I. Introduction

Our project was based on the theme of volcanoes. We thought that it would lend well to the different image processing techniques that we had learned throughout the quarter. The main techniques that we wanted to incorporate into our project were: filtering, translation, rotation, blending, morphing, perspective projection flyby, topographic data, zooming, and filtering. We also wanted to be able to do something at the end that would be very unique while also using ideas from techniques we used in our class.

II. Description

Our project was based on three different major scenes. We start out with the introduction to our project which gives the title and introduces our group. In the typographic animation, our members’ names are translated around the screen, and they are also blurred using blur filtering and convolution techniques. We also used pictures of each of the group members, and we morphed them together. This blends into the next major scene which consists of the perspective view flyby and topographic view. The flyby has the perspective projection of the big island, and this view is rotated and seen at different heights throughout the flyby. This fades into the topographic view of the big island. We made a sequence where the island was illuminated from different angles that rotated around the island. Next, we went through a series of zooms and translations until we were centered on the highest peak of the big island. The peak then blends into a volcano; a flow of lava is blended into the peak. This fades into the final major scene, which is the 3D view of lava spurtting. The lava spurtting consisted of particles shooting out at random angles from the center of the image, and coming down in a gravitational field. This lava spurt then freezes frame, and the camera begins to fly around at different angles and heights through these particles. The movie ends with the freeze frame unfreezing, and the lava begins spurtting again.

III. Discussion

Part 1- Typographic Animation

In the first part of the project, we concentrated on making a title animation. We used our names for the part: Rick, Zune and Josh. Here we applied several image processing
techniques such as blur filtering, translation, blending (transparency), and morphing. First of all, we tried to construct a few blur filters: blur, Gaussian blur, and motion blur to distort original text images. Also we used those techniques to transit from one to another scene. The title animation is composed of three sub scenes. We applied combination of image blending and blur techniques to transitions to first two scenes. For the 3rd scene, morphing was used. Morphing is a most suitable way of transiting from a member’s face to those of others, so we used it for an introduction to us. To give a different atmosphere to the first part, we decided to use only gray-scale mode there like unvoice movies. An extra technique was used, that is an exploding effect. As seen in Fig.1 and 2, our names will explode at the time of transition.

Here are more details on each sub scene. In the first scene, the text of the class title and members’ names fade in and fade out. To each fade-in, a blending technique just was applied (we made a Matlab function, ‘transparent’). Each text image fades in from a black background and then it fades out using Gaussian blurring and black-blending. Over the time we set little bigger values to the Gaussian function to get more blurring effects on the image together with blending. To the last two text images (Zune Lee and Rick Yang), motion blur effects were adjusted. We can see the names slightly moving to left-bottom or right-bottom corner in the frame.

In the second scene, the animations of the names repeat differently. We made an exploding effect by modifying blur filters and some names are exploding with moving to the left-top corner. Especially, in the animation of Josh’s name we’d like to maximize the motion filter effects (Fig.3 and 4) and you can see the name blurred and shifting to right and left.

**Part 2- Flyby**

During the second segment of the video, we wanted to do a flyby of the big island, and then we wanted to zoom into the highest peak of the island, where we wanted our volcano to be. The transition used between the introduction and the perspective flyby was made using a fade technique where the ending scene of the introduction faded into black, and the first scene of the flyby faded from black.

The bulk of this section of our project was the perspective flyby. The perspective view of the big island was created by creating a 2D projection plane that came between the viewpoint and the actual 3D image. The projection transforms a point in x,y horizontal and z vertical space into a point on a u,v plane that is parallel to the original x,z plane. The perspective can be changed by changing the position of the viewpoint. We wanted to be able to create a feeling of riding a roller coaster track around the island from the viewers’ perspective, so we wanted to combine both a rotation as well as a variation in the vertical viewing angle.

The first scene of the perspective starts off with the viewpoint with a small vertical offset from the ground plane, so the first view is almost from one side. As the movie progresses, the perspective rotates a full 360 degrees while the viewpoint also keeps
going higher and higher. By the time the island rotates the full 360 degrees, the viewpoint has risen quite a bit, and the viewer can see the entire island from an angle that almost looks like a topographical map. Once the video reaches this peak, the island rotates back around and the viewpoint sinks back down to the original position. The color scheme of the island is the same as the one we used in homework 8. The easiest way to deal with the color was to only use the data in the elevation that was greater than zero. Before the perspective view was projected, we created a background that was entirely blue to represent the water. We took several measures to get rid of the holes that often come up in the perspective view. We mapped each of the pixels from the 3D image to two pixels in the u,v plane. We also ran median filters on each of the color channels to take care of any random noise that was left. We also had to make sure the vertical scaling wasn’t too large; otherwise there would be many gaps in the image.

