wall where he suspects that the Pokemon might be hiding. Because the Pokemon might be blended into the wall, he tries to apply edge detection technique, and he finds out that it indeed helped him locate the Pokemon. He then can easily attack the Pokemon with a water gun.

Image Processing Techniques

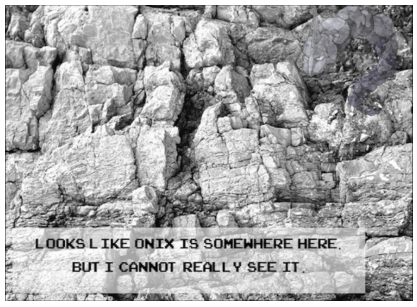
1. The main image processing technique that was used in this room is histogram stretching to make an image brighter. We stretched the intensity histogram (instead of stretching histograms of each color individually) using different means and standard deviations (mean ranging from 10 to 160 and SD ranging from 5 to 80). In general, the mean to SD ratio was kept to be 2:1, and the final mean and SD values of 160 and 80 were chosen because they resulted in the most visually pleasing image.

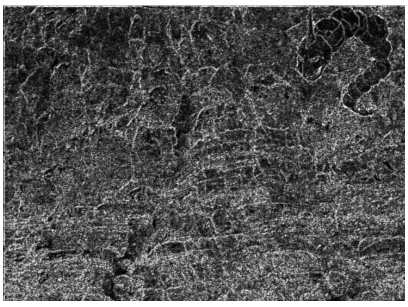
Frames illustrating change in image histogram from (a) dark to (b) bright:

(a)  (b) 

2. The next technique that was used is edge detection. There was a scene where the background is a rock wall and a Pokemon is hiding (blended to the background with 60% weight from the Pokemon and 40% from the background). This makes it hard to visually spot the Pokémon. We applied a Laplacian mask to the image, which resulted in an image whose edges were sharpened. With edge detection, the Pokemon is clearly distinguishable, and the main character can easily attack him.

Frames illustrating (a) original image with faint Onyx image and (b) edge-detected image (note dark Onyx shape in top right corner):

(a) 

(b) 

3. Other minor techniques that were used include image blending, translation, and extracting color component. Blending and translation were mostly used to transition from scene to scene or adding new components to the image. Color extraction was used in the last scene. To highlight the Pokémon, only blue component of the image was extracted to show that the Pokémon is weakened and about to disappear.

Room 5

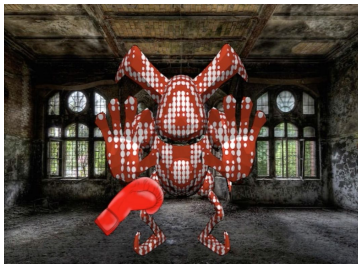
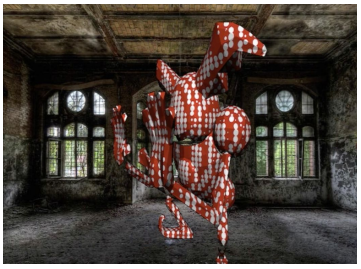
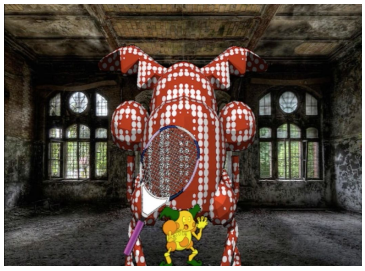
Plot Sequence

In this room, the main character sees a giant Pokemon that looks rather harmless. He tries to attack the Pokemon but fails. He then realizes that this might not be a real Pokémon and perhaps the real one is hiding behind it. So he turns the giant Pokémon and finds Mr. Mime (the real one) behind. He then attacks Mr. Mime using a racquet.

Image Processing Techniques

1. In this room, we created a 3D rotation of a big Pokemon by putting 20 images from a 3D model viewed from different angles in sequence. Another image processing technique that we used is image rotation. Once the big Pokemon was turned around, a small Pokemon was revealed. He was hit by a racquet that was rotated using image rotation.
2. Other techniques that were used in Room 5 include zooming, blending, and color extraction. At the opening scene, an image of a doorway was shown and then we zoomed in to the door at the end of the hallway using first-order zoom. Similar to Room 4, blending was used mainly for transition from scene to scene, and color extraction was used to highlight the Pokémon. In this case, we extracted only red and green components of the Pokémon image to highlight the Pokémon with yellow.

