SPEED-LINE FOR 3D ANIMATION

A Thesis

by

WON CHAN SONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2005

Major Subject: Visualization Sciences

SPEED-LINE FOR 3D ANIMATION

A Thesis

by

WON CHAN SONG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Ergun Akleman
Committee Members,	Carol LaFayette
	John Keyser
Head of Department,	Mardelle Shepley

December 2005

Major Subject: Visualization Sciences

ABSTRACT

Speed-Line for 3D Animation. (December 2005)
Won Chan Song, B.F.A., Ringling School of Art and Design
Chair of Advisory Committee: Dr. Ergun Akleman

My thesis describes a tool which creates speed-lines automatically in 3D computer animations. Speed-lines are usually used in comic books to express fast motions in a still image. They are also used in 2D animations. Although animations don't need speed-lines for motions, they are interesting graphic elements and give more sense of speed to the audience. However, speed-lines are not used in the 3D computer animation, so I have implemented the graphic element into the 3D computer animation. The combination of the 2D-looking speed-lines and the 3D computer animations makes a unique look that some animators might want. In addition, making speed-lines based on motions of moving objects or cameras, this tool can make 3D speed-lines which are very difficult to draw by hand.

ACKNOWLEDGMENTS

I am deeply grateful to my thesis committee chair, Dr. Ergun Akleman, for his teaching and caring help throughout the entire process of my thesis. I would also like to thank my committee members, Prof. Carol LaFayette and Dr. John Keyser for their advice and feedback. My gratitude also goes to Hobart Chan for letting me borrow his car model, which saved me a huge amount of time when I was making an animation as a demonstration of my thesis. I would also like to thank Piotr Krawczyk and Karthik Swaminathan for their helpful ideas to solve some scripting problems. I would also like to thank Lei Han and Zhang Xiao for giving me information and helping to start my research. I would also like to thank Julie Garcia for her encouragement and good feedback. Thanks especially to Jaemin Lee, Hyemee Choi, Jinnah Yu, Jaewook Lee, Sungwha Lee, and Heeyeon Jo for helping me to adapt myself to the Viz Lab and showing me gracious friendship. Thanks also to all of the faculty, staff, and students of the Viz Lab for their kind help in every way. I am thankful to Seungwhan Lee and Jungwook Son for helping with my research on speed-lines in comic books. I am also thankful to all the authors in my references for their ideas and methods that have affected my thesis.

Especially, I would like to express my gratitude to my family from the bottom of my heart for their love and support throughout the years. I would also like to thank Youngjin Kim for lightening my heart with her cheerfulness whenever I was frustrated. My gratitude also goes to Prof. Seongwoo Nam's family for showing me unconditional love when I was in the hardest time in my life. I would also like to thank Taesik Yang and Woohyun Nam for their continuous friendship and refreshing my mind. Thanks also to Jaehong Park for pushing me to finish my thesis and for his encouragement. I would also like to thank Sungwon Ryu and Jaehak Jung for

comforting and pleasing me all the time. I would also like to thank Sunjune Yoon for motivating me to get interested in the special effects of movies and to decide to study in the U.S. I would also like to thank Chanyeon Cho and Inkyu Park for being my mentors and providing their guidance in the most important time in my life.

I would also like to thank Pastor Chong Kim, Youngsik An and their families for guiding me to God and giving me a new life. Thanks also to Deacon Gyusung Lee's family for their love and prayer. I would also like to thank Deacon Sangyun Lim's family for leading my bible study group, showing a great trust on me, and caring everything in my life here in College Station. My thanks also goes to Deacon Siyoung Jang's family for sharing the life in God with his words. Thank you to Deacon Kyungho Lee's family for being wonderful fellow workers and filling my life with pleasure. I would also like to thank Jangwook Lee's family for praying together and helping me to learn programming. Thanks also to Chansung Park, Sungmin Cho, and Hojin Kang for their strong and precious fellowship. Thanks to everyone in Vision Mission Church for helping me to keep my faith and praying for me continuously. Finally, I would like to express my deepest thanks to God for loving, meeting, and saving me. He has given me the power to get through all the troubles in my life and led me up to this time. I can never thank him enough.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

One of the earliest artists who tried to convey dynamic movement in a static image is Hokusai Katsushika, who is one of the best known Ukiyo-e[1] artists. He is called "Father of Manga[2]" because of "Hokusai Manga" which was published in 1814. The most famous piece from the book is "The Great Wave" where his observation is as accurate as a camera with shutter speed of 0.0001 second (see Figure 1.a) [1]. He was very interested in movements expressed in a picture and he tried several techniques for it. Koukasen (effects-line) was one of them. In Figure 1, he added effects-lines to describe invisible motions in his pictures and that makes them look more realistic [2].

(a) The Great Wave (b) Mt. Fuji from Kajikazawa (c) Fish Basket Hokusai Manga vol.13

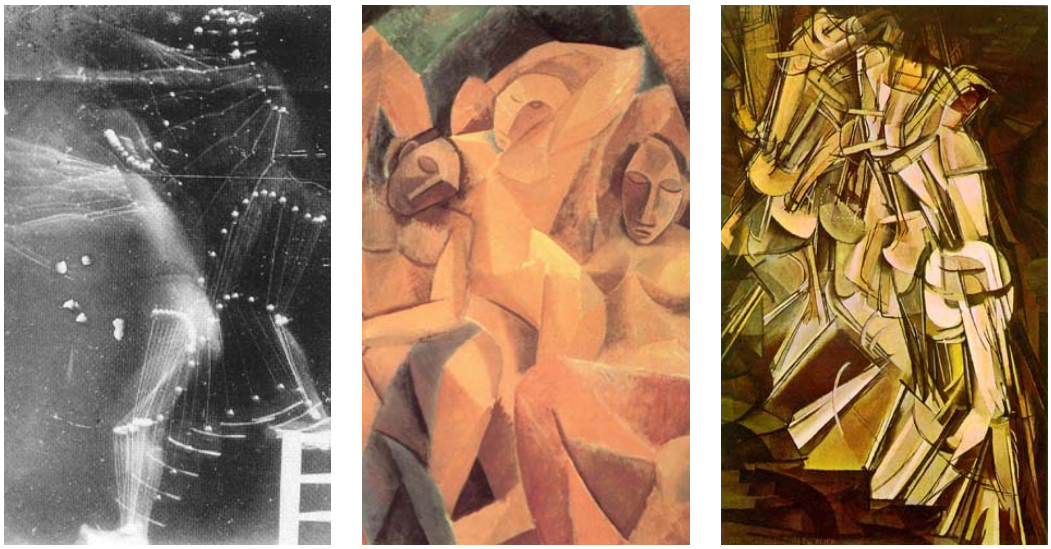
Fig. 1. Effects-lines invented by Hokusai

The journal model is *IEEE Transactions on Visualization and Computer Graphics*.

[1]A Japanese term for a type of popular art that was favored from the sixteenth century, particularly in the form of color woodblock prints. Ukiyo-e prints often depicted the world of the common people in Japan, such as courtesans and actors, as well as landscapes and myths [3].

[2]A Japanese word for comics or cartoons. Manga is literally translated "random pictures [4]."

In the late nineteenth century, the development of photography technology enabled photographers such as Edward Muybridge and Etienne-Jules Marey to capture moments of moving animals and people [5]. Marey created his photographs by showing successive phases of the movement of humans in the 1880s. Marey's photographs influenced Nude Descending a Staircase of Marcel Duchamp [6]. Duchamp took the principle of decomposing movement into a sequence of figures from Marey's photographs. He wanted to paint a nude different from classical paintings, so he decided to add movement in his painting [6]. He was also influenced by Cubism and tried to develop his visual vocabulary of Cubism in the period [7]. He simplified a form of a human figure with just lines. The repetition of linear elements was called 'elementary parallelism' and it served to show the movement on a static image. The lines were used to represent not only the figure but also the motion lines to improve the mobile quality of the painting [7] (see Figure 2).



Etienee-Jules Marey Pablo Picasso Marcel Duchamp

Fig. 2. Influences on the motion line in the western art

Comic book artists also took the idea of the motion line and developed it over a long period of time [8]. In the early stage of the comics, the motion line depicted movement but it seems not to have been done deliberately for the composition of the image (see Figure 3.a). Over the years, the motion line became more refined and stylized and it became the essential element of the comics, especially the action ones like Figure 3.b and 3.c. However, most uses of the motion lines were limited to just following a moving object [8]. In the 1960's, Japanese artists applied an interesting photographic trickery, which is that the background is streaked, instead of a moving object, when the camera is moving with the object (see Figure 4.a). This technique became fairly common when making a dynamic background for a speedy shot, and new styles of the motion line continue to be invented [8].

(a) The Yellow Kid 1896

(b) Superman 1941

(c) Elektra: The Hand 2005

Fig. 3. Improvement of the motion line [9] [10] [11]

The motion line is not only for representing motion but also for giving greater sense of the speed to the viewer [8]. So, the animation has also been adopted in the technique for the speedy shots (see Figure 5). Especially, the Japanese animations use

this technique actively because it is very effective in the limited animations prevalent in Japan. Since limited animation has fewer frames and speedy shots, Japanese animation uses the effects very frequently. The Disney Studios, which make full animations, also use motion lines, but they usually rely on motion itself of moving objects from frame to frame.

(a) Golgo 13 (b) Yul-Hyul-Gang-Ho (c) Flash

Fig. 4. Various styles of the motion line [12] [13] [14]

Naruto

Fig. 5. An example set of animation frames using motion lines [15]

The computer graphics technologies started in the 1960's resulted in the emergence of 3D computer animation in the 1980's [16]. In 3D animation, animators set key frames and in-between frames are created automatically, and each frame shows accurate motion. So, speed-lines compensating for in-between frames might not be

necessary any more. However in-between images generated by computer are too sharp, even though they are in the middle of a fast motion, which makes the animation look unrealistic. That is why motion blur has been developed [17].