This goes into another blend which fades from the perspective to blue and back into the topographic data. The first sequence that happens creates the illusion that the sunlight is hitting the island from different angles. The illumination starts from the left, then rotates around at 45 degree angles to the bottom left, bottom, etc. The effect we were trying to create was an effect of the sun moving around the island. This moves into the zoom and translation stage of the topographic view. The image does a gradual zoom from a view of the entire island centered in the middle to an image of the island taking up almost all of the space. The island gets rotated around by 180 degrees and the image is translated to focus on the big peak on the island. Here, we used a blend to create the image of the volcano forming on the peak. We used a simple blend because the background image was the same; we only wanted to create the lava at the peak. Then the image zooms and translates so the volcano is centered, and then the screen fades to black.

**Part 3-Volcano**

The volcano effect relied on a 3-D particle engine built by Josh. The particle engine stored each particle as a four-vector, containing Cartesian coordinates and color. By specifying a camera location, and a “look direction” location, the camera projected these points onto a projection plane perpendicular to the look direction vector. The projection plane can be defined to an arbitrary size and distance from the camera.

The 3-D engine returns the 2-D projection of the points as a three-vector, returning the x and y coordinates of projection points, as well as the z-depth of the points. The z-depth is particularly important, allowing depth discrimination of complicated particle fields. The 2-D projection is rendered by drawing circles on an imagebuffer, taking care to use the z-depth to make sure particles closer to the camera accurately block particles farther from the camera.

There is a final element in creating the particle effect – the accumulation buffer. The accumulation buffer is a digital processing technique often used to create motion blurs and other photorealistic effects by emulating the finite response time of motion cameras. The accumulation buffer stores successive frames of the movie, allowing a continuous “trail” of images to accumulate. The accumulation buffer is also continually blurred by a
3x3 uniform blur, which creates a gradual fading effect. New frames are introduced into the buffer, and as they are continually blurred, they gradually lose definition, contrast, and brightness, until they are undetectable and fade out.

To create the volcano effect, I used a four-part sequence.

1. **Volcano Erupts**

To begin the volcano eruption sequence, I first built a small physics simulation to generate the particle locations for the 3-D engine to render. The particles were all given the same initial location – (0,0), and a velocity. The velocity was first defined in spherical coordinates, so that they could be given a uniformly random distribution of velocity angles in the horizontal plane, and given a very tight Gaussian distribution of velocity angles in elevation. I also gave a small variation in the velocity magnitude to add realism. These spherical coordinates were translated to Cartesian coordinates by the MATLAB function sph2xyz. Each particle was given a random starting time within the first two seconds of the animation.

Whenever a particle fell below the $z = -1$ plane, that particle was removed, and given a new starting velocity at the origin. The created the continuous fountain effect seen for the next part of the clip. The number of particles could be arbitrarily set – for the purposes of this final project, it was set to 450, although the amount is limited only by the memory of the computer and the patience of the student.

2. **Freeze**

The particles are frozen in midair simply by stopping the physics simulation. The new frame is no longer added to the accumulation buffer, so the accumulation buffer slowly fades away, revealing only the image generated by the current particles.

3. **Matrix-style panning**

To perform this panning, we simply allowed changed the camera location and the “look” location. We accomplished this by first defining a series of spherical coordinates with linspace, and then transforming them with sph2xyz. For this portion of the simulation, we did not use the accumulation buffer, as that illusion is only maintained for constant camera angles.

4. **Continuing volcano eruption**

After the matrix-style panning has returned the camera to its original location, the first animation continues with no loss in continuity.
IV. Technical implementation details

Part 1: Typographic Animation
For this part, we designed several filters: blur, Gaussian blur, motion blur, transparent and exploding filter.

**Blur**: we just used a simple convolution of an image and 3x3 mask that we learned about in the class. Instead of using conv2 function, we used fft2 multiplication in frequency domain and then used ifft2 to transform the result to spatial domain in order to speed up computing. But this method has a problem. We cannot control the intensity of blurring. We think one of ways to increase blur effects is to repeatedly apply the blur filter to the blurred image. Finally we realized that Gaussian blur is one of the best ways to overcome this issue.

