2018-12-01

The Bird and The Fish: Motion Field-Based Frame Interpolation in the Context of a Story

Thomas Sterling Ellsworth
Brigham Young University

The Bird and The Fish: Motion Field-Based Frame Interpolation in the Context of a Story

Thomas Sterling Ellsworth

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Seth Holladay, Chair
Parris Egbert
Mark Clement

Department of Computer Science

Brigham Young University

ABSTRACT

The Bird and The Fish: Motion Field-Based Frame Interpolation in the Context of a Story

Thomas Sterling Ellsworth
Department of Computer Science, BYU
Master of Science

Animating by hand can be a long and challenging process in part because of the necessity of drawing every frame by hand. 3D animation media minimize this problem with the use of automatically interpolated frames, but despite significant research no universally acceptable techniques have been demonstrated for 2 dimensional interpolation. In this paper we explore computer-assisted optimizations to the animation pipeline. Specifically, we utilize 3D motion fields to create more realistic in-between frames for sets of 2D "key frames." We demonstrate our method by using it to create 2D special effects for a 30-second clip of an animated short film.

Keywords: In-betweening, motion vectors, 2D and 3D hybrid animation

ACKNOWLEDGMENTS

Every good thing about this thesis and my life has come from the Lord. He is the source of every good thing. But in my life as in most others, he often works through earthly angels, and I would like to acknowledge a few of them here.

I would like to first thank my advisor Seth Holladay. Seth has been a friend and mentor for many years. He read countless revisions and made time to give me notes during nights and on weekends. He is one of the most selfless people I have ever met, and has taught me by his example that to be a teacher is to serve other people.

Thank you to BYU and the faculty and staff of the Computer Science department. You have been my employers and mentors for much of the last 10 years and I am so much better for having known you and worked with you. In particular I would like to acknowledge Parris Egbert, who supported me through much of my journey as a masters student, and Jen Bonnet, who always helped make sure that I was on the right track.

A special thank you to Sean Flynn and Jeremy Oborn, who shared the lab with me and frequently took time out of their own studies to help me with mine. This would not have been possible without you.

Thank you to my family. Thank you for all the early morning phone calls to wake me up before work to write a few pages. Thank you for all the reminders and the goal setting. Thank you for being my greatest support.

Thank you to all the friends who have supported me through the years. Some of you encouraged me in my decision to return to school, some of you were the inspiration for this film, and some of you just made life and school much more joyful.

Last of all, thank you to the hundreds of people who let me show them a little video about a bird and fish.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rise of 3D animation in recent years and the decline of hand-drawn animated feature films in the United States, the future of traditional hand-drawn animation at one point seemed quite bleak. However, in recent years, this form of 2D animation has made a resurgence as an important stylistic option for animation.

Japan has continued to revere 2D animation as an art form and as a result the highest grossing films in Japanese animation continue to be done in 2D. In 2016 the hand animated film "Your Name" became the 4th highest grossing film in Japan of all time, only barely trailing Disney's "Frozen".

Likewise, the studio Cartoon Saloon in Ireland has continued finding success using 2D animation crafting films like "The Secret of Kells" and Academy Award nominee "The Song of the Sea".

Another avenue for traditional animation has been in layering 2D and 3D animation. Many shows now combine traditional character animation with computer generated effects or backgrounds, a tradition that began early in computer animation's history with films like Golgo 13, Akira, and later in America with Rock & Rule. Today, Disney has used traditional animation for such films as the Academy Award winning short film Paperman, the Academy Award winning Feast, and their Academy Award nominated film Moana, where traditional animation was used for the tattoos of a main character named Maui. SPA studios has also used computer animation to assist in the creation of their upcoming hand-drawn film "Klaus".

Television shows, many of which are still traditionally animated, have picked up this trend, with 2D and 3D integration becoming very common in both American, Japanese, and European television. Of the current most popular animated shows according to IMDB[2], the first 19 are a form of 2-dimensional animation, with many of them being hand-drawn key frame animation.

In addition to financial value, hand drawn animation has an important artistic and stylistic role. However, traditional animation can be a laborious and expensive process, particularly for small studios and hobbyists, and will require new tools and improved workflows in order to meet the demands for skilled artists in the near future. The area of Non-Photorealistic Rendering in Computer Graphics has improved the ability of computer-generated animation to mimic some styles of traditional animation, but has not achieved enough success to replace it except in a few instances.

A hand-drawn animation consists of roughly 720 drawings per layer per minute[23]. The animation begins as series of drawings of extreme or important poses called "key frames" (Figure 1.1). These drawings capture the essence, motion and critical moments of the performance. Key frames often constitute only one-fifth of the total drawings. The remaining drawings are called in-betweens. In large studios, creating the in-betweens is often done by an assistant, rather than by the original animator. These in-betweens can be the source of great expense and a considerable amount of time in creating an animated movie. They can also create a barrier to entry for the animation hobbyist.

Automating the in-betweening process, though still unsolved, has been the focus of significant study over the course of the last 60 years. While many advances in in-betweening have been made, most results are unsatisfactory as they are often have trouble applying the core principles of animation. This leads to non-physically-based motion and a lifeless quality due to a lack of human-introduced imperfection in the motion. The current modern industry approaches to automating the in-betweening process are found in production software such

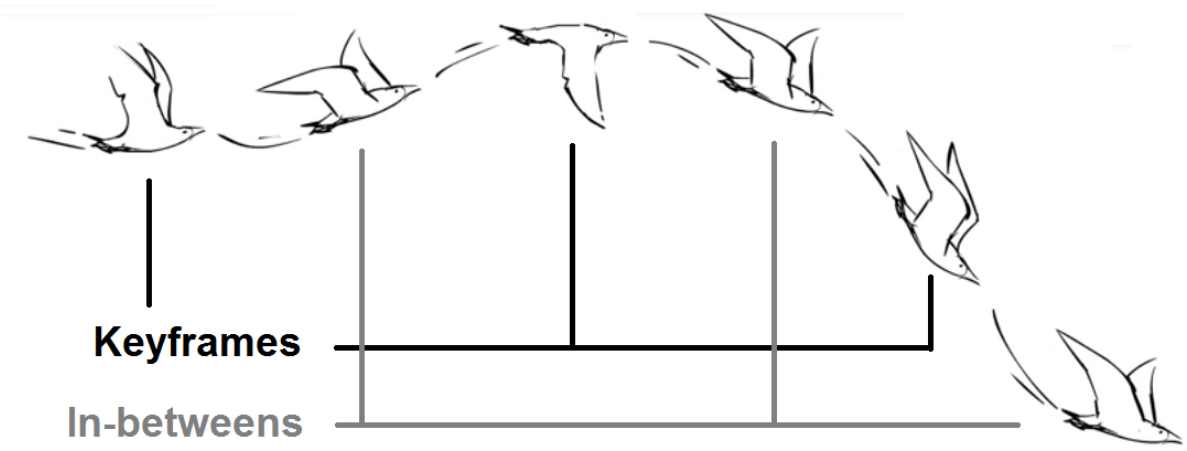

Figure 1.1: Only a portion of the drawings in an animation are key frames. Key frames are key poses whereas in-betweens represent non-linear interpolations of the key frames.

as Toon Boom Harmony[1], CACANi, and RETAS (now CLIP STUDIO PAINT), but aren't frequently used due to these unsatisfactory results.

In 2012 Disney Animation used a new approach to computer automated in-betweening in its mixed 2D/3D Academy Award-winning short film, Paperman [16]. The method successfully generated aesthetically pleasing in-betweens by first creating a 3-dimensional representation of the scene and then combining data from that representation with 2D key frames. Their method uses the motion of the 3D objects in the scene to more convincingly interpolate the motion of key frames or to extrapolate new frames into the future. This hybrid approach, created by the research team at Disney and developed into the software package "Meander[3]," provides a basis for significant advancements in the field of in-betweening, as well as the generation of 2D animation in general[4]. However, released materials for the program did not conclusively demonstrate the ability to handle cases where the 3D model overlaps the lines or where multiple lines overlap.

Our paper extends Disney's hybrid approach by improving upon Disney's method of storing animated motion data. Like Disney, we use the velocity of an animated 3d model. This velocity is stored as a set of 3-dimensional vectors representing the Cartesian motion of each vertex. However, We have changed the way that these "motion vectors" are stored by

keeping them in a 3-dimensional grid rather than a 2-dimensional image. Because of this change, our method is able to manipulate the 2d input key frames in new and important ways using 3D data. For example, our method allows for the independent movement of overlapping lines from a key frame, something that appears to be missing from Meander's results. Furthermore, our work allows us to move lines that are temporarily invisible to the camera, but still exist, in the 3D representation, such as when a car passes in front of a character. This is something also not demonstrated in previous results.

We also utilize this motion data to create 3D fields of motion, representing a character's per-vertex motion over time through a space. This allows us to create 2D line motion that is dependent on higher-order motion, such as the character's overall motion arc.

We demonstrate these methods by applying them to the creation of a small film clip that mimics a significant portion of the full production pipeline of an animated film. We designed, modeled, rigged, and animated characters to provide a complex test bed for our method. This includes overlapping motion vectors and occluded lines. We also define an aesthetic to our film independent of the method to demonstrate that our method retains artistic flexibility.

We will utilize our 3D motion fields to automatically generate a 2D contrail effect for a short film clip. These drawings will utilize a drawn key frame and motion vectors derived from a 3D animated character to extrapolate a 2D contrail effect for a bird.

Chapter 2

Related Work

Ed Catmull outlined some of the problems facing the computer-assisted animation research community in 1978[9]. The paper included a large section exclusively on automatic in-betweening, which he called the "focal point" of most computer-assisted animation systems. One of his conclusions was that "The principal difficulty [in automated in-betweening] is that the animators' drawings are really two-dimensional projections of the three-dimensional characters as visualized by the animator, hence information is lost."