I think that motion lines are not only an extension of the motion blur technique, but they are also great graphic components in a shot (see Figure 3 and Figure 4). My goal in this thesis is to develop a system that makes motion lines automatically for 3D computer animations. Like motion blur, the lines will be created based on the movement of objects in a scene.

CHAPTER II

PREVIOUS WORK

In computer graphics, there has already been an interest in speed lines. For instance, a system to create speed-lines has already been developed [18]. Using the speed-lines, Masuch et al. were able to convey information about movement without smearing the shapes of moving objects. Their solutions were to present motion in a static image by drawing speed-lines, to indicate contour lines of former positions of moving objects, and to illustrate arrows of moving direction. Figure 6 shows the different categories and their combination. According to their heuristics about where and how to draw them, they found that speed-lines and contour lines:

- are drawn in the opposite direction of the movement (thus reaching into the past),
- start at "characteristic" points of the moving object,
- embrace the minimum and maximum extent of the object,
- are more or less equally sized, shaped and directed, without intersecting each other,
- are uniformly, but not too regularly distributed.

Their approach to generate motion lines, arrows, and repeated contours uses a polygonal 3D model along with key frame data of an animation as input. They use non-photo-realistic rendering techniques to make the 3D objects go along with motion lines. Since their motion lines follow moving objects, they are not appropriate for movements directed towards the viewer.

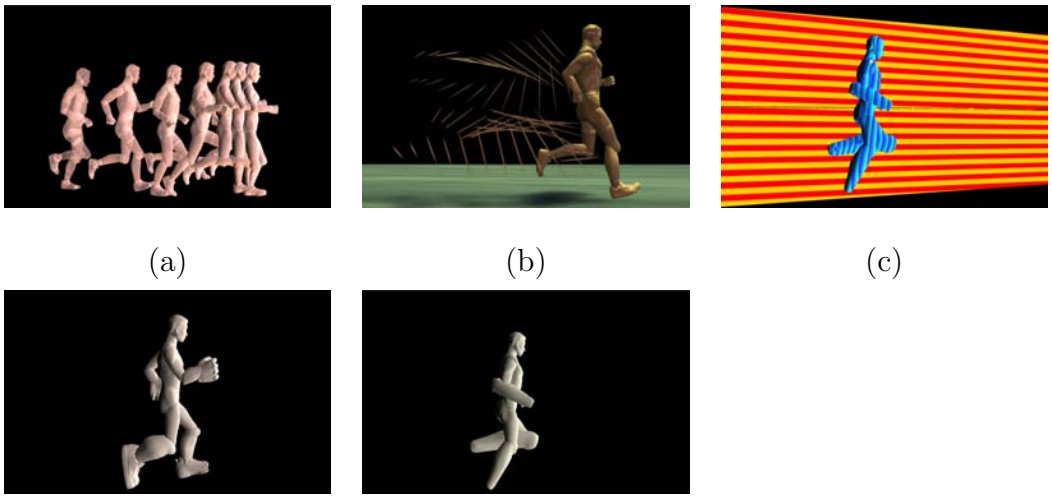
Different Styles of Lines Depicting Motion

Combinations of Different Styles

Fig. 6. Speedlines, contours and arrows

Chen et al. developed an abstract rendering system for the motions of animated objects in 3D computer animation by using techniques from traditional arts [19]. He tested five traditional techniques: image overlapping, shadowing line patterns, creating a central line of motion, distorting the object, and exaggerating the pose. Figure 7 shows the results of each test. To determine the best style to emphasize the dynamics of the motion, he tested the techniques individually and combinations of them. This project was limited to only the motion of a human running whose data were generated by the Runner Simulation developed by Jessica Hodgins [20].

McNamara et al. developed a system for drawing smooth motion lines in a video footage in a class project [21]. His system is based on motion blur, which is the most common and intuitive way to convey the feeling of motion. However, using his system, the details of images are lost. The system loads two frames of footage and uses them to draw lines with width, length, opacity, and color that users want. The efficacy of this system depends on the simplicity of the action and the mood of the image (see Figure 8).



(a) (b) (c)

(d) (e)

Fig. 7. Artistic renderings of dynamics actions

(a) (b)

Fig. 8. Motion lines for video

Kim et al. developed a semi-automatic system adding expressive illustration of motion on video [22]. Their system recognizes moving objects on video footage and adds visual effects using the masks and the edges of the moving objects to accentuate their movement. Their results show four styles of nonphotorealistic and expressive illustrations rendered with video footage inputs. The four styles of methods depicting motion are particle-effects, time-lapse, temporal-flare, and speed-lines (see Figure 9).

Particle-Effect Time-Lapse

Temporal-Flare Speed-Line

Fig. 9. Video-based nonphotorealistic and expressive illustration of motion

CHAPTER III

METHODOLOGY

To create speed-lines for 3D animation, we have to first identify the types of speed-lines. Animators have their own style and there is a huge variety of speed-lines. After watching many animations and looking for speed-lines, I found that the most commonly used speed line types are (1) lines following a moving object, (2) lines across the screen in a side view, and (3) lines covering the area around the camera in a perspective view. In my thesis, I have focused only on these three speed-line types.

(a) (b) (c)

Fig. 10. Three types of speed-lines

A. Speed-lines following objects

Figure 10.a shows the most common speed-line. The speed-lines trace a moving object and each line lasts for a few frames because the lines would look like just tails if all the same lines stay on the object too long during the shot. Almost every frame shows different lines, so the flickering effect gives the viewer a sense of a fast motion. For this effect, each line is assigned a series of timings when to start and a duration to be shown. The length of each line should be different, too. A stroke of each line could be a single line or a group of thin lines depending on the style of an animator. To

make this type of speed-line in my method, you select vertices on a moving object which will draw speed-lines on their trails. You also input the ranges of length, life, speed, number of lines in a stroke, and distance between lines that the speed-lines will be based on.

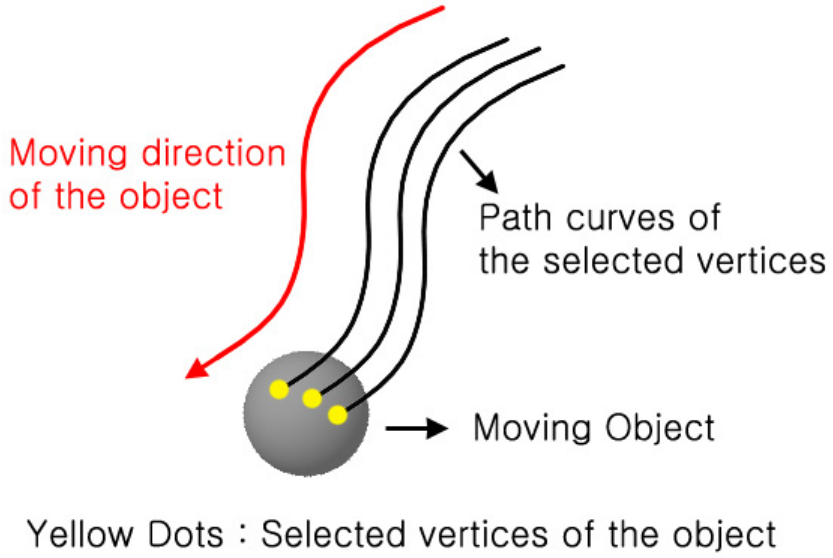


Fig. 11. Selected vertices and their path curves

To make speed-lines follow a moving object, you need path curves that trace selected vertices during animation (see Figure 11). Figure 12 shows that each point of the path curve is where the selected vertex passed at a past frame. When you have a path curve and move a speed-line along with the path curve, the movement of the speed-line will match the movement of its object. There are some important elements to animate this type of speed-line.

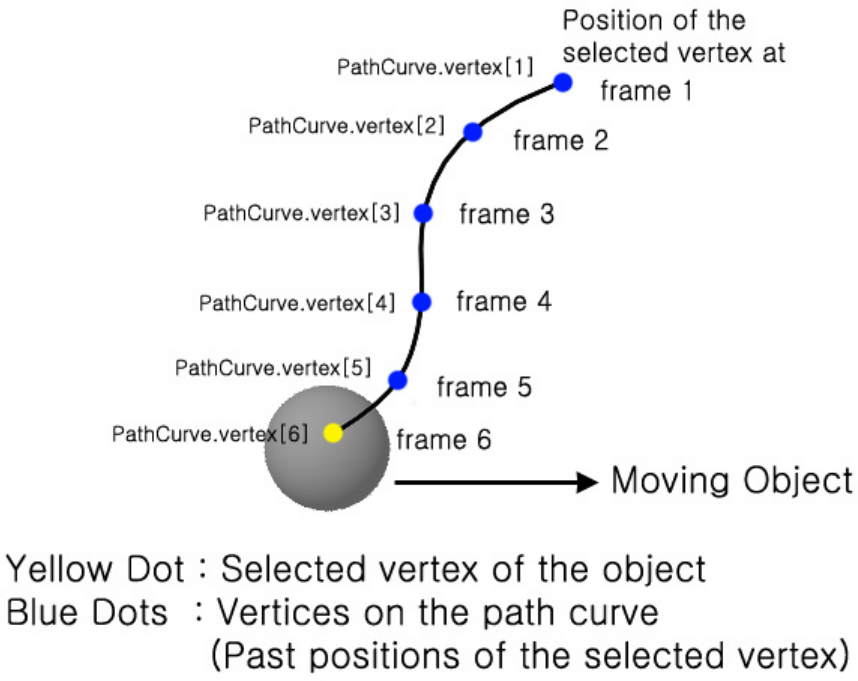


Fig. 12. Relationship between a selected vertex on a moving object and the vertices on its path curve