Frames illustrating different rotations of the large Mr. Mime figure with (a) fist in the middle of translation, (b) mid-transition of large Mr. Mime, and (c) image subsetting of tennis racket and yellow-colored little Mr. Mime.

(a)  (b)  (c) 

Boss Battle

Plot Sequence

The scene leading to the boss battle starts out by zooming into a smaller and smaller part of the original hallway, using a convolutional first order hold. When the viewer reaches the door at the end of the hallway, the scene morphs into the gates of hell image. The text “Woah” fades in with its textbox, using the subset technique. Once we enter the room, the boss and the main character rotates into view, getting larger/brighter with each rotation. Before the scene settles, the earthquake effect is created by a series of rotations in opposite directions with decreasing magnitude. Then, the image turns into an indexed color image and back rapidly, creating a thermal-image/thundering effect.

The scene enters a series of panes where the viewer is “selecting” the move to choose. This is achieved with a series of pre-prepared text-containing images that can then be resized to be our

frame size and the zeros (black letters) can be identified. The player selects low pass filter, and the filter morphs the background (only the Titan, leaving the wall/Pikachu intact) to an image only containing the lowest frequencies in each of the colors. The low frequency image morphs back to normal, suggesting the attack did not work. The player then selects color kill, and several arcs simulate the drainage of the color red from the Titan. The Titan remains with none of the red component in the RGB image, and Pikachu unleashes a series of thunder attacks, using subsets. The duration between each lightning decreases over time, and the Titan is defeated. The entire scene morphs into a door, and the game completes. The scene fades out into a complete white, and we fade into a picture of Professor Zebker, telling of a student in EE168.

Image Processing Techniques

1. **Morph**: hallway door to Gates of Hell

Technique description: We used the standard morph we learned in class, with 11 intermediate images that we generated. A morph was a reasonable choice, since we were changing a door to another door - there are specific displacement vectors that could be defined. In addition, we wanted to create a illusion of stepping into this “hell,” and the morph provided the effect.

Implementation: 20 displacement vectors were defined, each corresponding to a feature on the door or the background.

2. **Rotation/brightness scaling**: earthquake

Technique description: The image was rotated and brightness-scaled simultaneously as the Titan came into view. Then, using a pre-defined list of angles of alternating signs, the earthquake effect was created.

Implementation: The *imrotate* function in MATLAB was useful, and the images were scaled/rotated according to two arrays that contained the appropriate scaling/angle at each step. The general equation for each step was: *final_image = scale_factor * imrotate(original_image)*.

Frames illustrating (a) morph into gate of hell and (b) earthquake effect from rotation:

(a) 

(b) 

3. **Indexed Color**: Lightning Effect

Indexed color technique was used to create the thermal effect as the Titan is introduced. Here, instead of defining our own index-to-color map, we used the imagesc function in MATLAB with only the red values of the image.

Implementation: The default colormap, image , so the MATLAB-defined color indexing was used to produce the image. Then, the *cData* variable from the getframe method was used to retrieve the image that was shown, and the matrix was resized to fit our frame.

4. **Text Display**:

In many of the scenes in the boss fight sequence, the text along with the textbox was used.

Implementation: This was achieved by having a black/white image of the textbox containing the black characters, and then padding the image with 255's so that a white background image with only the black textbox frame and the text itself was generated. The indices were the text image were 0's (hence representing the box frame + text) were stored, and the pixels at those locations in the original image was set to 0. This created the effect have having black texts/box.

5. **Text Fading**:

We used several text/textbox fading in during the boss battle sequence. A blend was used to achieve this effect.

Implementation: The blend between the original image and the final image with the textbox/text inserted, to create the effect of the text/textbox being more and more defined (those pixels more close to 0 value).