Despite these challenges, "tweening" has continued to be a well-studied area of computer graphics. The results of this research have failed as yet to find a visually pleasing correspondence using only two key frames as input, and no solution has found widespread industry use or acceptance.[12]

One of the critical difficulties of the automated in-betweening problem is correlation: finding an appropriate mapping between control points of two vectorized "key frames"[23]. A powerful graph-based minimization solution that continues to find success today was developed in 1992 by Sederberg et al.[20] Sederberg models 2D lines as pieces of wire and then computes the correlation of control points that would require the least energy to bend and stretch from one key frame pose to the other. In 1993 Sederberg et al.[21] extended their method to reduce inaccurate deformation in their interpolation functions. They interpolated the relative angles between sets of control points and the lengths of line segments rather than interpolating control point position. This work was used by Disney Animation in their 2010 paper on computer-assisted tweening for tight in-betweens[25]. They designed a

fully-functioning in-betweening system. The system is primarily meant for in-betweening two key frames that are very similar, but improves on all previous work in the field by interpolating control points of vectorized drawings along natural curves.

Michael D. Smith[22] continued the work of Sederberg et al.[21] by creating a commercial in-betweening software package that combined the improved correlation and interpolation functionality with a classic rig-based solution to in-betweening. While this approach can produce impressive results, rigging the drawing represents additional overhead and resulting animations often look artificial.

Noris et al.[18] solved an adjacent problem, converting implicit raster images into parametric vector curves that are then usable for in-betweening.

Petrovic et al.[19] used an inflation algorithm to turn a 2D image into a best-guess 3D model and allow a background image with a 2D character animation. Jain et al.[14] expounded on this by generating 3D animation data for a set of 2D drawings using user-driven segmentation and motion capture. This allows a 2D drawing to interact with 3D forces and simulations. This is the inverse of our problem: Jain is using existing finished 2D animations to create 3D animation, rather than using 3D motion to create 2D drawings.

Bregler et al.[7] further exploited motion capture techniques, re-targeting motion from 2D animated samples onto 3D models, photographs, and other 2D shapes. While this technique is robust, it can only imitate existing animation. Hornung et al.[13] similarly re-targeted motion capture data onto still images, a very similar, but less complex, version of the motion capture to the in-betweening problem. This approach often suffered from undesirable deformation at points of rotation.

Agarwala et al.[5] approached the problem from a computer vision standpoint, utilizing in-betweening to semi-automate rotoscoping in 2D video and extrapolating the motion to animated curves. Benard et al.[6] added temporal coherence to this method, generating a series of stylized in-betweens from the 3D motion of a simply-lit and textured model and a style example image. Browning et al.[8] applied this principle to fluid effects, using video of

a fluid simulation to drive a stylized 2D water animation. While our method bears strong resemblance to Browning's method, it uses 2D data as the input and, as such, is unable to handle situations where one object occludes another.

The "Meander" software developed by Brian Whited et al., though not yet publicly available as a technical paper, is the basis for this work. Because Meander uses 3D animation data to drive the in-betweening process, the computer generated in-betweens will demonstrate the principles of animation to the same degree as the 3D animated input. Furthermore, this allows 2D artists access to many powerful 3D animation tools. Some of these tools, such as the basic in-betweening tools used in 3D animation, add powerful features by having full knowledge of the 3d scene description, as opposed to 2D tools which only have access to the final projection.

In "Meander", Whited et al. used full 3D data as the input to the system. A partial derivative is taken of the position of a 3-dimensional vertex over time to create a velocity value. This process is repeated for each vertex in a 3D mesh and then rendered by a 3D camera in order to generate a 2D image containing the object's velocity data. This data is then dilated as a safety net for the 2D lines.

Our method will expand on the Whited and Browning methods by using raw 3D motion vectors rather than flattened, rendered images as the data input to move 2D key-framed drawings around and automatically create in-betweens. This will allow the 2D input drawings to be moved in 3D space, allowing for the lines to overlap and enable automatic depth ordering. Our method will also continue to correctly advect lines that are attached to geometry that has been occluded by another mesh. This enables hybrid animation systems to operate in new situations and using fewer 2D key frames as inputs. Further, we will derive special effect animations from the motion of other objects, rather than needing to create and animate a direct 3D proxy for the effect.

Chapter 3

Thesis Statement

We can create 2D special effects animations from a set of animated 3D motion vectors and 2D key frames, which we store in a new way to support overlapping 2D input lines, occluded input motion vectors, and motion paths over time. We demonstrate this method by using it to make a set of special effects for a short animated clip.

Chapter 4

Generating Motion Data

One significant aspect of the problem we address is determining what motion data to produce from the 3D animation and how to generate it. We leverage this data to take advantage of some of the strengths of 3D animation and simplify generating the desired 2D style animation. We initially began with the more typical approach of creating a 2-dimensional image of vector motion data at each frame of the animation. From there, we utilize the Houdini software package to generate a 3-dimensional representation of the motion data and write it to a file format we can use in Python.

Using a 3D grid of motion data as the basis for our method allows us to treat input 2D lines as if they existed in 3D space, naturally supporting the use of overlapping lines or lines being driven by motion data that is occluded to the camera.

Fortunately, our initial goal of writing 2D velocity data from 3D animation is used for other common artistic 3D workflows. For example 2D images of motion data rendered from a 3D camera are frequently used by compositing workflows to approximate and control motion blur. Each pixel in this type of image may either store gray-scale data to represent motion in a single dimension, or a color to represent motion in 3D Cartesian space. In Browning's work[8], for example, gray-scale 2D images generated by an external physically-based fluid simulation are used to represent motion in the X and Y axes.

Likewise 3-dimensional fields of velocity data are frequently used in volumetric simulations such as when generating physically-based representations of 3D fire, water, dust, or other phenomena. In [4] (Paperman) this kind of 3D motion data is collapsed into a 2D

image like that used in [8], but instead of storing gray-scale values, colors are stored. These colors contain positive or negative values in each of the 3 channels, and thus can represent both forward and backward motion in each dimension.

4.1 Camera-Based Velocity Gradient

Our first attempt was to generate a 2D image representation of the motion vectors similar to that of other methods. This means flattening the 3D information in the scene into a 2D image.

We chose to do this by "rendering" a view of the world through the camera in Maya. While any orthogonal view of the space would have worked, a camera gives us the ability to use Maya's animation system to sync our camera's movement with the 3D characters.

4.1.1 2D vs 3D gradient images

Maya provides two default options for generating camera-based motion vectors through the MentalRay renderer that comes packaged with versions of Maya prior to 2017. One option is a normalized 2-dimensional image and the other is a raw 3-dimensional orthographic snapshot of motion in space.

Initially we tested with the normalized motion vectors. This allowed us to quickly pull pixel values from a file stored in the .png format and apply them. However, because one of the goals of our experiment was to differentiate between overlapping lines by including the third dimension, we decided to use the 3-dimensional, non-normalized images. Because these images contained negative data with high levels of precision, we needed to store them in a more precise file format than a simple 8-bit depth .png.

4.1.2 OpenEXR

Most commonly-used image formats are best suited to represent a discrete number of positive values. Often each pixel stores an integer from 0 to 255. Some images provide greater detail

by allowing 16- or 32-bit depths per pixel, but because many of our values were going to vary significantly in magnitude (values in our example often range from 1ê-7 to .1) and can be either positive or negative, we needed a way to efficiently store floating point data for each pixel.

Instead of storing our data using scaling and offsets with a 32-bit pixel depth, we chose to utilize the OpenEXR file format developed by Industrial Light and Magic[15]. The format was originally developed to store 3D floating point data and as such was perfect for our needs. This gave us the ability to store our motion data with multiple channels and high precision. This was particularly important because any lack of precision in the motion data would propagate errors across time steps until they derailed the simulation.

Fortunately, Maya has built-in support for the OpenEXR file format, and we were able to export directly from Maya. The challenge then became how to import the data for use in Python. To overcome this challenge we had to use a set of open source bindings for the OpenEXR library and write a custom .exr importer. The importer converted the OpenEXR data into NumPy arrays for use in our simulation.

4.1.3 Sub-Stepping

Because the velocities contained in our animation were so high, a great deal can occur in a single frame (one twenty-fourth of a second). In order to achieve greater precision, we utilized a sub-stepping technique commonly employed in 3D simulation. Put simply, this means super-sampling in the time dimension, generating multiple time samples between each frame.

The trade-off is the need for more processing time. With ten sub-steps, we must process ten times as many frames. Because our method is neither time-constrained, nor prohibitively slow, we opted for the greater precision.

4.1.4 Dilation

Even with sub-stepping, our method is still a discrete linear approximation to a non-linear motion curve. For example, in a case where our method was tracking a vertex near the silhouette of a character is decelerating, our method, which only moves linearly per time step, could overshoot the bounds of the character's motion vectors on the next frame. If the deceleration was severe enough, our method could become completely orphaned from the motion data for the remainder of the simulation. Thus, sometimes a pixel will overstep the model's actual movement. In order to protect against these errors, we dilated the motion vectors outward from the model's surface.

This technique was also used in Meander[4]. They calculate the motion vectors by creating topological cylinders of the mesh, calculate the tangents relative to the viewpoint of those topological cylinders, and then use these tangents to calculate the normals of the model along the silhouette at these tangents. These normals are then used in combination with per-pixel motion vectors to extend the silhouette of the motion vector image. This technique creates a "safe zone" of pixels around the character with similar motion to that found at the edges of the current motion vector image. Any line that errs a few pixels outside of the silhouette is likely to be recaptured later by the simulation.

Our method used the same idea. We created a small Python library based out of NumPy to do the dilation. We initially attempted to use existing OpenCV implementations of a 3-channel image dilate, but they lacked the flexibility we needed to handle some common edge cases.