1. Length Unit

   The unit of the length is a frame instead of a unit of a coordinate system. As shown in Figure 12, the indices of points on a path curve matches the past frame numbers of its moving object. So, using a frame as the length unit makes it easy to position a speed-line on a path curve. For example, a speed-line that consists of three vertices will be drawn as in Figure 13.a when it starts from PathCurve.vertex[5] and the length is 1. The second and the third vertices of the speed-line are overlapped. But the same speed-line will be like the one in Figure 13.b when the length is 2. Another reason for using a frame as the length unit is that it changes the actual length of speed-lines automatically according to the speed of the object leading them. When a length is 1, that means the

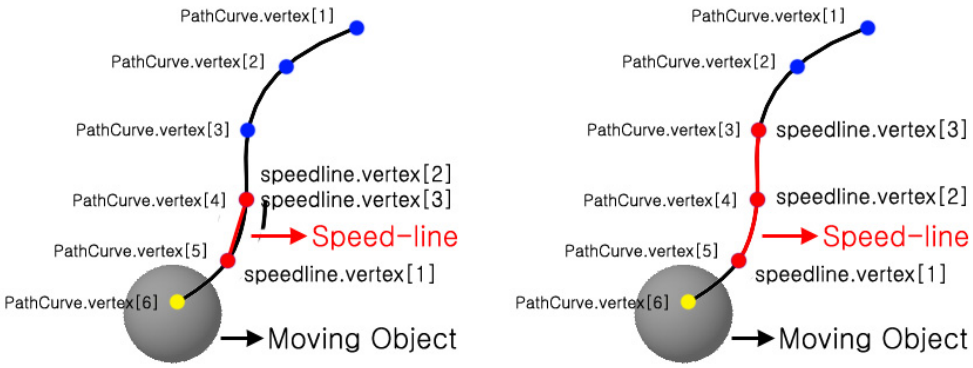


Fig. 13. Using a frame as the length unit of a speed-line

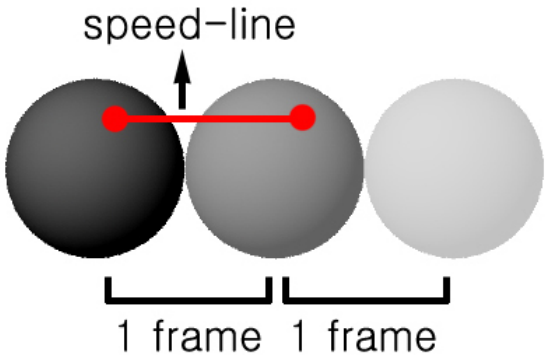
distance an object moves for one frame. So, a speed-line will be short when its object moves slow and a long speed-line will be drawn when its object moves fast (see Figure 14).

2. Visibility Timing

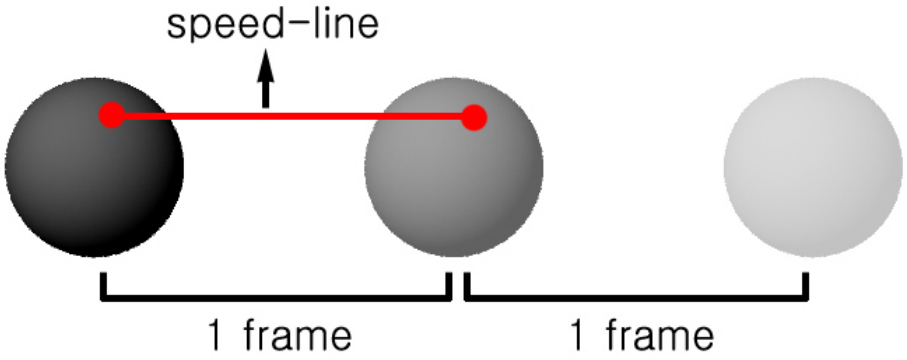
   Speed-lines appear and disappear repeatedly during the animation. For this flickering effect, each speed-line needs visibility timings. Figure 15 shows how visibility timings are assigned by using the life attribute and probability.

3. Life

   Speed-lines also have their life attribute. They show up at the assigned frames and disappear after their life frames. The life attribute is also a random number, so speed-lines appear and disappear at different times.



(a) Speed-line Length : 1, Sphere Speed : 1



(b) Speed-line Length : 1, Sphere Speed : 2

Fig. 14. Speed-line length depending on the speed

Frame	Random number	Process	Visibility
1	1.3	Higher than the probability (1). Skip the frame	Off
2	0.7	Lower than 1 Assign this frame as a visibility timing	On
3		Skip frame 3 and 4	On
4		because the life of the speed-line is 3	On
5	1.6	Higher than 1 Speed-line disappears	Off
6	1.9	Higher than 1 Skip the frame	Off
7	0.1	Lower than 1 Another visibility timing	On
8		Skip during the life	On
9			On
		⋮ continue until the end of the animation	

Probability : 1.0, Life : 3.0

Fig. 15. Example of the process assigning the visibility timings

4. Probability

You need a probability to assign visibility timings to speed-lines. Let's say you input a number between 0 and 2 for a probability. In a frame when a speed-line is not on the screen, a random number is generated in the range of 0 and 2. If the random number is lower than the probability you input, the frame is assigned to the speed-line as a visibility time. If it is not, another random number is generated and compared with the probability in the next frame (see Figure 15). So, a higher probability enables speed-lines to show up more frequently (see Figure 16). As a result, you can see more speed-lines at a frame (see Figure 17).

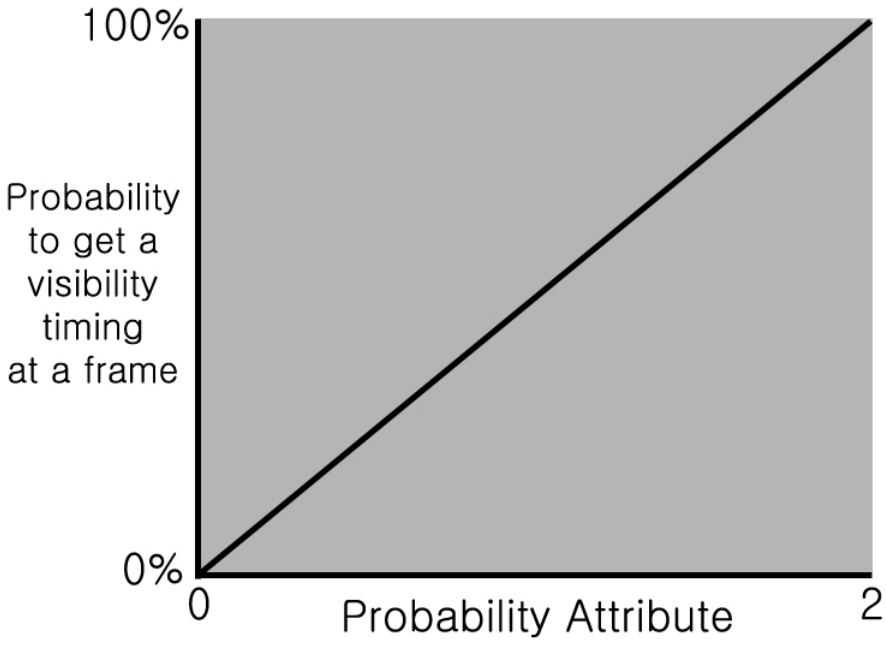
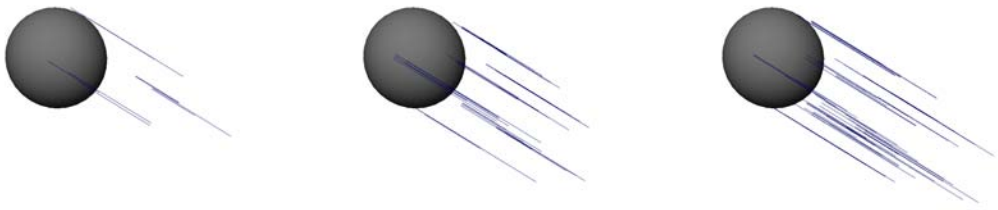


Fig. 16. Probability attribute and the visibility timing assignment

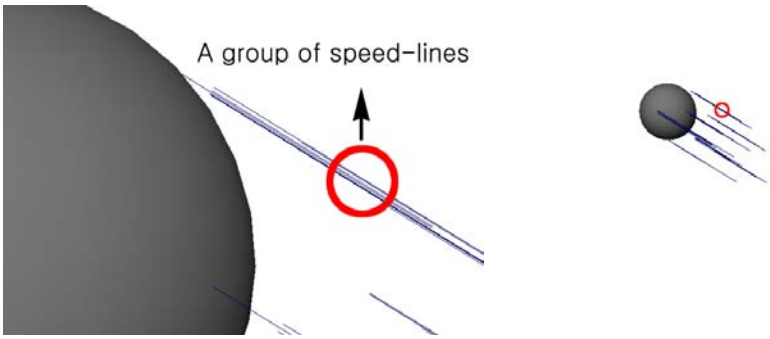


(a) Probability 0.2 (b) Probability 1.0 (c) Probability 1.8

Fig. 17. Probability attribute and the number of speed-lines at a frame

5. Group Life

Basically, selected vertices on a moving object draw speed-lines in this type. However, a single speed-line per selected vertex is too weak to give enough feeling of speed. So, a selected vertex on a moving object draw a group of speed-lines and all of them appear and disappear frequently with the flickering effect. The speed-lines in the same group are very close to one another and they look like a single line from a distance (see Figure 18). Since they have different visibility timings and life, it is hard to see all of the lines in the same group disappearing at the same time. Some of them disappear, but others still remain on the screen because of different timings. That makes it hard to get the flickering effect explained above. Since the speed-lines in a group are so close to one another, you won't notice disappearing of a speed-line when other ones around it still remain on the screen. A group life attribute is needed to solve this problem. Every group has its own life and a group life gets higher priority than an individual life of a speed-line. Even though a speed-line is not older than its life, it disappears when its group is older than the group life. So, a speed-line can show up only when it is not older than its individual life and its group is not older than its group life either. This attribute enables a group of speed-lines to be flickering together.