**Gaussian Blur**: we could get a hint to design Gaussian blur from the high pass filter design to sharpen a blurred image using Gaussian kernel in the HW 5. We can use standard deviation to control the shape of Gaussian kernel. Instead of reciprocal of the Gaussian distribution, only what we need is to use the distribution itself for getting low-pass components. But we need to improve this method because it is relatively slower at about std >10.

**Motion Blur**: this filter is implemented by the combination of Gaussian blur and image translation. Fig.3 and 4 show the effect of this filter. Here is the pseudo code.

```plaintext
MotionBlur (image vector, standard deviation, afterimage factor, direction)
1) Take a 2d image vector, standard deviation, afterimage factor, direction;
2) Apply Gaussian blur to the input image and store it in a memory (img);
3) Copy the result, shift it as the direction argument, and store it in a temporary memory (tmp);
4) Reapply Gaussian blur one more to the temp;
5) Blend img and tmp by applying transparent filter, and output this vector;
```

As seen in the pseudo code, we need to set three arguments to perform motion blur effect **Standard deviation** (std) can control Gaussian blur effect which affects the blur motion effect by dispersing the input image more. **After image** is how strong the filter blends the two image, which means how strong the filter blend blurred image (img) and shifted-blurred image (tmp). It is related to transparency of two images. **Direction** is the direction of motions and we can set left, right, up, and down.

**Transparent**: transparent filter is a blending method for two gray-scale images using by linear interpolation that we learned in the class.

\[ \text{Blending out} = (1-a) \times \text{image1} + a \times \text{image2}, \]

where \( a \) is opacity factor, and \( \text{image1} \) and \( \text{image2} \) are input images.

**Exploding**: It is a simple tricky method to make an exploding effect. We added a Matlab code at the end of output:

\[ \text{Out} = \text{BlurredImg} + \text{BlurredImg}; \ \text{---------(a)} \]
Or,

\[
\text{Out} = \text{uint8}(\text{BlurredImg} + \text{ceil}(0.2*\text{BlurredImg})); \quad \text{----(b)}
\]

These two codes make a glowing and exploding effect by accumulating intensity. Code (a) causes the output to be too much bright, so code (b) is able to make a better effect. Constant 0.2 in the code (b) is just an arbitrary scaling factor, so you can change it case by case. It is very useful for white fade-out effect.

**Part2: Flyby**

We wanted to be able to make transitions between our scenes run smoothly, so we decided to blend the scenes together with fades of different colors. The implementation of these was fades was pretty straightforward, we used simple blend techniques. We would first take the last frame of the initial scene and create an image from it. We would blend this frame by frame, changing the proportion of this image over several frames into an image of solid color. Once this blend was complete, we would do the same thing again, in reverse order, with the same solid colored image and an image of the first frame of the next scene. This fade/blend transition was used to transition from the intro to the perspective flyby, from the perspective view to the topographic view, and then finally from the volcano blend into the lava burst.

The flyby was done using the perspective projection equations which mapped the x,y,z points into a u,v plane which corresponded with the x,z plane. The flyby sequence was created inside a loop which generated the rotation and viewpoint height change. Each new frame would be generated by rotating the original elevation data and slope data by 6 degrees in the clockwise direction, setting the height of the viewpoint to be 13 higher than the viewpoint was before. The original starting height of the viewpoint was 200, so by the time the loop finished running, the height was 980. We thought about also scaling the distance of the viewpoint from the projection plane, but we thought that the sequence looked more realistic with a constant distance of 700. The original elevation data of the big island had a scaling of 20 meters of elevation per unit of the elevation data. We found that we could create the best and most realistic looking image by scaling the elevation data down by 6. This way, the projection would not have too many holes, and it wouldn’t look too flat either. We mapped each point in the 3D image to a 2x2 box on the projection plane to take care of the holes using the repmat() command. The projection had to be done from the back of the image to the front so the closer part of the island wouldn’t be overwritten by the farther part. Once the viewpoint reached its peak, the flyby path went back in reverse until the final frame was back with a rotation of 0 degrees and the viewpoint was at a height of 200 and distance of 700.