One such case is demonstrated in Figure 4.1. Choosing the which motion vector to copy to a dilated pixel in an area of the surface with convex curvature can become ambiguous in a naive approach to dilation. Imagine, for instance, the pixel outside the bounds of a 90 degree corner. Neither corner can accurately represent that motion that pixel should have alone, and since most of these types of curves occur in areas of significant deformation, they

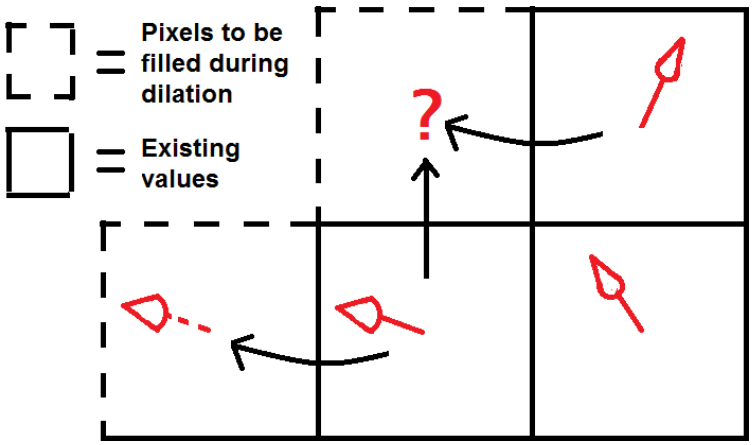

Figure 4.1: External cells with only one neighbor inherit their neighbor's value. Cells with multiple neighbors benefit from averaging. Red arrows represent velocity vectors.

tend to frequently have opposing or diverging motion. Thus each such pixel may be more accurately portrayed as the average of all surrounding pixels that contain a motion vector.

In order to do this efficiently in Python, we developed a method in which we created a binary representation of the image and dilated it by one pixel as shown in Figure 4.2. We then subtracted the original pixels and iterate through the difference, using the original image as a look up to generate an average of all non-zero 4-connected neighbors.

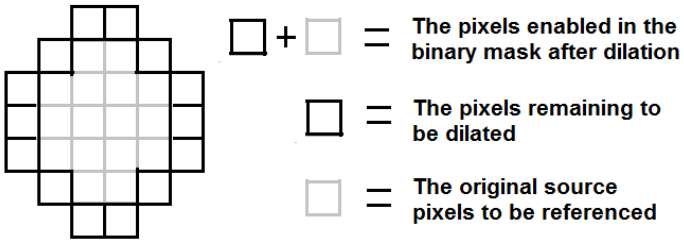

Figure 4.2: Gray pixels (the original image) are subtracted from a binary image which is used as a selection mask. The values of remaining black pixels (the new pixels due to dilation) are then computed using the original image. This process is repeated for each iteration of dilation.

This process can be repeated as many times as necessary to grow the dilation while continuing to average with each progressive layer. Because this operation is time consuming, we initially limit the size of the dilation to the greatest velocity in the image, converted to

pixel space and increased by 1, as a buffer against minor errors. We developed equation 4.1 to govern this dilation, with delta representing the number of pixels to dilate the motion data and the function representing the motion data. This provides us with sufficient samples for our advection method, described in chapter 5. The number of iterations can be correspondingly increased if the 2d key frame lines are drawn farther outside the silhouette of the mesh.

$$\delta = max\left(f\left(x_1\right), f\left(y_1\right), f\left(z_1\right), ..., f\left(x_n\right), f\left(y_n\right), f\left(z_n\right)\right) \tag{4.1}$$

4.1.5 The Problems with Gradient Images

A gradient image is an image representation of the velocity of objects in the scene in pixel space. Gradient images within our method also bring various challenges. One challenge is the conversion from 3D world space units to 2D pixels. A motion vector image in Maya and most other 3D software is generally stored as the distance traveled by a point on a 3D model's surface. This vector is stored in some unit of measure internal to the software. These are generally laid out based on a global Cartesian origin and all objects in the file are transformed relative to this origin. In order for pixel transformations of 2D lines to be useful, we need to have a conversion between these "world space" units and the canonical view port of the 3D camera.

However, with this method, the conversion of a motion vector distance to pixel space can be difficult without significant effort. With an orthographic camera, converting is simply a matter of calculating a trivial ratio between the orthographic window size settings in Maya and the pixel resolution of the render. In 3D projection space, the problem becomes more difficult, as the number of world units traversed per pixel varies with the depth of the object.

In our system we chose to focus on other aspects of the problem and simply satisfied this problem by inserting a best-guess ratio based on observation of the exported motion data.

A problem specific to the use of a 2D image as storage for the motion data is the ambiguity of overlapping pixels. In other methods, if at any point in time the motion data of objects in the scene overlaps from the camera's viewpoint, such as a hand passing across a character's body or when a character spins or turns, only motion from the vertices closest to the camera would be stored. Occluded vertex motion is lost as the 3D space is flattened into an image. In this case, the lines corresponding to this motion are often occluded as well, but this causes a problem when those lines reappear in a different location than the original point of occlusion.

An example would include a humanoid character turning in a circle. Imagine standing facing someone as they turn. As the person rotates, their arm disappears to our view on the one side of their body and then, moments later, reappears on the opposite side of their body. It is important to note that the shape and position of the arm has changed during that time, so correlating the returning motion data with the original lines is difficult.

Neither Browning et al.[8] nor Meander address this issue, limiting the ability of the technique to handle more complex and interesting motion, or, as in the case of Browning et al., restricting the space of the technology to only 2D motion.

Our solution to this problem was to move to a storage format for the data that could support motion data at multiple depths. Doing so allows our method to support both occluded and overlapping lines and shapes.

4.2 Voxel-Based Velocity Gradient

In order to solve the ambiguity along the Z axis, we moved from a 2-dimensional image representation of the motion data to a 3-dimensional voxel representation. "Voxel" is a term similar to pixel that refers to a discrete element of volumetric 3D space, sometime referred to as a "cell" in a 3D grid. Voxel grids are frequently used in special effects for 3D graphics, storing velocity, density, and other data for use in 3D simulations.

4.2.1 Generating Voxel Data

Maya has no native support for generating voxel representations of motion data, so in order to generate usable data, we had to programmatically generate it in Houdini. To do this we used central differencing, a common method for creating gradients. Central differencing is defined by the equation:

$$\left(\frac{\alpha u}{\alpha x}\right)_i \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{4.2}$$

In our case the equation uses 2 positions of the vertices of the mesh at each time step as the input.

4.2.2 OpenVDB

In order to transfer the voxel data from Houdini to our simulation, we first attempted using Dreamworks Animation's OpenVDB file format[17]. Open VDB provides a set of algorithms for storing and modifying sparse volumetric data in a discrete 3D grid.

We wrote VDB data from Houdini using its built-in OpenVDB write node and imported it using the provided Python bindings. We enjoyed multiple advantages by using this method. The files output by OpenVDB were generally very small, meaning that we could store large amounts of 3D data in a few hundred megabytes. Likewise, the write and read times on the files were relatively efficient, with dilation being the largest bottleneck of the system, followed by writing and reading the VBD files.

However, as we tested our method in OpenVDB we discovered because of OpenVDB's dynamic scaling it was often difficult to ensure that indices would remain constant. We believe with further work OpenVDB could be a good storage format, but we were eventually able to meet the needs of our system using a simpler method.

4.2.3 Houdini to Python

When our implementation of an OpenVDB importer was unable to meet our needs we moved to a more direct, but less compact, data alternative. After exploring several other options, we decided to export directly from Houdini to text-based data files, already formatted for Python, using a built-in function in the Houdini API for this purpose.

This implementation gave us some advantages we lacked with OpenVDB. It was simple and easy to read; simply casting the string file into a NumPy array was all that was required to read in the data. The data was also consistent; we were able to verify that the voxel values in specific key voxels matched identically the voxels as seen in Houdini across multiple frames.

4.2.4 Velocity-Based Dilation

In the OpenVDB format, file size is dependent on the amount of data in the voxel grid. Similarly in the native Houdini representation, the write speed is directly correlated with the amount of data in the grid. In order to minimize the amount of data we are storing, we adjusted the way we determine the number of iterations to dilate the data.

Up to this point, we had been dilating the data using our custom algorithm in Python. However, with the larger data set, our Python implementation had become slightly slower, and our programmatic solution in Houdini for generating the 3D voxel space also provided an implementation of this same dilation algorithm. Thus we switched to Houdini's implementation of dilation and wrote a series of expressions to implement our dynamic dilation in the Houdini node network. We set the number of dilation iterations to be 1 plus the maximum magnitude of the velocity field.

4.2.5 Results of Velocity Field Generation

At this point we had a valid motion field for advecting points in 3D space, already an improvement on previous methods which relied on a 2D image for velocity data. This allows

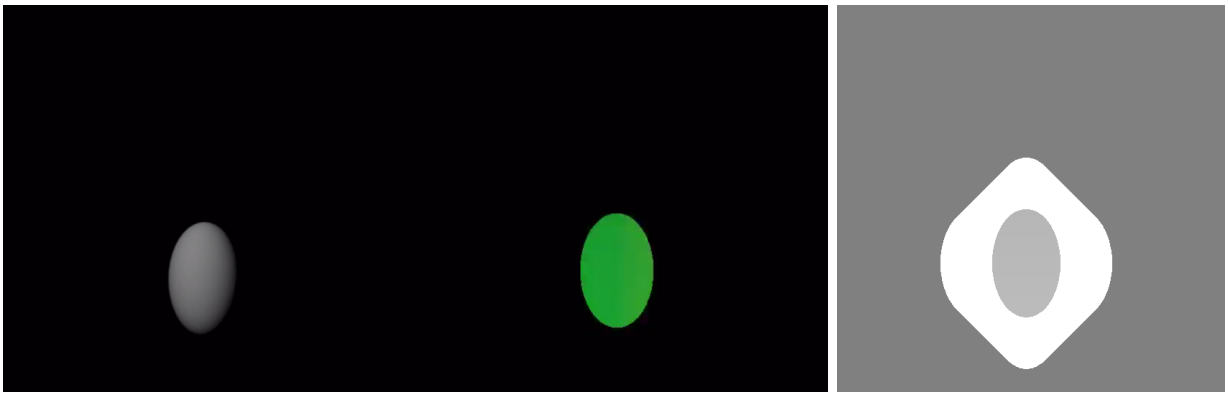

Figure 4.3: Initially we tested this method on a simple sphere. After having animated the sphere as a bouncing ball, we exported its 3D motion vectors and and dilated them as previously described.

us to advect overlapping points on the image differently as long as they have different values in Z relative to the camera viewpoint. We had dilated the motion field based on the velocity of the mesh and imported the voxel data into Python for use in generating 2D images. An example of this process is shown in Figure 4.3

4.3 Generating a Motion Path for External Drawings

Given a way to store the 3D motion data, we came up with a way to allow input lines to be affected at each frame by both present and past motion data. As our contrail moves it will follow the path of the character's motion, which is made up of the character's current and past positions. Allowing control points farther outside the silhouette of the character to be effected by past time step data creates the trailing effect we would like to see in a contrail or slipstream.