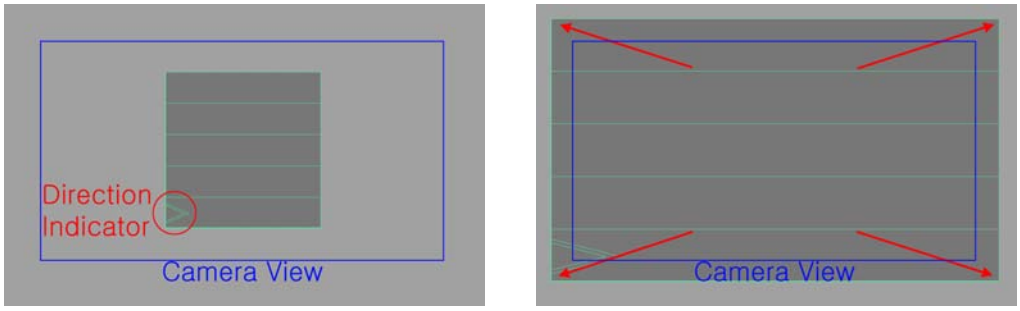
(a) Close View (b) Distant View

Fig. 18. Speed-line group

B. Speed-lines across the screen

Figure 10.b illustrates speed-lines across a screen. This kind of shot is usually close up to a moving object in a side view. The camera moves along with the object and the background behind it gets blurred. The speed-lines keep moving straight across the screen in the opposite direction of the object, and that makes the object look even faster. In this type of the shot, speed-lines are the closest element to the camera and they are very dominant. My method has two parts for this type of speed-lines. The first part makes a plane to be used for animating speed-lines. The plane include an arrow that indicates which way speed-lines will move. The second part actually position and move speed-lines by using the plane created in the first part.

Part 1 A plane is created in this part and the number of its horizontal lines are decided by your input. The arrow on a corner indicates the direction of speed-lines which will be generated later (see Figure 19.a). Speed-lines will be placed and animated on the plane. So, you have to adjust the orientation, the size of the plane, and the distance between the camera and the plane in order to get speed-lines you want (see Figure 19.b).



(a) Basic plane (b) Plane adjusted to cover the screen

Fig. 19. Plane setup for speed-lines across the screen

Part 2 Once the plane is set, this part calculates the attributes of speed-lines and animates them. Visibility timings are also needed for this type but life attributes are not. Speed-lines start from a point on an edge of the plane at their visibility times and they disappear when they go beyond the other side (see Figure 20).

Once you have a starting point and an ending point on the plane, you can find the direction V_D (see Figure 21).

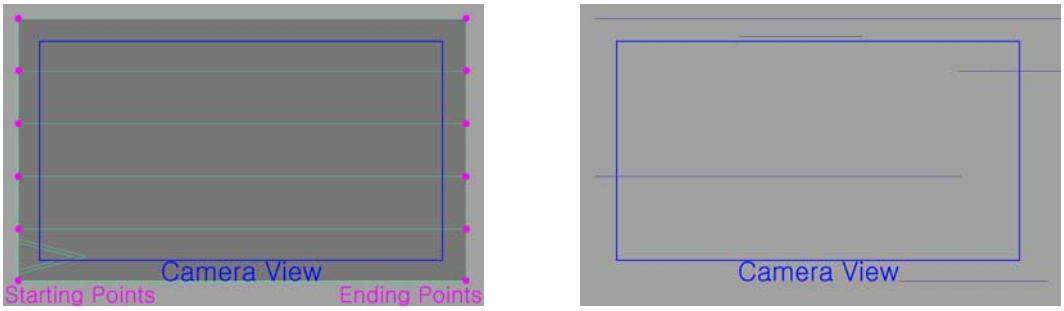
$$V_D = unit(Pt2 - Pt1)$$

When you know how many frames the speed-line has moved, you can position it by moving its first and last vertices.

$$P_S = Pt1 + (V_D \times speed \times frames)$$

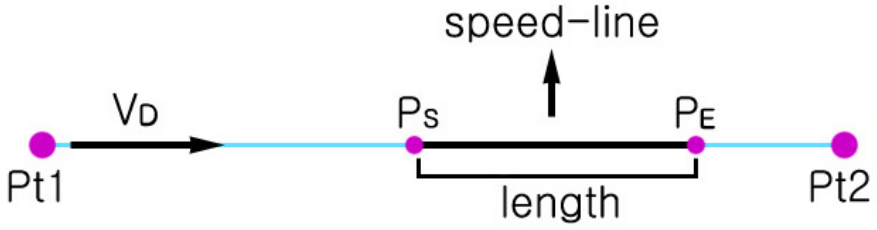
$$P_E = P_S + (V_D \times length)$$

Unlike the previous type of speed-lines, this type of speed-lines use the width of the plane as the length unit. In this way, you can easily set up the range of the length



(a) Starting and ending points of speed-lines (b) Speed-lines

Fig. 20. Drawing speed-lines by using a plane



Pt1 : Starting point on the plane

Pt2 : Ending point on the plane

V_D : Direction of the speed-line

P_S : Starting position of the speed-line

P_E : Ending position of the speed-line

Fig. 21. Drawing a speed-line between starting and ending points

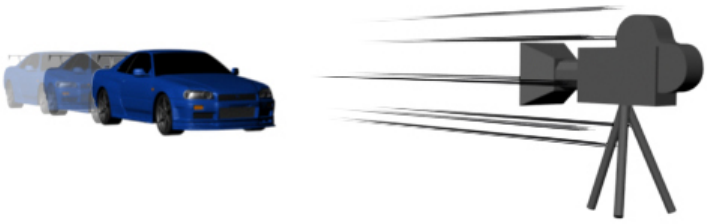
attribute no matter how far the plane is located from the camera.

C. Perspective speed-lines

Figure 10.c depicts an example of the speed-line where a moving object is getting closer to the audience. When you see a motion from the side, you can see the distance an object moves. But it is hard to see the distance and the speed of a moving object in a perspective view. So this type of speed-line is very useful to express the sense of speed in a perspective view. Unlike the types of speed-line above, the perspective speed-lines are drawn in great depth, which makes the composition more dramatic. Another benefit of using these lines is that you can draw the viewer's attention to wherever you want to by converging the lines into a specific area in the screen. I have developed two approaches to create the perspective speed-lines. One of them is for the case where the camera is fixed on a position and only a focused object is moving toward the camera. The other case is that both of the camera and a focused object move together in the same direction.

1. Perspective Speed-lines of a Fixed Camera

   You select a moving object as the goal of the speed-lines in your scene. You also input the number of speed-line strokes and the number of lines per stroke. You define the ranges of the length and the speed of each line. Based on the positions of the camera and the goal, straight lines are created from the camera to the goal every frame changing the length of the lines randomly (see Figure 22). The fast change of the lines gives you the sense of speed.

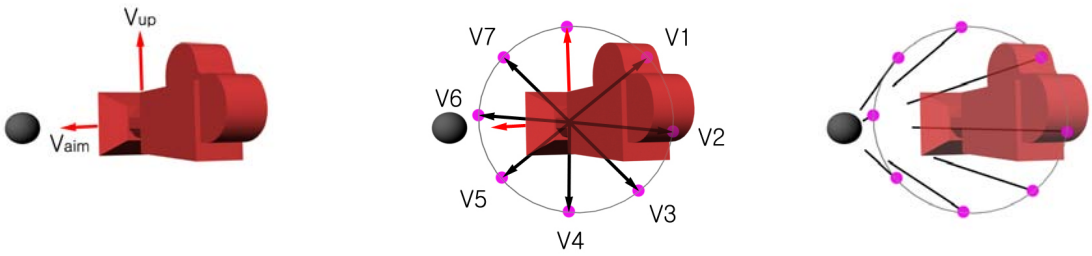


(a) Lines from the camera to the goal

(b) Camera view

Fig. 22. Perspective speed-lines of a fixed camera

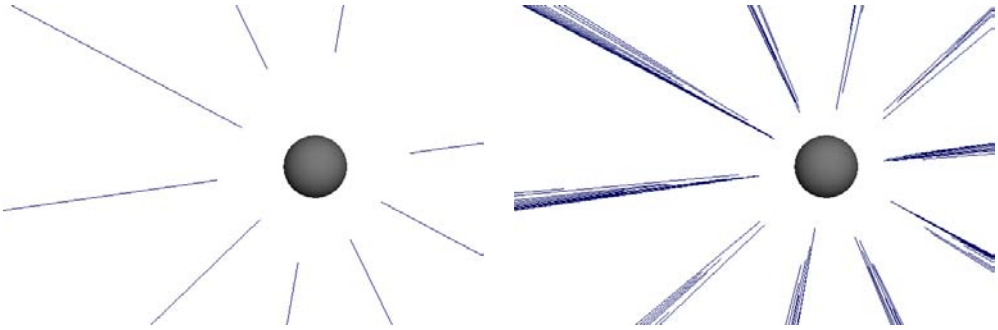
Since the lines are straight, you can draw them by using starting and ending points. You can get starting points by using the up vector and the aim vector of your camera (see Figure 23.a). When you rotate the up vector around the aim vector, you can get the positions of some starting points around the camera (see Figure 23.b). Once you have starting points, ending points are placed somewhere between the starting points and the goal object according to the length of each speed-line (see Figure 23.c). The length of the speed-lines changes every frame with a random number, which makes the flickering effect.