6. **Low pass filter**:

As an effect of one of the attack, a low frequency image of TItan was shown. To generate the low frequency image, we defined a filter in the frequency domain and the filtering was carried out in the frequency domain, then converted back to the spatial domain.

Implementation: For each color (red, green, blue), a Fourier representation of each channel was generated using FFT2. Then, a low frequency filter (a square of width 300) was defined in the frequency domain. The filter and the DFT image was multiplied, and each of the result was converted back to spatial domain. Finally, the three images were put together in the traditional RGB fashion.

7. **Red arcs**: Gaussian blur on concentric circles

The red arcs in the color draining scene were created as parts of a circle then was blurred using a Gaussian filter.

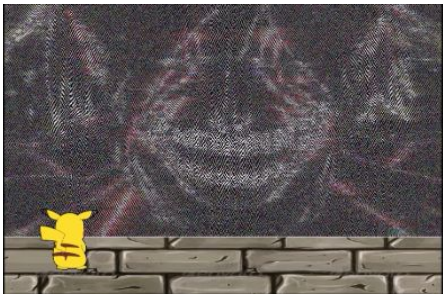
Implementation: The red circles were drawn using the *viscircles* function, then blacked out if they did not fall under the range of angles. Then, the *imfilter* function was used with the pre-defined Gaussian filter.

8. **Color modification**: The weakened Titan

The red component of the Titan was reduced more and more as the Titan was affected by the color kill attack.

Implementation: The red component (first indexed matrix) was scaled by each element in the vector ranging from 1 to 0 in sequence. This created the effect of the color red slowly decreasing in the TItan part of the image. Because the color loss was only desired in the Titan and not the Pikachu and the wall background, a new image had to be stitched together at every step of the red color drainage (every time the Titan loses a little bit of red).

Frames illustrating (a) low pass filtered Titan and (b) decreasing red component:

(a) 

(b) 

9. Door morph: The Way Out
The background with the Titan was morphed into a door with a starry background.

Implementation: This morph in implementation was nearly identical to the gate of hell morph used in the beginning of this sequence. However, a key difference in this morph was that a random set of displacement vectors were used to create a “blackhole” effect. The Titan background and the door/space background had no real corresponding points, and the random vectors created a desired effect.

10. Whiteout: Escape!
Each frame in the sequence was multiplied by an increasing scalar to increase the overall intensity, until the image became completely white. Then, we switched the image to the picture of Professor Zebker (but equally as white) and slowly scaled back to the normal intensity.

Implementation: At every step of the brightening, the frame was multiplied by an element in an array that held the intensity multipliers. By the the time the image was multiplied by 7, the entire image seemed white.

Frames illustrating (a) random morph and (b) whiteout:

(a) 

(b) 

Conclusion

Effectiveness of Results

We effectively applied image processing techniques that we have learned in class to create a short movie. Not only did we use different image processing techniques to attack Pokémons, but we also used image processing techniques for transitions from scene to scene or from room to room to make sure that the movie flows smoothly. We believe that our movie demonstrates that we can apply the techniques we learned in class effectively: the techniques applied in each room were related to the visual effects we sought to generate. In Room 1, when Zoroark uses Illusion ability to duplicate across the screen, correlation was a naturally and physically fitting technique to fight with. In Room 4, we situated the opponent Onyx against a rock background, which lent itself well to the application of edge detection. To simulate different environments, we adjusted image color histograms and individual color channels to generate the effects we wanted (e.g. bright vs. dark scenic mood).

From a methodological standpoint, our division of the movie scripting in MATLAB was effective both for dividing work between group members and for rendering the large number of frames required for the approximately 4 minute long video, given the speed and power of MATLAB. We also vectorized as many operations as possible in order to improve the speed of our code.

Future Improvements

One improvement that we could add to the movie is perspective and fly-by. We could add one more scene where the main character needs to look at an object or place from different perspectives to figure out where the opponent Pokémon is. This would take better advantage of the perspective and projection equations from class to mathematically compute the flyby scene solely within MATLAB. Given more computational power and thought, more detail could be given towards rendering more physically realistic and/or nuanced effects during the fight scenes. Depending on the anticipated use of the video, better code efficiency may be helpful if we were to produce our movie with larger frame sizes and thus more data.