To do this, we created a moving window of the 3D volume. This window is simply a 3-dimensional grid of velocity vectors. Each time step, the values of the current volume are shifted to match the motion of the camera and new values are added. We chose to add the values as replacements, rather than doing averaging, as we do not want past frames interfering with the current frame's motion. Figure 4.4 visualizes a 2D slice of this window. The amount to shift the window each time step should be based on the movement of the

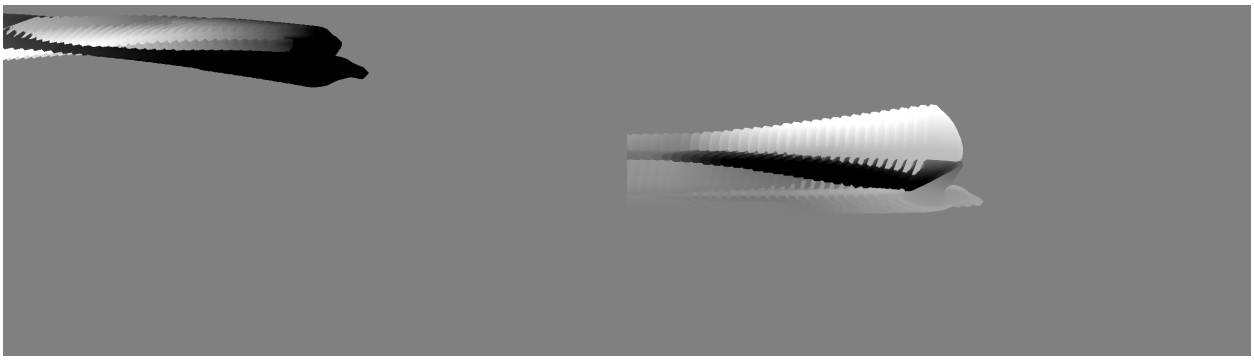

Figure 4.4: A flattened visualization of the Y movement stored in the voxel grid. Values are centered on 50% gray, with darker values being negative and lighter values being positive.

camera, but in our case we simply defined a user parameter, since the camera movement is constant across all frames.

For larger time steps with greater distances over time, blending would need to be done on the convex envelope of the two volumes to address unset values. However, this should not be typical, as our example already contains high velocity values and relatively few time steps.

Creating this volume window containing past data allows some special effects, such as the contrails displayed in our method, the ability to be affected by the second derivative of the motion, allowing for the shape of our drawings to follow the curvature of the character's path. This could be particularly useful for any trailing effect such as hair, cloth, speed lines, or contrails.

Chapter 5

Line Advection

Having generated an acceptable and accessible representation of the characters' motion, we next need a way to utilize this motion data to accurately move the hand-drawn input key frame. This includes choosing a way or representing lines in our system. Most drawing software shares a similar user interface, but the way they store 2D line data may vary significantly. Choosing the way in which line data store can have a dramatic impact on the types of problems that need to be solved in order to advect the lines in an aesthetically pleasing way.

5.1 Line Representation

Generating 2D line drawings requires storing the lines, mutating them to new interpolated positions at each frame, and outputting them as viewable images. The two most common ways to store lines for graphical purposes are as either raster or vector graphics.

Both representations, vector and raster graphics, are commonly used in the world of digital art and animation. Vector graphics are a parametric representation of lines and shapes represented by control points or equations, whereas raster graphics are defined as a pixel array, such as in a pixel-based image.

Different representations are used by the two different papers on which our research is based. Disney relies on a vector representation, having pioneered an advanced raster to vector conversion system in 2012[24]. Meander is the software container for this algorithm,

Figure 5.1: Several frames showing the progression of our raster test.

along with the artist workflow for creating vector images. Meanwhile, Browning et al.[8] used a raster representation, even using hand-drawn images as their input.

5.1.1 Our First Attempt with Raster Images

Raster images are the most common 2D image representation. They are the medium of most professional animation artists and often the most faithful representation for digitized traditional 2D artwork. For example, when capturing animated drawings, animation studios traditionally drew on paper and took photographs of the result. Later, as computers were introduced, these photographs could be scanned. The resulting scanned images are most often raster representations such as .jpeg or .png. As a result, we resolved to first attempt a naive approach, implementing our method with a raster image.

We created a 3D animated sphere and exported 2D motion vectors with dilation using the methods detailed in chapter 4. We drew a single 2D key frame of a ball on top of the sphere and ran our simulation. Our 2D circle then followed the motion of the sphere whose 3D vector field was driving the animation, seen in Figure 5.1.

These initial results (Figure 5.1) demonstrated significant divergence between pixels. As pixels advected imperfectly in a discrete space, gaps begin to form in the image. These gaps widen as the displaced pixels inherit bad motion data from their new position and the positional errors compound over time. Some of these problems were simply due to imperfections in our method at the time, and others were due to the nature of working with discrete representations of continuous data. Browning et al.[8] addressed this issue by using

a complex blending algorithm, but the results still frequently look blurred or the motion of the pixels often appears abnormal when compared with the overall flow of the animation.

One possible approach we did not explore here is to apply "reverse Euler's method", a technique often used in 3D fluid simulation, where each empty pixel is mapped back to the previous image, rather than from the previous to the new image. This allows multiple pixels to map back to the same origin pixel and alleviates gaps.

While the contributions of our method apply equally well to a raster based system, we chose to move to a vector graphics-based approach so as to focus on our specific contribution to the field rather than dealing with the additional challenge posed by pixels maintaining temporal coherence across all other pixels.

5.1.2 Using Bezier Curves

Moving to a vector graphics-based approach allowed the lines to stretch and bend slightly without the tearing found in the raster-based solutions. We chose to use Non-Uniform Rational Basis Splines (Equation 5.1) also known as NURBS. This subset of B-spline curves provides generally desirable interpolation between control points that allows us to represent complex shapes while not having to worry about maintaining a tight temporal coherence.

$$C(u) = \sum_{k}^{i=1} \frac{N_{i,n} w_i}{\sum_{j=1}^{k} N_{j,n} w_j} P_i = \frac{\sum_{i=1}^{k} N_{i,n} w_i P_i}{\sum_{i=1}^{k} N_{i,n} w_i} \tag{5.1}$$

To further alleviate the problems found in our raster test, and to ensure control points did not wander in the motion field, we stored each non-zero pixel's position as a float separate from the image representation. That way, a pixel's position is not limited to pixel positions and floating point remainders of the advection calculation are not rounded or truncated. Such errors, if left untended, could cause a significant deviation over time.

5.2 Moving Control Points

Using B-spline control points reduces the need for a node to be aware of its neighbors and we can focus on maintaining parity with the 3D model's motion. Because we are in 3D space, we have access to perfect information about the movement of the character. However, without exporting the animation in a parametric format, we must work with discrete representations of a continuous system. As a result, no attempt to match the motion of the 3D data will be perfect, and so our effort was to best approximate the motion for each sub-step.

Since we are using our motion data to move novel points that may not actually lie on the mesh, we instead must extrapolate how a novel point would have moved as part of the model. To achieve this, we use the control point positions found in the input key frame as our initial positions. We then use the motion data generated by the 3D animation to approximate a derivative of the points' positions over time. This is an ideal candidate for applying numerical methods for approximating ordinary differential equations. We chose to solve this as an initial value problem (IVP).

5.2.1 Euler's Method

The first method we attempted was a basic Eulerian stepping approach, advecting each control point by the value of the derivative stored at each time step. This method assumes that the values to be calculated are equally distributed as defined in Equation 5.2

$$t_i = a + ih, \quad for\, each\, i = 0, 1, 2, ..., N. \tag{5.2}$$

In this equation h is called the step-size and in our method is simply the frame or sub-step of input. a is some initial time value and t_i is value of the next time step. This works because our sub-steps are already equally spaced in time by Maya and Houdini's exports.

Euler's method is then uses Equation 5.3 to calculate the next time step.

$$w_{i+1} = w_i + hf\left(t_i, w_i\right), \quad for\, each\, i = 0, 1, 2, ..., N. \tag{5.3}$$

In this equation w_i is the initial position of the current control point being evaluated (or the value in the previous time step), and w_{i+1} is the new position. $f\left(t_i, w_i\right)$ is the derivative, in our case it is the change in position described by the 3D motion vectors.

However, this method proved to be insufficient for our needs as it would quickly deviate from the motion trail of the character on time steps with large derivatives. In these cases, Euler's method often overshot the actual motion, particularly in time steps where the character was decelerating.

5.2.2 Runge-Kutta

In order to improve the accuracy of our method and better approximate the motion of the 3D model we moved from Eulerian stepping to Second Order Runge-Kutta. Runge-Kutta significantly improved our accuracy and allowed our method to better adapt to abrupt changes in the motion of the character.

To even further improve our accuracy, we eventually moved our method to the more advanced Fourth Order Runge-Kutta. This method, commonly known as RK4, provided us with even greater accuracy.

We implemented each of these methods in Python. In implementing RK4 we used the following Equation 5.4:

$$
\begin{aligned}
k_1 &= hf\left(t_i, w_i\right), \\
k_2 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1\right), \\
k_3 &= hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2\right), \\
k_4 &= hf\left(t_{i+1}, w_i + k_3\right), \\
w_{i+1} &= w_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), \\
&for\, each\, i = 0, 1, 2, ..., N.
\end{aligned}
\tag{5.4}
$$

The Runge-Kutta Method uses a weighted average of 4 approximations of the slope to create a more accurate guess at the actual change in the function over the time step. Our implementation uses an approximation of the slope at the beginning, ending, and 2 midpoint slope approximations.

To test the accuracy of our method and our implementation, we created a key frame with a single vector point at the same location as the rear-most vertex of the tail. We then advected this point using the motion vectors and compared the motion of the control point with the actual vertex location after 1250 time steps (125 frames), significantly more than generally needed to in-between 2 key frames. The greatest resulting deviation was a distance of 1.4 units. The dimensions of the bounding box of the movement was 200 units along the longest edge, meaning that a deviation of 1.4 units was incredibly accurate for such a long number of time steps.