(a) Up Vector & Aim Vector (b) Circle around the camera (c) Speed-lines

Fig. 23. Starting points of the speed-lines

A single speed-line per starting point is weak to give enough sense of the speed (see Figure 24). So, the starting points are copied nearby to make more speed-lines (see Figure 25).



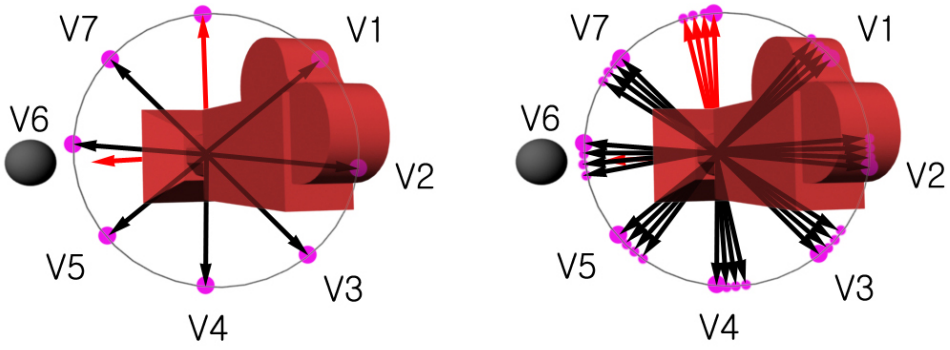
(a) Single Speed-line (b) Speed-line Groups

Fig. 24. Groups of speed-lines

When you have groups of speed-lines, there is a problem with the length of speed-lines. Every speed-line changes its length every frame. Since there are many speed-lines in a group, the overall lengths of the groups are similar to one another even though the speed-lines in a group have different lengths (see Figure 26.a). To differentiate the lengths, the group length attribute is needed. A group is assigned a random length and it makes the maximum length in the

group. The individual speed-lines in the group change within the limit.

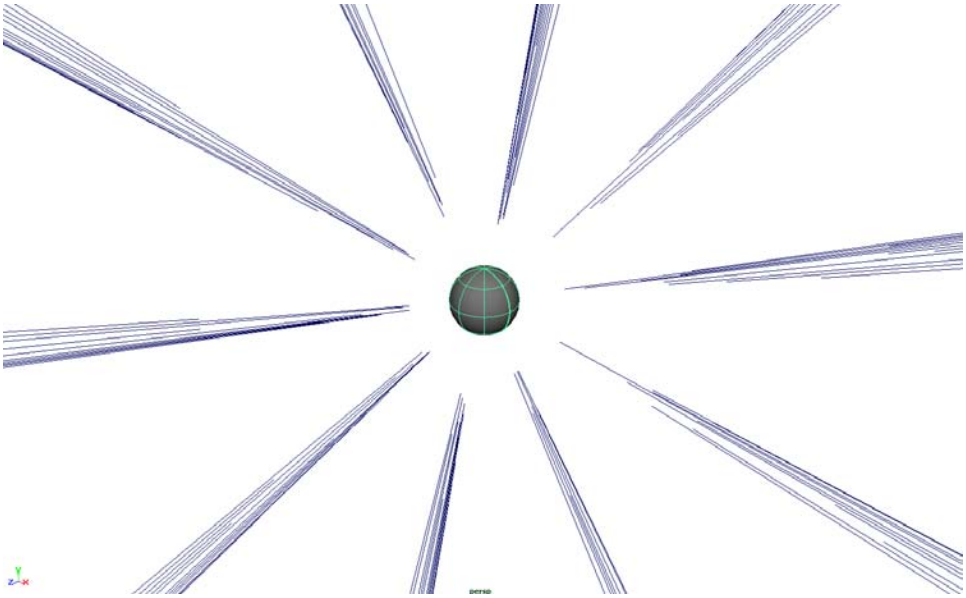
$$L_{individual} = ratio \times L_{group}$$



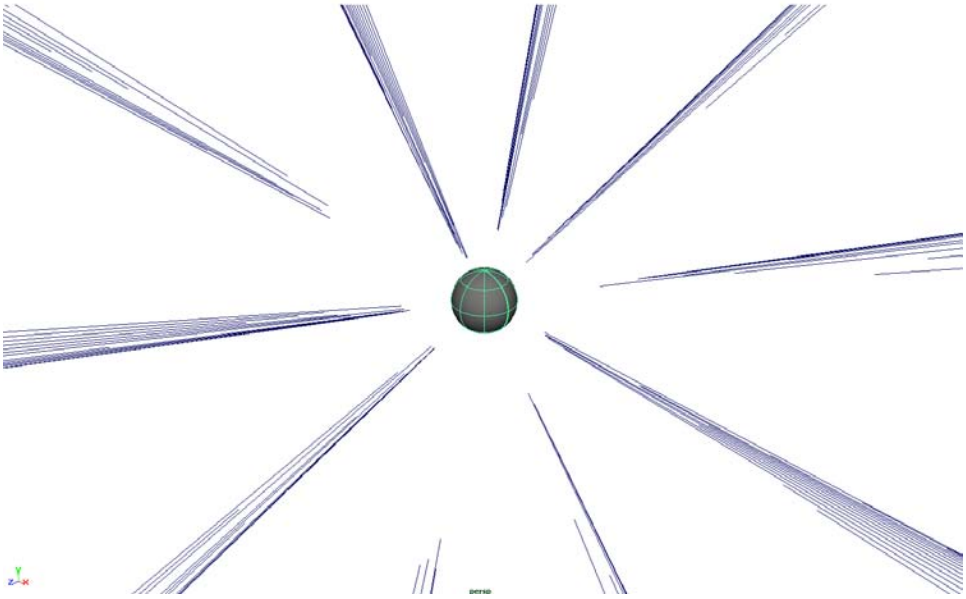
(a) Single starting points (b) Starting point groups

Fig. 25. Starting points copied for more speed-lines

$L_{individual}$ is the length of a speed-line, and the ratio is a random number that is assigned to every speed-line every frame. The ratio should be ranged from 0 to 1. L_{group} is the group length attribute and it is the maximum length in the group because the limit of the ratio is 1. The group length attribute enables us to avoid the uniform length of the speed-line groups (see Figure 26.b).



(a) without group length

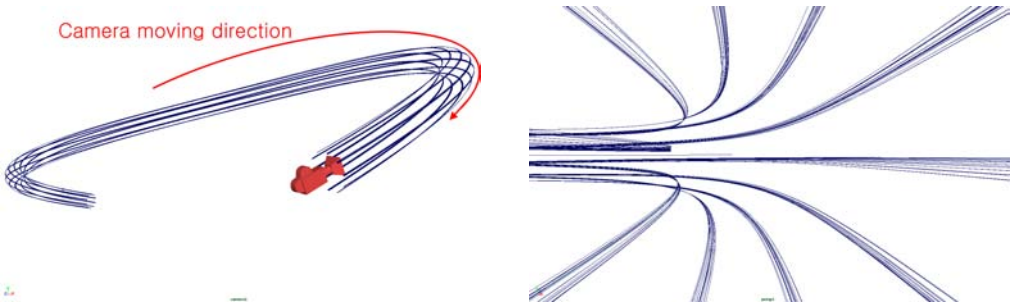


(b) with group length

Fig. 26. Group length attribute

2. Perspective Speed-lines of an Animated Camera

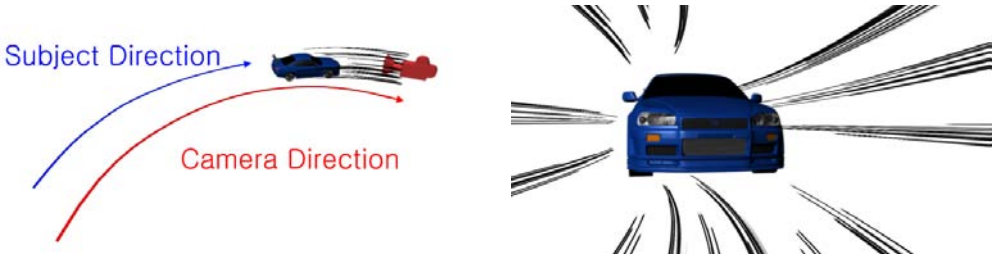
In this approach, there is no need to set up the goal object because this approach uses the position of the camera in each frame during the time you define, and the speed-lines will be created according to the trail of the camera. You just input the same information as you do for the speed-lines of the fixed camera. The information you have to add here is the starting and the ending frame to get the trail of the camera. When the camera is fixed, you can get the straight lines only. When the camera is animated, however, you can also get the curves according to the motion of the camera.