After a fade into the background water and into the topographic view of the island, we decided to create an illusion of having the illumination of the island rotate around it. This was done by convolving the shaded relief with different masks that defined the direction of the light. The image started of being illuminated from the left using a mask of \([1 \quad -1,1 \quad -1]\). We switched the angle that the light was hitting the island by using a loop that ran through each mask in sequence so the light would continuously rotate around the island.
This moved into a zooming into the island. In order to do this, we basically resized the image to be larger and larger, and we just took the portion of the image that we wanted to zoom into. We wanted to keep all of the frames the same, so we would take a 640x480 portion of the image that was resized larger and larger, and this would give the effect of actually zooming in on the image. We ran a loop that would resize the original image by a certain percentage, and then we would take the 640x480 portion of the bigger image that was centered in the same place in each frame. Once we had zoomed into the view of the island we wanted, we translated our 640x480 box on the larger image until it was centered where we wanted it. This was accomplished by just adding x and y offsets in sequence to the position of the 640x480 box we were looking at. Rotating the island was also similar to the translation. Instead of rotating our viewing box of the island, we just took the enlarged image and rotated under our viewing box using imrotate().

The end of the scene consists of setting up the volcano on the peak. This was done easily by using a blend on the image of the peak, and blending it into an edited version of the image which had the red lava on it. The blend was made using the techniques described above, and the blend created a sequence which made it look like lava began flowing over the peak. The most inefficient code and the longest running code from these scenes was the perspective flyby. Mapping each point from the 3D representation of the image onto the 2D projection plane for each frame took quite a few computer cycles. Adding color and having the median filter on each color channel also added to the compute time. Because we wanted to be able to flyby the island 360 degrees and then go back down, it was much easier to run the code once to create frames that rotated 360 degrees and increased the height, and then just copy the frames backwards and append it to the original movie array. If we had just ran code that created the perspective views over again when our flyby went in reverse, the compute time would have been doubled.

Part 3-Volcano

Having described the overall animation process in section III, we will now describe in detail the algorithms used in each part of the process.

1. 3-D Engine

The 3-D engine is rather special in that it does not use any trigonometric functions to render the particles. Since MATLAB uses double-precision variables, trigonometric functions are generally very slow, and not practical for large scale particle rendering. Instead, only linear transformations are used.

The basic principle is very simple:
We first use the camera and look position to define a vector. From that vector, we define a projection plane perpendicular to that vector. Then we draw individual vectors from the location of each particle to the camera. It is a simple matter to find the intersection point of the plane with the vector. That intersection point is then expressed in terms of a 2-D vector in the projection plane. The distance of the particle to its point in the projection plane is used for the z-value.

To speed up this process, heavy vectorization was used, to take advantage of the inherently vector-based MATLAB functions. With vectorization, each frame (with 450 particles), took less than three seconds to render, which was about 10 times faster than the algorithm without vectorization.

2. The Accumulation Buffer
The pseudo-code for the accumulation buffer is:

```
AccumulationBuffer = zeros(640,480); % initialize buffer
for FrameNumber = 1:Numframes
    ImageFrame = GenerateFrame();
    AccumulationBuffer = max(ImageFrame,blur2(3x3mask, AccumulationBuffer));
    Movie(FrameNumber) = im2frame(AccumulationBuffer);
end
```

As the pseudo-code shows, the accumulation buffer successively adds and blurs images, creating a rather nice “motion-blur” or “ghost-trail” effect.
V. Limitations and Conclusions

In the process of creating our animation, we ran into limitations imposed by both algorithmic deficiencies and computational difficulties. For example, MATLAB would often stack overflow if the particle simulations were too large, if the frames were too large, or if too many frames were rendered. For this reason, the final volcano simulation was actually rendered in three separate scripts and concatenated together.

The other limitations imposed were algorithmic. For example, the perspective algorithm employed in the fly-by is inherently limited in its projection capabilities, and it is impossible to point the camera “down” without suffering image distortion. This was one of the primary motivations in developing the 3-D particle engine. However, the 3-D particle engine, because of its more robust capabilities, was more computationally intensive, and impractical for rendering the flyby animation. Thus, balances must be found between computational complexity and image fidelity.

We found that vectorizing our computations greatly increased our rendering speed. Storing intermediate results, while computationally intensive, prevented complex operations from being repeatedly calculated.

If we had more time, we would have spend more time improving our programs along these paths. More optimization, reducing computation complexity and memory footprint would allow us to create more complicated and intricate graphics. Another area we could not explore was down-sampling the final image from higher resolutions. If we rendered the image, at, say 1280x960, and down-sampled it to 640x480, we could have gotten much smoother, anti-aliased images. However, the memory constraints of the ISE and VINE computers prevented us from exploring this possibility.
Appendix: Pictures of the Final Movie

Fig1: Exploding effect

Fig2: Exploding effect

Fig3: Motion blur effect

Fig4: Motion blur effect

Fig5: Flyby

Fig6: Volcano