5.2.3 Interpolation

For control points that did not directly lie in the center of a voxel grid cell, we utilized a traditional tri-linear interpolation algorithm. This required us to specify in Houdini the location at which the values were stored in the voxel grid and then to match that in our own implementation. We chose the center of each cell as the seat of the cells' value.

Chapter 6

The Film

We created an animated short as a test bed for our method. The film provided additional challenges, as we had to ensure that our solutions held up under the difficulties of actual production and difficult edge cases. Additionally, the film provided an artistic goal to the research and a visceral demonstration of the results.

Though we only took a small portion of the film to it's final state in the process of our research, our production mirrored much of a standard animation pipeline.

6.1 Creating the Story

Before creating any 3-dimensional assets for the film, we formulated a story to drive decision-making and provide us with an artistic problem to solve. The story was created in Abode Flash and drawn by hand, settling on the current two-and-a-half minute version for our test.

In order to generate the most compelling story possible, we relied heavily on feedback from a multitude of sources, including story artists at Dreamworks Animation and other professionals. While the current version of the film's narrative is not yet finalized, its compelling narrative provided us with context for our method and an emotive scene as our test bed.

The scene we chose to test our work is an approximately 6-second clip of the main characters of the film, a bird and a fish, engaged in a synchronized dance near the surface of the ocean. This scene draws heavily from two similar scenes. One, the introductory video of

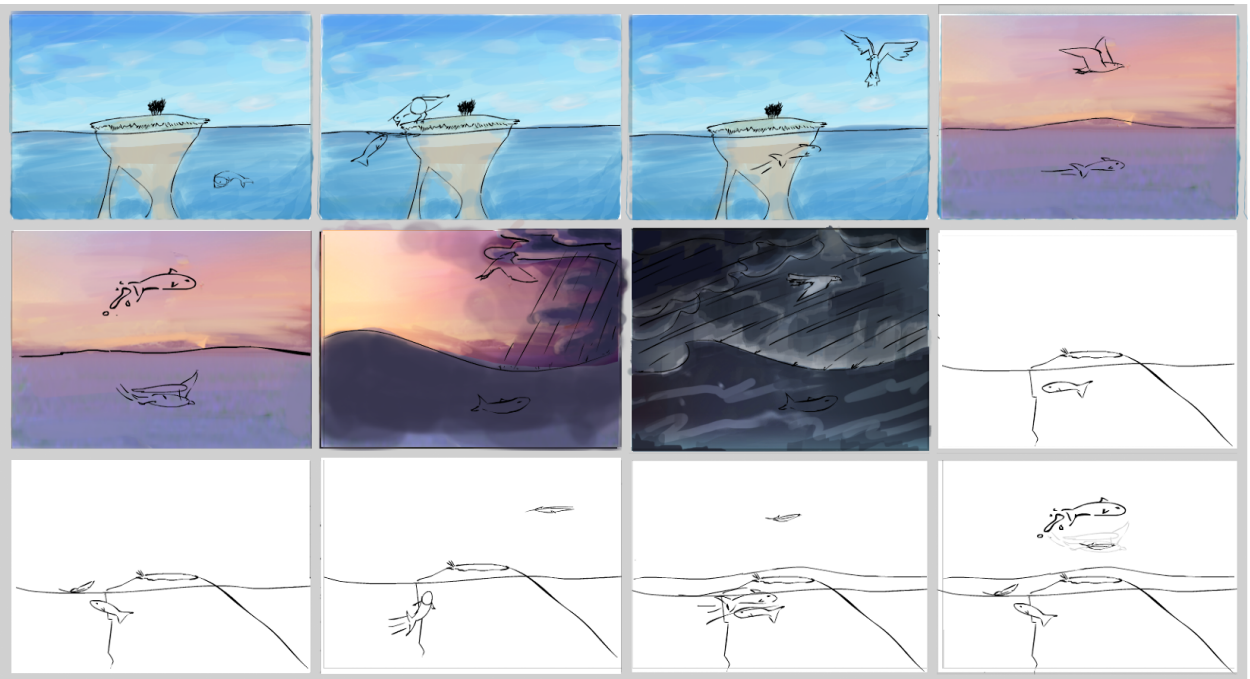

Figure 6.1: A sample of the storyboards and color keys for the short film.

an artistic video game called "Entwined," and the other, the "define dancing" scene from Pixar's WALL-E.

6.2 Creating the Look

We went through a rigorous artistic refinement process to develop an aesthetic that would be achievable, attractive, and artistically constrictive on the quality of our results.

We began by developing possible designs for the characters and the style of the background. Some of these designs were done by an undergraduate student studying animation. These designs were added to a body of reference material from other films and photography and an initial painting study was done by the principal researcher.

Next, a target final image was developed. The purpose of the image was to give a blueprint for the creation of the 3D characters and a visual target for the special effects of the contrails. It also allowed us to finalize a look for the background before painting a larger version.

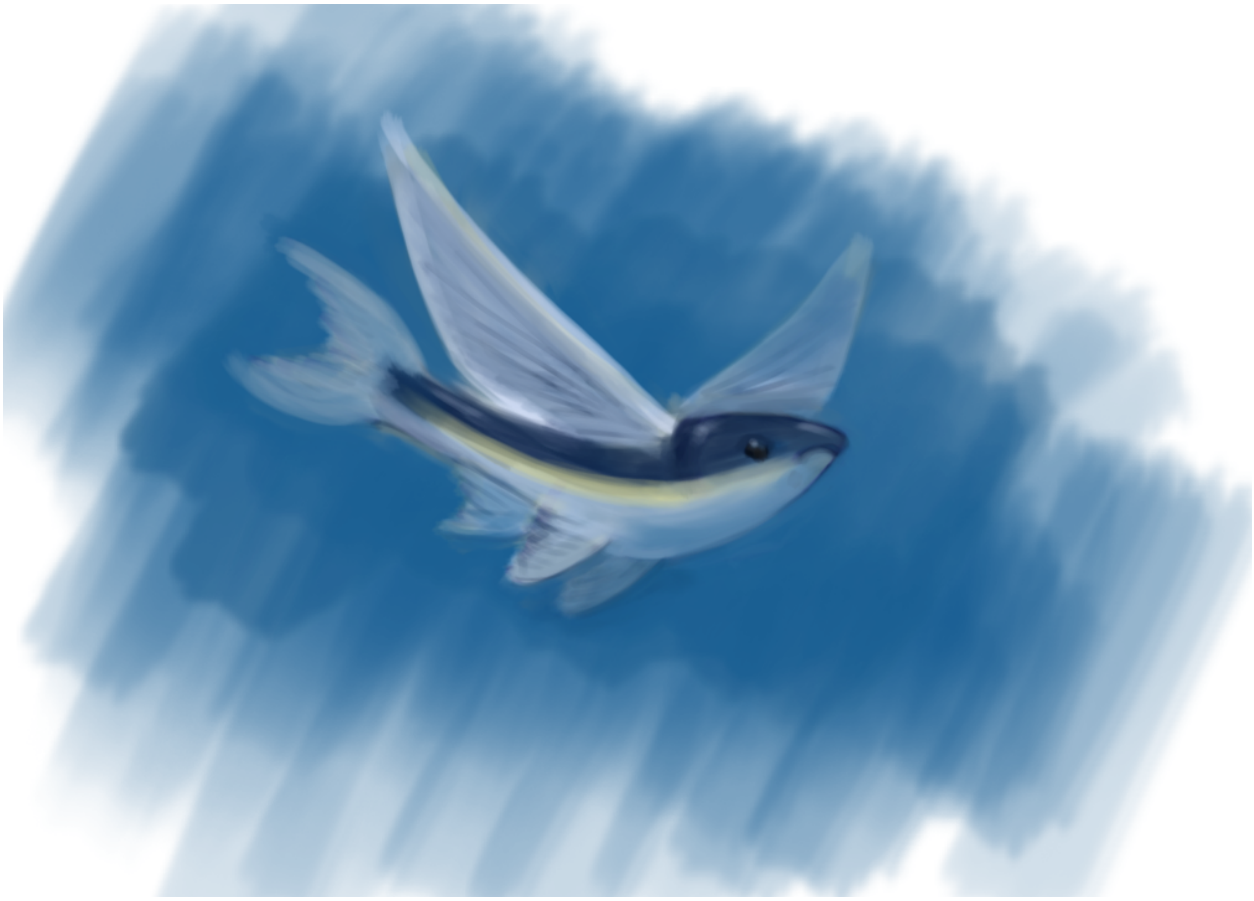

Figure 6.2: An early concept for the fish character.

Lastly, a 2-dimensional background "plate" was painted. Because the characters are moving across a semi-static world and the camera is consistently orthogonal to that world, were were able to represent the world as a 2D "matte" painting, similar to what is frequently used for distant backgrounds in CG films. In the end, we generated a 4000 x 60,000 pixels background painting in layers (Figure 6.4). These layers could then be parallaxed to give a greater sense of dimension (Figure 6.5).

6.3 Producing Usable Character Animation

3D animation of complex characters is usually founded upon several preceding processes. These processes primarily include creating an appealingly-deformable 3D mesh and creating a "rig" to simplify character motion.

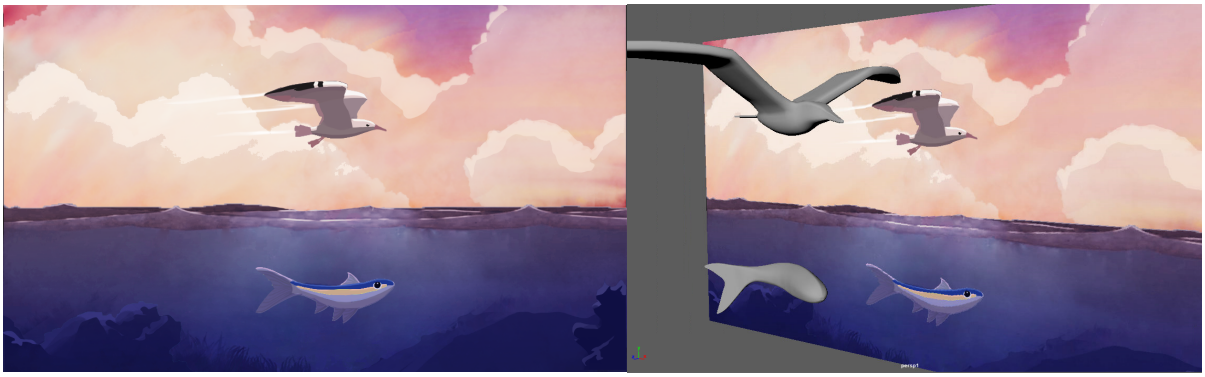

Figure 6.3: A painted visualization of a target final frame of the film next to the 3d scene during the animation process.