(a) Animated Camera & Path Curves (b) Camera View

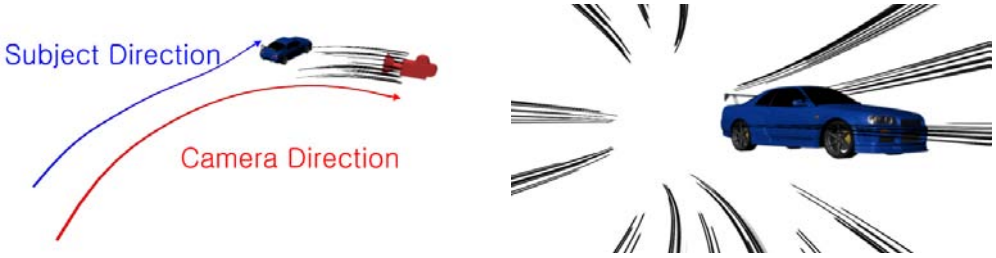
Fig. 27. Path curves of the CVs around an animated camera

Some points are generated around the camera as shown in Figure 23. They draw their path curves according to the movement of the animated camera and the path curves consist of the points on the past positions of the starting points (see Figure 27). The path curves make a tunnel with the track of the camera on the center of it. To use this type of speed-lines, you have to animate the subject focused by speed-lines to follow the camera inside the tunnel and keep a certain distance between the camera and the subject (see Figure 28). Otherwise the subject will be off the focus area where the speed-lines converge, which makes the speed-lines useless (see Figure 29).



(a) Same directions (b) Camera view

Fig. 28. Right relationship between the camera and the subject



(a) Different directions (b) Camera view

Fig. 29. Wrong relationship between the camera and the subject

This kind of shot is usually used for a chasing scene. The camera moves along with a moving subject, so the subject is always focused and the background gets blurred (see Figure 30.a). As a result, the moving subject look almost stationary in front of the camera and the background moving away from the camera. So, passing objects in the background are the only elements to express speed in a regular chasing shot. Using perspective speed-lines helps to show greater sense of speed (see Figure 30.b).

(a) Blurred background (b) Perspective speed-lines

Fig. 30. Example of perspective speed-lines

One thing you need to note is that the camera should move backward to see the speed-lines. When the camera moves forward, the path is behind the camera and you can't see it (see Figure 31).

(a) Camera moving backward (b) Camera moving forward

Fig. 31. Direction of the camera motion for perspective speed-lines

The shots including speed-lines usually last for a very short period of time since they describe a fast action and it is hard to pay attention to the lines. The demonstration of my thesis, however, shows speed-lines in a shot for several seconds in order to make it easy to demonstrate the results of my methods.

CHAPTER IV

IMPLEMENTATION

I have developed a speed-line program with MEL scripting for Maya animators to use these effects practically in their animation because Maya is one of the software packages that is used commonly in the 3D animation industry (see Figure 32).

Basically, my program makes as many curves as users need right after running the scripts and then moves each point of the curves according to the animation. You are able to adjust the speed-line properties such as line numbers, curviness, length, speed, life, and gap between lines. Users define the ranges and the scripts generate random numbers in the ranges. Once a random number is assigned to a speed-line, the speed-line will keep the property. If a new random number is calculated every frame, it will make Maya run too slow when there are many speed-lines. The same properties of speed-lines will not make the animation look boring. Since speed-lines will be used in very short and speedy scenes, the repeated properties will not be noticeable.

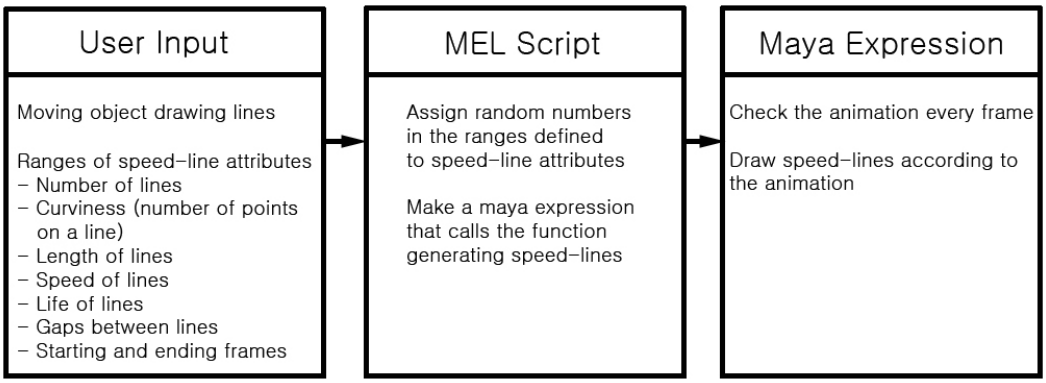


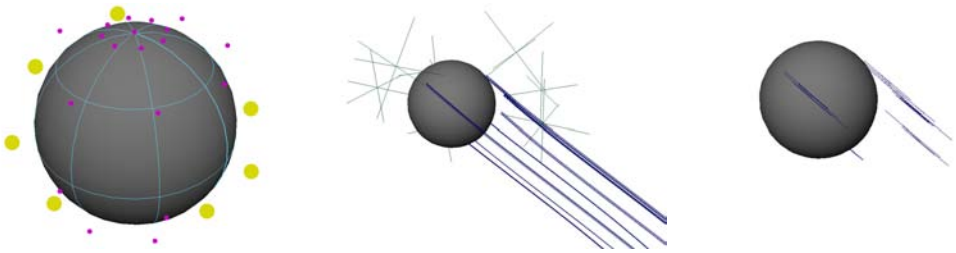
Fig. 32. Basic work flow of the speed-line program

Once the curves are created, you make a brush stroke you want and attach it to

the curves for rendering by using Maya Paint Effects. You can also use Renderman with MTOR (Maya To Renderman) to render the speed-line curves. You add MTOR Curve Width attributes and assign a Renderman shader to the curves. For the result animation shown here, I have used a Paint Effects brush.

A. Speed-lines following objects

The speed-lines following an object are generated around the object and pass by repeatedly during the animation (see Figure 33.c).



(a) (b) (c)

(a) The yellow points are selected

(b) During the first playback, curves and locators are generated

(c) Real-time speed-lines

Fig. 33. Steps of generating the speed-lines following an object

You select some points (or Control Vertices) of a moving object from which you want speed-lines to be drawn (see Figure 33.a). When the script runs, the animation is played once automatically from the beginning to the end for the following purposes (see Figure 33.b):

1. Every selected point draws a curve that traces its path. As a selected point moves, the script adds a position of the point to a curve every frame during the

first playback (see Figure 34). So the trail of the point makes a final curve at the end of the animation. The purpose of the curve is to get the past positions of the point at any time. With MEL scripting, I can get the current position of a point using the pointPosition command, but not the past ones. Once you have the curve, you can get the past positions of the point using the control vertices(CV) of the curve. Let's say the animation runs from frame 1 to frame 30 and you want to get the positions of the point *Sphere.cv[2][3]* at frame 1 and 17, when *Sphere.cv[2][3]* is a cv on the moving object named *Sphere* and *Curve* is the curve that is generated from the selected CV *Sphere.cv[2][3]*. To get the position of *Sphere.cv[2][3]* at frame 1, you can use the current position of *Curve.cv[0]*, which is the first cv on the curve *Curve*, because the frame 1 is the first frame of the animation and *Curve.cv[0]* is the first CV on the curve.

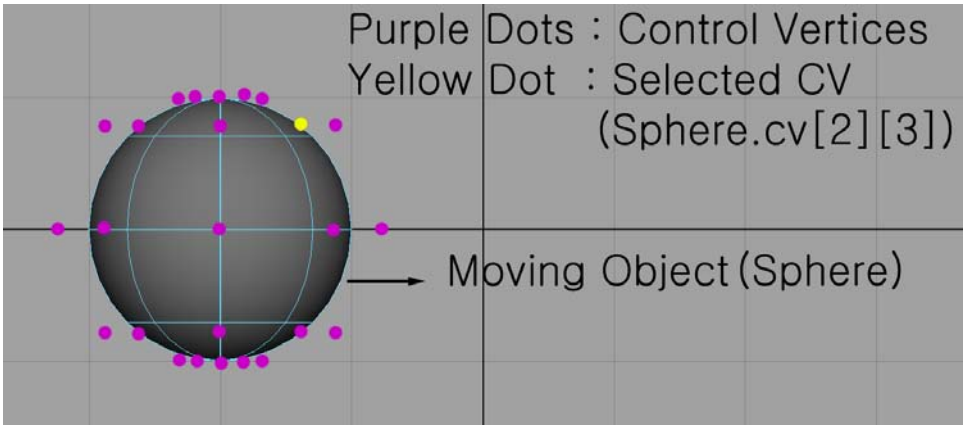
pointPosition Curve.cv[0]

The current position of *Curve.cv[16]* will give you the position of *Sphere.cv[2][3]* at the frame 17.

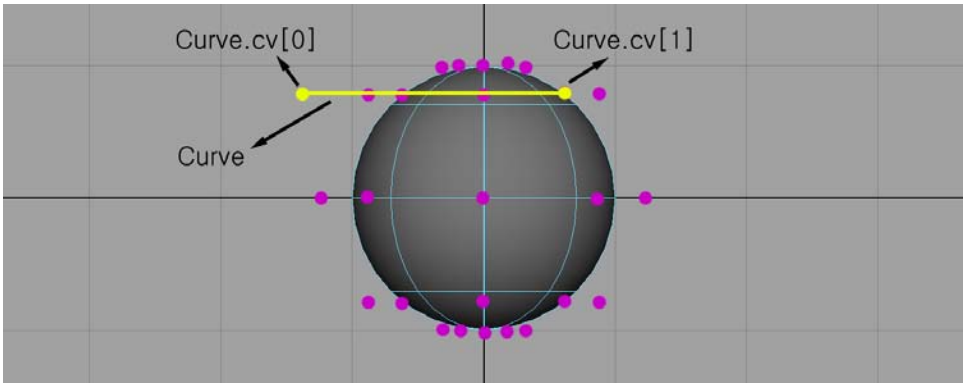
pointPosition Curve.cv[16]

In the Figure 33.b, the long curves are generated by tracing the selected CVs in this way. Once the curves are done, they become hidden to keep the work space simple and clear. You only need the information of CVs on the curves, not the curves themselves.

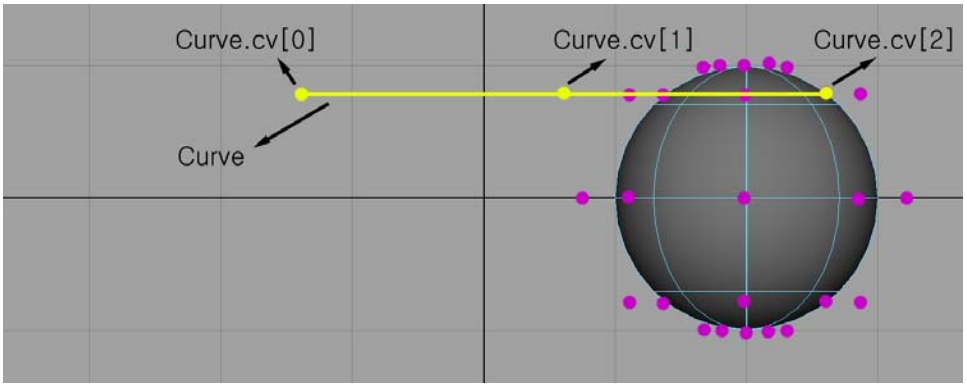
2. A locator is created for each selected CV to find the surface normal from its CV. A locator is created on a selected CV and constrained to the object containing the CV by the normal constraint of Maya (see Figure 35.a). Now the locator



(a) Frame 1



(b) Frame 2



(c) Frame 3

Fig. 34. Curve tracing the path of a selected point

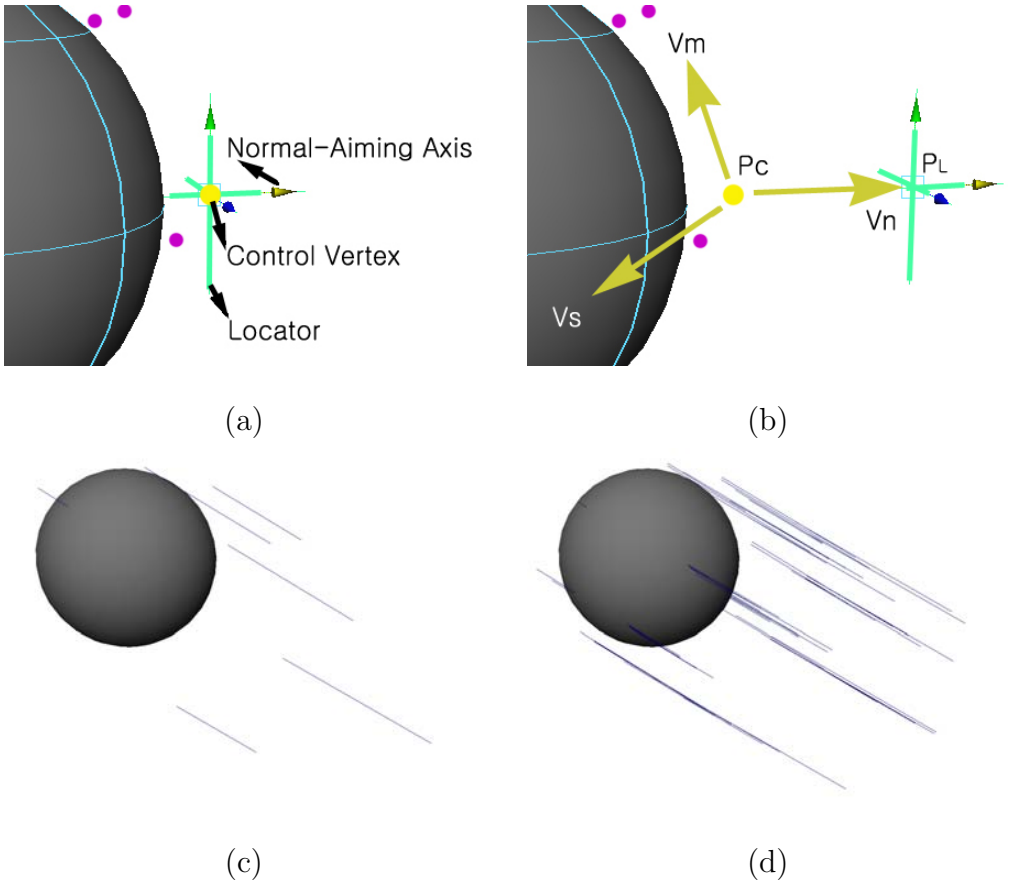
is always aiming in the direction of the surface normal. So I can find out the normal from the selected point. I need this normal to figure out which way to spread out speed-lines (see Figure 35.b).

$$V_n = P_L - P_C$$

$$V_S = V_n \times V_m$$

A single line is not enough to convey sense of the speed, so each selected CV is copied in the spreading direction as many times as you input before running the script. The copied CVs also draw speed-lines like the selected ones (see Figure 35.c and 35.d).

Once the first playback is done for the purposes above, the path curves and the locators will be hidden and you can check the speed-lines in real-time (see Figure 33.c). The script makes a function drawing the final speed-lines a Maya expression. The expression locates the CVs on the hidden path curves and move CVs on speed-lines to the proper positions according to their attributes (see Figure 13).



(a) (b)

(c) (d)

(a) A locator on a control vertex that is normal-constrained to the surface

(b) P_c : Position of the control vertex

P_L : Position of the locator

V_n : Normal Direction

V_m : Moving Direction

V_s : Spreading Direction for More Speed-lines

(c) A single speed-line per a control vertex

(d) Multiple speed-lines per a control vertex

Fig. 35. Locator, surface normal, and spreading speed-lines

B. Speed-lines across the screen

The speed-lines keep moving straight from one side of the screen to the other side. Two scripts are needed for this type of speed-line.

Script 1. You select a camera view to activate and run this script. A plane is created on the center of the screen as in Figure 19.a. You have to adjust the size and the orientation of the plane and the distance between the camera and the plane in order to get speed-lines you want (see Figure 19.b).

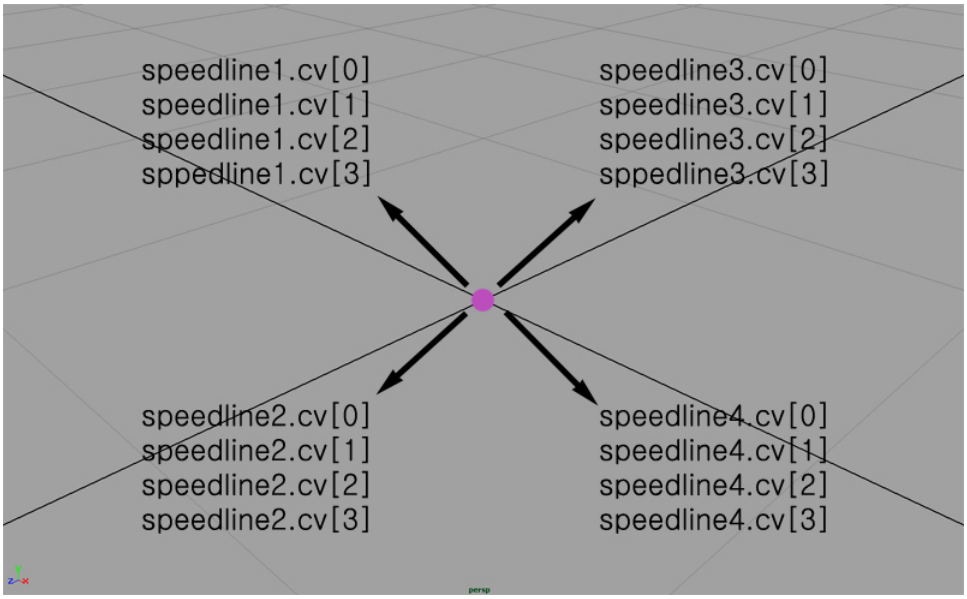
Script 2. Once the plane is set, the directions of the speed-lines are decided from the CVs on one side to those on the opposite side (see Figure 20.a). The attributes of speed-lines such as the length and the speed are assigned to each speed-line with random numbers. The visibility timings of each speed-line are assigned by using the probability variable. In the last part, the script makes a Maya expression to draw the speed-lines. The expression checks the positions of the starting and the ending points on the plane every frame and draws speed-lines as shown in Figure 21.

The plane is parented to the camera, so speed-lines are always in front of the camera whether the camera is animated or not.

C. Perspective speed-lines (fixed camera)

The perspective speed-lines of a fixed camera are drawn from the edges of the screen to the goal object you select, leading attention to the goal object. You select a camera view that you want to render from eventually. You also select an object that will be the goal of the speed-lines. When you run the script for the perspective speed-lines of a fixed camera, it processes the following tasks.

1. The script stores all the information you input such as the name of the goal object you select, the number of speed-line groups, the number of speed-lines in a group, and the maximum length of the speed-lines.



4 speed-lines, 4 CVs for a speed-line

Fig. 36. CVs merged on the origin

2. The speed-lines are created and all the CVs of the speed-lines are merged on the origin, so they are not visible (see Figure 36).

3. The name of the currently activated camera is stored and the attributes of the camera will be used to generate the speed-lines.

4. The script makes a Maya expression that checks the animation every frame and draws the speed-lines, changing their length. The speed-lines are drawn in the following ways:

    - The expression recognizes the position and the orientation of the current

active camera.