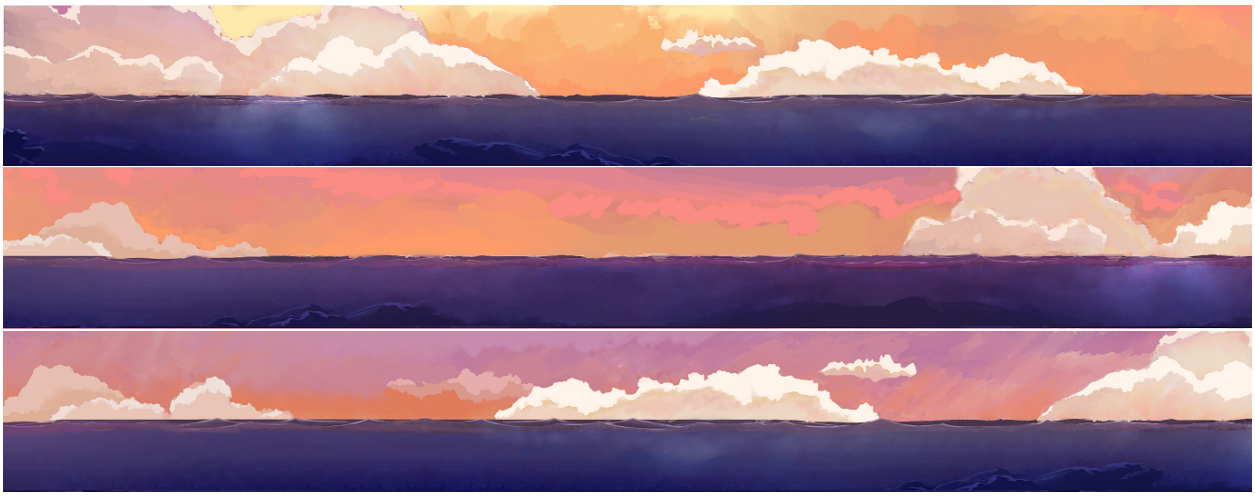

Figure 6.4: 3 segments of the final painted background.

6.3.1 Preparing the Characters for Animation

Modeling is the name given to an artistic workflow used to generate a large grouping of vertices (points in 3D space) that are organized to suggest a cogent volume. The vertices must be placed in such a way as to allow uniform curvature in areas of high deformation. (Figure 6.6)

The creation of these vertices must be combined with artistic decision-making, as creating a 3D model of a character is essentially projecting a 2D drawing into 3D space, much the same as sculpture. For this film in particular, modeling the fish character required many

Figure 6.5: The final background was broken into layers so that it could be parallaxed. Clouds, water, and sky all needed to move at different speeds to create the illusion of a real oceanscape.

iterations, as some ambiguities in the 2D drawing resulted in difficult artistic decisions in 3D space.

The result of the modeling process is a character model made up of hundreds, thousands, or even millions of vertices. Parenting the motion of these vertices under 3D transforms called "joints," a skeletal structure, can simplify the animation process. This process is called rigging.

Because rigging birds is frequently challenging, and highly successful rigs are often the result of research, we used a simplified custom rig based off of multiple common solutions. As a result, the rig for our bird character can experience some undesirable deformation but is sufficiently functional for our constrained short film.

By contrast, the rig for our fish character was simpler. We based our fish rig on the rigs used by a small animation studio that produces work for National Geographic[11]. It utilizes a deformer that constrains the fish's motion to a NURBS curve; several other layered deformers create secondary motion.

6.3.2 Animation

The dynamic nature of the characters' motion required by our story posed several interesting challenges when choosing how to animate. Character animation is represented as a transformation on each vertex per frame.

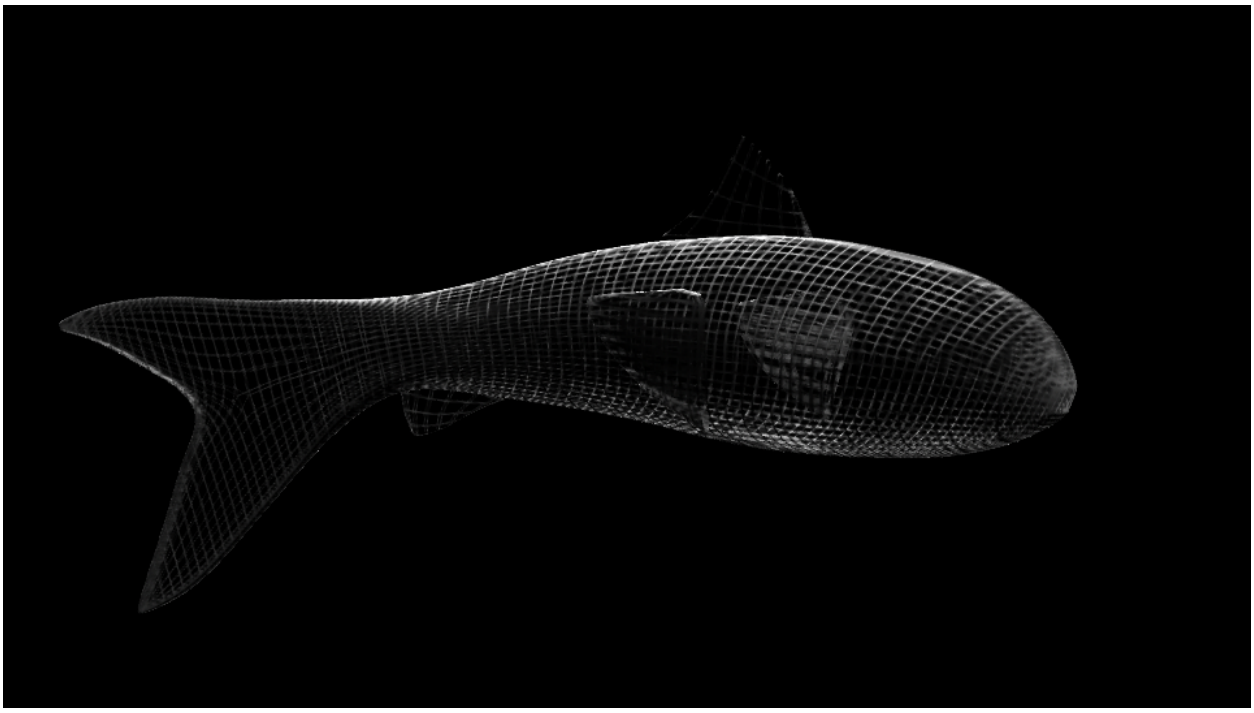

Figure 6.6: A visualization of a 3D "mesh".

This motion is displayed to the artist in several user interfaces. A common representation, and the one used predominantly in Maya, is a simple spreadsheet that displays positional data related to a selected vertex, joint, or control. This motion data can also be viewed over time as a series of parametric NURBS curves, in a separate window called a "graph view." In this view, an animator can modify the curvature of these parametric lines to affect the motion of the character. Thus, while the underlying output representation is traditionally discrete (the "frame" being the most common unit of discretization), the motion created by the user is at some point continuous over the domain of the animation. We successfully approximate this continuous motion for the positions of our control points using the Runge-Kutta method, but finding an analytical solution using this internal parametric data could lead to interesting possibilities to achieve greater accuracy and eliminate the need for dilation.

We used these interfaces to match the storyboards and add additional detail to the animation. We animated the characters in all 3 Cartesian dimensions, even though the camera positioning gives the viewer no indication of movement in z.

Animation Relative to the Camera

One consideration in the animation of our film in particular was the consistent motion along the positive X axis (left to right in screen space). The film suggests a near constant left to right motion throughout the duration of the short, with only a few rests. This motion is matched perfectly by the camera, which follows along with the characters at the same speed. Thus while the characters are frequently moving large distances each frame, they can be completely static relative to the camera.

While we initially animated our characters with X velocity, we soon removed the overarching motion. Including the motion initially made sense as animating multi-dimensional movements such as jumps felt simpler and more organic. However, as we began exporting the data we realized that unlike velocity fields exported for motion blur, our method requires only velocity relative to the camera. Because the velocity data exported by Mental Ray is based in the world-space movement of the vertices, our bird and fish would report a large amount of X-axis motion even while remaining motionless in the frame.

Solutions to this problem include modifying the Mental Ray source code, using a custom renderer, or simply using the motion of the camera to offset the export velocity data. For the sake of simplicity, we created a workflow in which we kept a static camera and zeroed out all non-individual X movement. Or, in other words, we preserved only those animation curves in the X-axis where the character moved relative to the camera center as shown in Figure 6.7.