- With the up vector and the aim vector of the camera, the expression can get a circle around the camera by rotating the up vector around the aim vector (see Figure 23). The number of CVs on the circle is the same number you input for the number of speed-lines in your scene. When the angle between the up vector and V1 is θ, you can use the MEL command *rot* like this.

$$V1 = rot(V_{up}, V_{aim}, \theta);$$

- The speed-lines are drawn by moving their merged CVs from the origin to their positions between the starting points on the circle and the selected goal object (see Figure 37 through Figure 39).
- The length of the speed-line changes every frame with a random number, which makes the flickering effect.

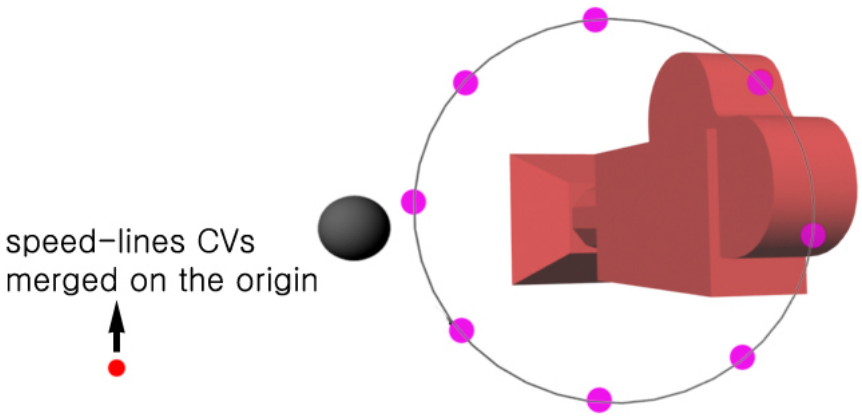


Fig. 37. A speed-line starts from the origin (see Figure 36)

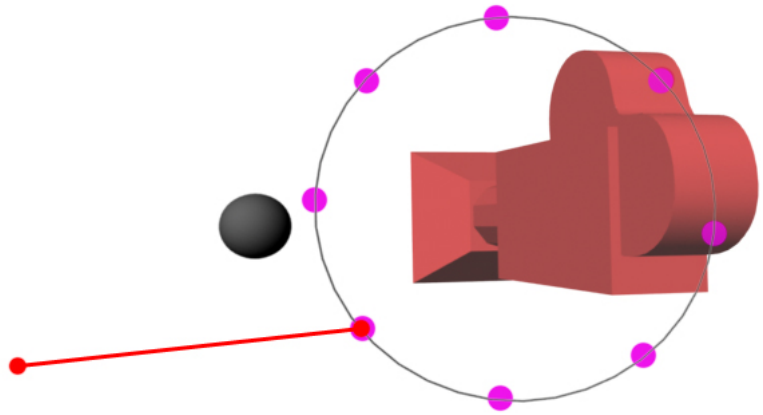


Fig. 38. Starting point moved to a CV on the circle

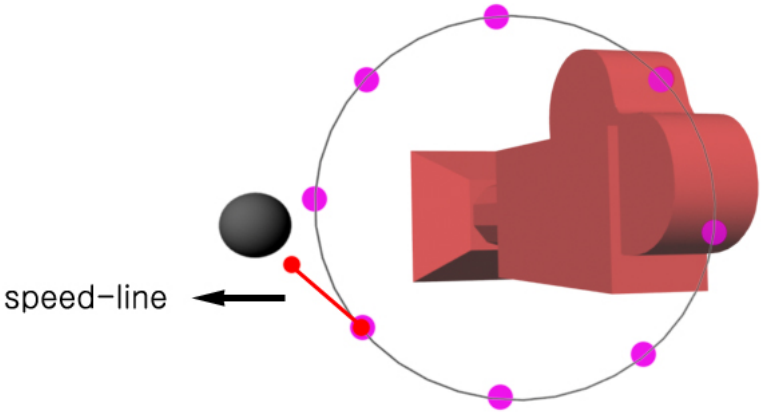


Fig. 39. Ending point placed between the starting point and the goal object

D. Perspective speed-lines (animated camera)

Unlike the perspective speed-lines of an animated camera, you just select a camera view that you want to render from eventually. Then you run this script and it processes the following tasks.

1. You need to input the number of the speed-lines, the number of the groups, the maximum length of the speed-lines, the range of the gaps between speed-lines, the starting and the ending frames of the animation, and the radius of the circle around the camera (see Figure 23).

2. The speed-lines are created and all of their CVs get merged on the origin (see Figure 36).

3. The script recognizes the camera view currently selected and stores the name of the camera.

4. According to the number of the speed-lines you input, the CVs on the camera circle are created as shown in the Figure 25, and the animation is played once automatically drawing path curves (see Figure 27).

5. Once the path curves are created, they are hidden and used to get the past positions. A Maya expression is created at the end of the script and the expression assigns a new group length to every speed-line group every frame (see Figure 26).

6. The speed-lines are drawn from the CVs on the camera circle with a length assigned every frame.

CHAPTER V

CONCLUSION

This thesis presents a 3D animation tool that implements the speed-lines which are usually used in action comics or 2D animation to show fast motions. However, this is not the only purpose for the speed-lines. They are also important graphic elements that make the composition of the images including them very interesting. So I applied the graphic element to the 3D animation (see Figure 40 and Figure 41).

As the result of my thesis, I made MEL scripts for the animators who use Maya, which is one of the most popular softwares in the 3D animation industry. All the animators have to do is to simply input the attributes of the speed-lines and select the camera view from which they want to render the scene. Running the scripts plays the animation once automatically and generates the speed-lines. The users can check the animation of the speed-lines in real time for positions and lengths. Since the speed-lines are just curves without any surfaces, the users have to attach a paint effects brush to the curves to render them out in the final images. It is fairly easy to draw the speed-lines by hand if they are just straight lines, but it is hard for the complicated motions. So this tool is useful for creating the speed-lines when the goal objects or the camera move in curvy lines.

I have worked on only the three types of speed-lines that are most commonly used. Those types are the speed-lines following the objects in motion, the speed-lines across the screen, and the perspective speed-lines. For the perspective speed-lines, I have developed two different scripts for a case when the camera is fixed and for the other case when the camera is animated because the perspective speed-lines look different in those cases. However, the three types are only the beginning. All the cartoonists and the 2D animators have their own styles of speed-line. In addition

to the speed-line, they use various effect-lines to express the emotion, the invisible forces like wind, sound, and so on. So, there are many possibilities to expand this work combining 2D and 3D animations.

The same style of the speed-line could be rendered in many different looks according to its brush stokes. So far I found that the Maya curves can be rendered by using the Maya paint effects brushes or the Renderman shaders, and it could be an interesting work to develop a collection of brushes or shaders for the speed-lines.

Fig. 40. Speed-lines with another cartoon effect

(a) (b) (c)

Fig. 41. Image sequences from the result animation

REFERENCES

[1] M. Narazaki, *Hokusai; The Thirty-six Views of Mt. Fuji*. Kodansha International, 1968.

[2] J.T. Carpenter, *Hokusai and His Age*. Hotei, 2005.

[3] M. Stokstad, *Art History*. H.N. Abrams, 1995.

[4] F.L. Schodt, *Manga! Manga!: The World of Japanese Comics*. Kodansha International, 1983.

[5] H. Honour and J. Fleming, *The Visual Arts : A History*. Abrams, 1995.

[6] P. Cabanne, *Duchamp & Co.*. Terrail, 1997.

[7] D. Ades, N. Cox, and D. Hopkins, *Marcel Duchamp*. Thames and Hudson, 1999.

[8] S. McCloud, *Understanding Comics*. Kitchen Sink Press, 1993.

[9] R.C. Harvey, *Children of the Yellow Kid*. Frye Art Museum in Association with the University of Washington Press, 1998.

[10] R. Goulart, *Great History of Comic Books*. Contemporary Books, 1986.

[11] A. Yoshida, *Elektra: The Hand*. Marvel Comics, 2005.

[12] P. Gravett, *Manga: Sixty Years of Japanese Comics*. Laurence King, 2004.

[13] J. Yang and G. Chun, *Yul-Hyul-Gang-Ho 33*. Daiwon C.I., 2004.

[14] M. Waid and B. Augustyn, *The Flash: Chain Lightning Part 1*. DC Comics, February 1999.

[15] M. Kishimoto, *Naruto Part 1*. Pierrot, 2003.

[16] I.V. Kerlow, *The Art of 3D Computer Animation and Effects*. John Wiley, 2004.

[17] R.L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Proc. Computer Graphics (ACM SIGGRAPH)*, vol. 18, pp. 137-145, 1984.

[18] M. Masuch, S. Schlechtweg, and R. Schulz, "Speedlines - Depicting Motion in Motionless Pictures," *Proc. Computer Graphics (ACM SIGGRAPH, Abstracts and Applications)*, p. 277, 1999.

[19] A. Chen and J.K. Hodgins, "Non-photorealistic Rendering of Dynamic Motion," http://www.cc.gatech.edu/gvu/animation/Areas/nonphotorealistic/, January 2005.

[20] J.K. Hodgins, "Three-Dimensional Human Running," *Proc. IEEE Int'l Conf. Robotics and Automation*, vol. 4, pp. 3271-3276, 1996.

[21] A. McNamara and A. Agarwala, "Visualizing Human Motion in Video," http://www.cs.washington.edu/homes/antoine/videoVis/, January 2005.

[22] B. Kim and I. Essa, "Video-based Nonphotorealistic and Expressive Illustration of Motion," *Proc. Computer Graphics International Conf.*, pp. 32-35, 2005.

VITA

Won Chan Song

Visualization Laboratory

C418 Langford Center

Texas A&M University

3137 TAMU

College Station, Texas 77843-2137

M.S. in Visualization Sciences, Texas A&M University

College Station, Texas 2005.

B.F.A. in Computer Animation, Ringling School of Art and Design

Sarasota, Florida 2002.