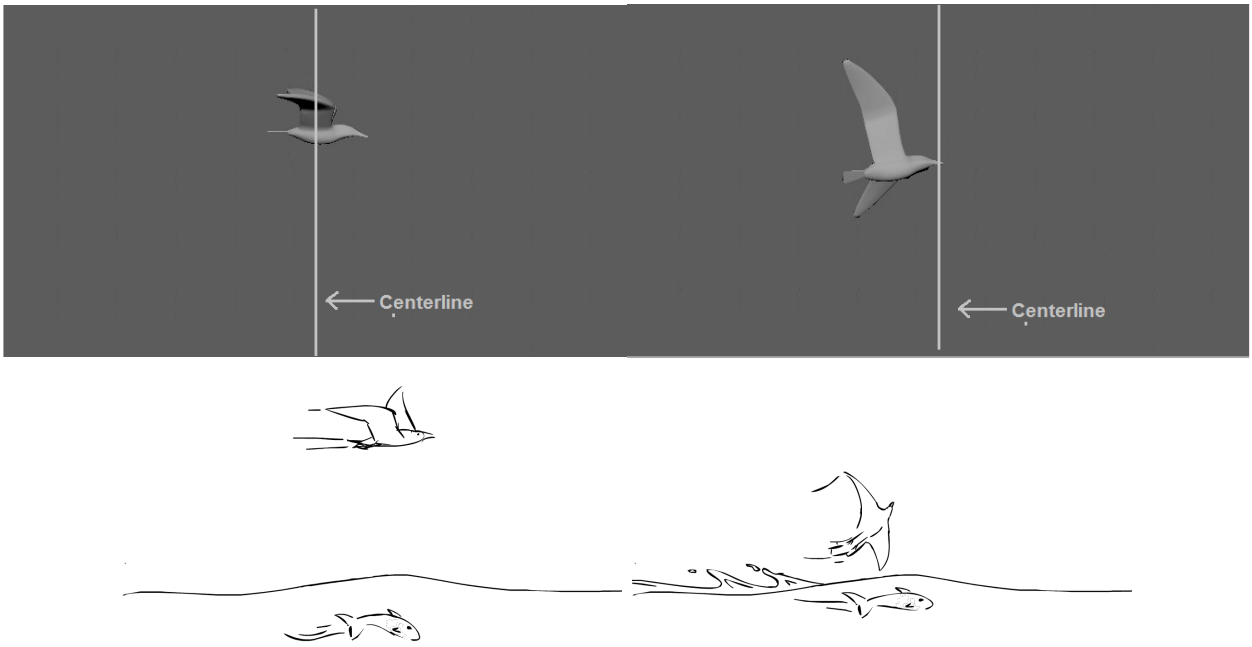

Figure 6.7: An example of X movement relative to the camera. Though both frames suggest X movement and were originally animated with X velocity, only the right image has X movement relative to the camera.

6.3.3 Special Effects

Special effects are broadly defined as anything that moves that is not a character. This can include something attached to a character such as hair and clothing, something natural such as flowing lava, or something imagined like a magic spell. Special effects animators have traditionally been a department of their own in large animation studios, both in 2D and 3D.

While 2D special effects are laboriously created by hand-animating complex artistic shapes after a careful observation of nature, 3D special effects artist today combine artistry with a in-depth knowledge of underlying physically-based simulations to generate computer-assisted representations of the effect. Browning et al.[8] sought in their work to use this physical simulation to ease the burden of the traditional 2D special effects artist, but their work was limited to non-overlapping visual effects and would be unable to properly handle the overlapping and occluded motion data present in our film.

Our method could be used to assist with the creation of any special effect by providing a 3D simulation of the effect or animation of a simplified geometric representation of the effect. It is most useful for a special effect that is tied to the motion of a 3D object. Thus, in our film, we targeted our method to help with the drawing of overlapping contrails for the bird's wings.

In some cases, additional effects could be generated by creating a simple animation (such as animating a motion path with a sphere or cube) and then using our method to generate a more complex shape.

Chapter 7

Results

Our method was able to generate over 100 frames of convincing colored effects animation for 3 separate contrails based on the motion data and an initial key frame and stay on track. In order to test the accuracy of our numerical approximation of the ordinary differential equation we placed a control point of the contrail directly at the position of a vertex along the back edge of the tail. Table 7.1 shows the final positions of the mesh vertex and control point in this test. Over the course of 1250 frames, the most significant deviation of the control point from the vertex was a distance of 1.42 units in a bounding box of motion that is 200 units along the longest axis. While we believe this error could be even further reduced, in this case it was sufficient to reduce the number of necessary drawings for the animation by a factor of over 50, though more frequent key frame inputs may be desirable in places to maintain more stylistic control. Though no official information has been openly published on the capabilities of Meander, released demos only demonstrate generating 50 or fewer frames of animation, half that demonstrated in our demo.

Point Type	X	Y	Z
Original Vertex Position	26.9161	35.2045	131.3750
Control Point Position	27.0459	35.6211	129.9540

Table 7.1: Final positions after 1250 time steps. The numbers on the top row represent the actual X, Y, Z position of a vertex on the mesh at the end of the simulation. The bottom row shows the final position of a control point whose initial position was the same as that of the mesh vertex.

Our method was able to successfully generate in-betweens in such a way that they are able to overlap in 3D space based on their actual depth, an improvement on [8] which

field-2.png field-2.png field2.png field2.png

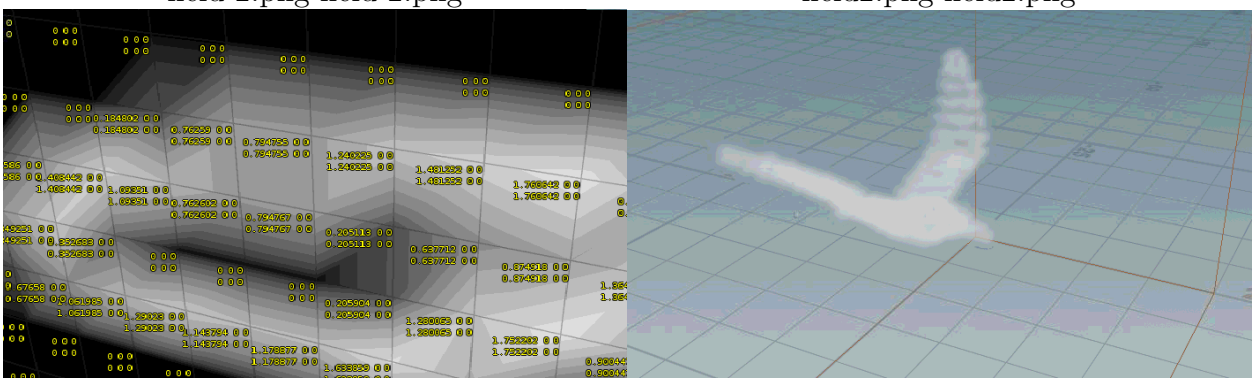

Figure 7.1: The result of our work in Houdini. We generated a 3-dimensional voxel representation of the motion vectors. Houdini visualized non-zero values as a cloud-like volume.

could only handle input animations in a single x and y plane. In many of the frames of our test animation control points are occluded from the perspective of the camera by other parts of the character. Because of our use of a 3-dimensional voxel grid as a data structure to store our data, and due to the stability of our numerical solve in all 3 axes, we are able to maintain accurate parity with the 3D animation even for cases that are normally difficult for 2D interpolation methods. Our method successfully handles overlapping control points and occluded lines. These resultant images are then projected from 3D space into 2D in-betweens at raster time and the draw order can be determined by simply respecting the z-depth ordering during the NURBS curve evaluation.

We were able to apply our technique to animate a drawing that did not have an analogous 3D representation, but rather we were able to extrapolate it's motion from the 3D motion of an animated character. Each contrail was made up of several control points drawn to form a polygonal shape, of which only 1 was initially within the envelope of the character's dilated silhouette. The motion of these contrail effects remained generally consistent and aesthetically pleasing, adhering subjectively to the general optical flow of the motion and maintaining the 2D look that we wanted. After nearly 800 sub-steps our method did begin to demonstrate irregularities in the generated shape. This could be repaired by increasing the number of key frames per frame, for example 1 key frame per 50 frames, or by enforcing a

loose temporal coherence constraint on the shape or by allowing the artist to make corrective adjustments where needed.

In order to test the success of our method we developed a short film with a 2D aesthetic, but that needed the advantage of 3D automated in-betweens to succeed. We created an aesthetic and planned out the motion of the characters. We created 3D models of each character, built rigs and bone structures, and animated their performances. We also created backgrounds, created surface material descriptions, and built a 3D lighting setup to generate the desired 2D effect for each character. While we could have used our method to create fully 2D representations of each character, it was not necessary to the artistic needs of our film.

We tested our method on a 6 second or 144 frame clip of our film. The clip we chose is challenging for any method, with extreme velocities in all 3 axes. In addition, much of the motion of the left wing's contrail is derived from the wing geometry that is occluded by the rest of the bird's body for a majority of the clip. Many of the control points had no associated motion vectors, but were placed freely in space behind the bird. Nonetheless, despite these challenges, these control points followed the character in a convincing and aesthetically pleasing way, as shown in Figure 7.2, being driven by the motion of past time steps as they were aggregated into a motion field. Our test vertex maintained near parity with it's parent vertex, never exceeding an error of 1.42 units during the duration of the clip.

Our system should greatly expand the range of possible 2D animations which can be reasonably created using 3D motion vectors as an underlying structure by enabling the use of overlapping lines and occluded motion data. Furthermore, because the 3D motion data is only created once, we can easily swap out the key frames for any other design and expect similar results. For example, if we were to swap our contrails for a trail of magical particles they would also accurately follow the model, without requiring the user to redo any additional work.

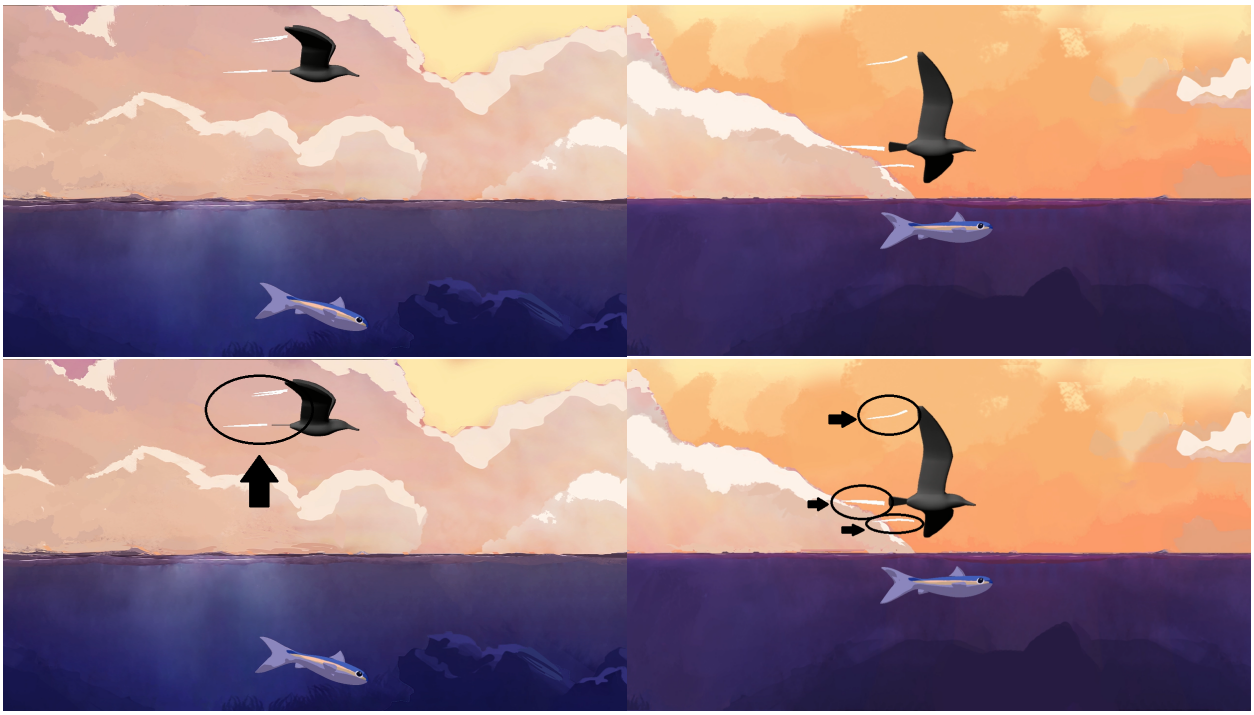

Figure 7.2: Frames showing the final result of our method. Notice that the contrails follow the curvature of the bird's movement, even without a 3D representation of the effect.

In summary, our work improves on the work of Disney[26] and Browning et al. [8] by storing the motion data as a voxel grid rather than an image, allowing support for overlapping and occluded lines without a loss of information or interference from vertices at similar x and y positions but different z depths. Furthermore, our work introduces the concept of persistent motion data over time in order to create motion paths for special effects with a trailing or time component. We also demonstrate the idea of using motion data from animated geometry to drive the in-betweening process for 2D images regardless of whether the animated 3D geometry is a modeled and animated representation of the 2D image.

Chapter 8

Conclusion and Future Work

In this paper we have explored additions to recently established methods of integrating 3D and 2D animation to begin to automate in-betweening, a central problem of 2D animation for many years. Our method builds on the process demonstrated by Disney with their Meander software, which they used on their Academy Award winning short "Paperman". We added to their method an underlying 3D data representation that allows for more complex and accurate interaction between the 3D data and the 2D output. We then applied that method to the creation of special effects animations that had no 3-dimensional representation, which lead to successful results that met the artistic vision, even with overlapping animations.

Our method builds on the current state of the art, and leaves room for significant further research. We have observed that the use of 3D animation as a framework for building 2D animation tools is a field of significant research opportunity.

One area in particular that would further improve our method is the use of dynamic vector line representations to model the line drawings. A dynamic vector art system that can create and destroy control points as needed would allow the system to support more difficult deformations, and allow more flexibility in automatically hiding and recreating occluded lines. Such a system was presented in 2015 at SIGGRAPH by Dalstein et al. [10]. The challenge of such a system is in determining when to create, destroy, or move vertices based on the motion data.

Another interesting new field of research to be applied is machine learning. We recommend 2 particular approaches to applying machine learning.

The first is the use of machine learning to better automatically create a correlation between control points of 2 separate key frames, the same problem Sederberg et al. attempted to solve in 1995[23]. Our method could be used to generate labeled training data for the machine learning algorithm which could provide more accurate correlations for larger point sets and could potentially function in more edge cases.

Another place in which machine learning could be used is in the in-betweening problem in general. Ideally any solution to this problem would include the ability to approximate a representation of the 3-dimensional scene and the forms of moving objects. Nonetheless, animation includes an impressive body of training data, with millions of frames to train on, and even a naive attempt to learn in-betweening could be potentially productive. This would require a way to programmatically label "key frames", though the chosen frames ideally would need to be those with the greatest change in direction, they would not necessarily need to accurately reflect the original key frames as designated by the animators.

An interesting Human Computer Interaction application of the work would be to experiment with new interfaces with the system. Particularly using real-time user generated motion data or webcam data to interactively deform a 2D drawing. Hornung et al.[13] experimented with this kind of interfacing by using a webcam to deform 2D cutouts. This research could be extended to include an underlying 3D character representation that best fits the user.

2D animation provides strengths and tools for some artistic tasks that 3D animation cannot yet match. Short-lived, complex shapes can often be easy to draw for a professional, but require significantly more overhead to create a 3D proxy. Many animation tools attempt to integrate traditional workflows into their user experience because of the speed they provide and many more may continue to integrate 2D and 3D workflows in the future.

We anticipate a bright future for 2D hand-drawn animation, particularly as an integrated part of a hybrid 2D/3D toolset that draws on the strengths of both the artist and the computer to generate animated content. As part of that effort we have created a

computer assisted in-betweening solution that solves the problem of overlapping lines and occluded motion vectors by utilizing 3D voxel-based storage of the animated motion data, and hope that this approach will aid those embarking on future research in this exciting area.

References

[1] Toon boom harmony. https://www.toonboom.com/products/harmony. (Accessed on 03/22/2017).

[2] Most popular animation tv series. URL https://www.imdb.com/search/title?genres=animation&title_type=tv_series.

[3] Meander. URL https://www.disneyanimation.com/technology/innovations/meander.

[4] Wdas technology projects: Computer assisted animation of line and paint in disney's paperman, Mar 2016. URL https://www.youtube.com/watch?v=84rl-T2yIls.

[5] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph.*, 23(3):584–591, August 2004. ISSN 0730-0301. doi: 10.1145/1015706.1015764. URL http://doi.acm.org/10.1145/1015706.1015764.

[6] Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing animation by example. *ACM Trans. Graph.*, 32(4):119:1–119:12, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461929. URL http://doi.acm.org/10.1145/2461912.2461929.

[7] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. Turning to the masters: Motion capturing cartoons. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 399–407, New York, NY, USA, 2002. ACM. ISBN 1-58113-521-1. doi: 10.1145/566570.566595. URL http://doi.acm.org/10.1145/566570.566595.

[8] Mark Browning, Connelly Barnes, Samantha Ritter, and Adam Finkelstein. Stylized keyframe animation of fluid simulations. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, NPAR '14, pages 63–70, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3020-6. doi: 10.1145/2630397.2630406. URL http://doi.acm.org/10.1145/2630397.2630406.

[9] Edwin Catmull. The problems of computer-assisted animation. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 348–353, New York, NY, USA, 1978. ACM. doi: 10.1145/800248.807414. URL http://doi.acm.org/10.1145/800248.807414.

[10] Boris Dalstein, Rémi Ronfard, and Michiel van de Panne. Vector graphics animation with time-varying topology. *ACM Trans. Graph.*, 34(4):145:1–145:12, July 2015. ISSN 0730-0301. doi: 10.1145/2766913. URL http://doi.acm.org/10.1145/2766913.

[11] Daniel Gies, Aug 2015. URL https://www.youtube.com/watch?v=mUmWFgwhUII.

[12] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2001. ISBN 1568811330.

[13] Alexander Hornung, Ellen Dekkers, and Leif Kobbelt. Character animation from 2d pictures and 3d motion data. *ACM Trans. Graph.*, 26(1), January 2007. ISSN 0730-0301. doi: 10.1145/1189762.1189763. URL http://doi.acm.org/10.1145/1189762.1189763.

[14] Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. Three-dimensional proxies for hand-drawn characters. *ACM Trans. Graph.*, 31(1):8:1–8:16, February 2012. ISSN 0730-0301. doi: 10.1145/2077341.2077349. URL http://doi.acm.org/10.1145/2077341.2077349.

[15] Florian Kainz, Wojciech Jarosz, and Rod Bogart. Openexr. URL http://www.openexr.com/.

[16] John Lasseter and Kristina Reed. *Paperman*. Walt Disney Animation Studios, 2012.

[17] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.*, 32(3):27:1–27:22, July 2013. ISSN 0730-0301. doi: 10.1145/2487228.2487235. URL http://doi.acm.org/10.1145/2487228.2487235.

[18] Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Trans. Graph.*, 32(1):4:1–4:11, February 2013. ISSN 0730-0301. doi: 10.1145/2421636.2421640. URL http://doi.acm.org/10.1145/2421636.2421640.

[19] Lena Petrović, Brian Fujito, Lance Williams, and Adam Finkelstein. Shadows for cel animation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 511–516, New York, NY, USA, 2000. ACM

Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.345073. URL http://dx.doi.org/10.1145/344779.345073.

[20] Thomas W Sederberg and Eugene Greenwood. A physically based approach to 2–d shape blending. In *ACM SIGGRAPH computer graphics*, volume 26, pages 25–34. ACM, 1992.

[21] Thomas W Sederberg, Peisheng Gao, Guojin Wang, and Hong Mu. 2-d shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 15–18. ACM, 1993.

[22] M.D. Smith. *TweenMaker: Adding Life to Computer Supported Cartoon Inbetweening*. Brigham Young University. Department of Computer Science, 2003. URL https://books.google.com/books?id=WSkuOAAACAAJ.

[23] F. Thomas and O. Johnston. *The illusion of life: Disney animation*. Hyperion, 1995. ISBN 9780786860708. URL https://books.google.com/books?id=2x0RAQAAMAAJ.

[24] B. Whited. Systems and methods for interactive vectorization, September 6 2012. URL https://www.google.com/patents/US20120223949. US Patent App. 13/040,837.

[25] Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W Sumner, Markus Gross, and Jarek Rossignac. Betweenit: An interactive tool for tight inbetweening. In *Computer Graphics Forum*, volume 29, pages 605–614. Wiley Online Library, 2010.

[26] Brian Whited, Eric Daniels, Michael Kaschalk, Patrick Osborne, and Kyle Odermatt. Computer-assisted animation of line and paint in disney's paperman. In *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, pages 19:1–19:1, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1683-5. doi: 10.1145/2343045.2343071. URL http://doi.acm.org/10.1145/2343045.2343071.