AUTOMATED GAIT GENERATION

BASED ON TRADITIONAL ANIMATION

A Thesis

by

SUNG HWA LEE

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

May 2005

Major Subject: Visualization Sciences

AUTOMATED GAIT GENERATION

BASED ON TRADITIONAL ANIMATION

A Thesis

by

SUNG HWA LEE

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

MASTER OF SCIENCE

Approved as to style and content by:

Donald H. House (Chair of Committee)	Ergun Akleman (Member)
John J. Buchanan (Member)	Phillip J. Tabb (Head of Department)

May 2005

Major Subject: Visualization Sciences

ABSTRACT

Automated Gait Generation

Based on Traditional Animation. (May 2005)

Sung Hwa Lee, B.S., University of Illinois at Urbana-Champaign

Chair of Advisory Committee: Dr. Donald H. House

This thesis describes the development of a tool to assist animators in doing walk cycles. In traditional animation, animators create expressive walk cycles with key poses. The process of generating walk cycles by hand is tedious and repetitive. To help animators, many researchers in computer graphics have worked on automating gait generation. However, almost all of them used methods that eliminate animator defined key poses. Although they produce realistic results, their methods are not suitable for expressive walk cycles that can be found in cartoons. The tool described in this thesis attempts to incorporate practices of traditional animators such as comparison of key poses and the use of arc into the program interface. With this tool, animators can concentrate only on setting key poses, which is the most creative task in animating expressive walk. The gait generation program can produce highly expressive walks like the double bounce walk and the sneak. With automated features of the developed tool, animators can save time and effort when animating expressive walk along a curved path.

To my wonderful husband, Sungtae

ACKNOWLEDGMENTS

First, I would like to thank Dr. House for his guidance and encouragement throughout the whole thesis process. I would like to thank Moonsung Lee who helped clarify my ideas as well as provide insight from an animator's point of view. I would like to thank my friends: Wonchan, Julie, Ellen, and Radhika, for their critiques and proofreading. I would also like to thank the Visualization Laboratory staff, especially Glen and Marge who always helped me gladly. Finally, I would especially like to thank my wonderful husband, Sungtae, for putting up with all my frustrations and supporting me in every possible way he could.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

No two walks are the same in reality. From a story telling point of view, a walk cycle can be a very important visual device. They are frequently used for expressing the emotional state and the personality of a character in traditional animation. The study of walk cycles in traditional animation is purely observational, and the poses at key frames play an important role. In his recent book for aspiring animators, Williams [7] explains traditional animation principles for walk cycles in a concise manner. As seen in Figure 1, a walk cycle consists of four very distinct poses that make up the entire motion. They are key frames called contact, up, down, and pass position. These four poses effectively describe every state of each body part during a walk.

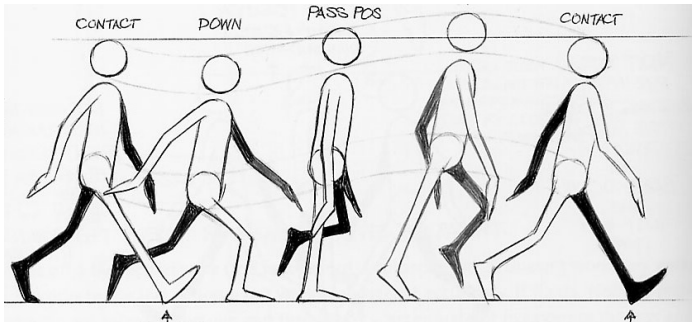


Figure 1: Four key frames of walk cycles in side view. [7]

This thesis follows the style and format of *Proceedings of SIGGRAPH 2002*.

By taking some liberty in changing one or more of these key poses or by breaking rules of the normal walk, an animator can make a very creative walk cycle. Figure 2 shows how much difference changing only one key frame can make in the walk cycle. For example, compared with the other three pictures, the walk shown in the second picture from the left looks sneaky.

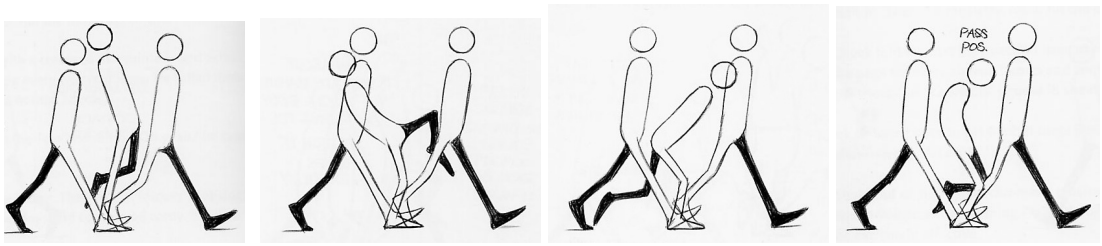


Figure 2: Change in one key frame gives different feeling to the walk cycle. [7]

Designing a walk cycle is very difficult for beginning animators and it still remains a tedious process for experienced animators as well. The motions for a walk cycle are complex, yet very symmetric and repetitive. Many researchers have worked on automating the process through kinematic analysis, physically based simulation, or a mix of both methods. Almost all of these studies have focused on automating rather than the arduous hand process of keyframing.

A lot of researchers on walk generation start with the assumption that eliminating the need for key frames will help animators. However, it is doubtful how useful the resulting studies have been for creating stylized animation. In animation studios, many 3D animators have been educated in traditional animation, where key frames, timing,

and spacing are essential. Omission of key frames is the same as taking away the only way the traditional animators know how to act out through an animated character.

With their emphasis on realistic motion and removal of key frames, the existing gait generation methods lack the expressiveness that is important in stylized animation. Highly expressive walk styles such as sneak and double bounce walk like in Figure 3 are not even possible with those methods. Therefore, they are not directly applicable to the stylized animations to which the principles of traditional animation apply.

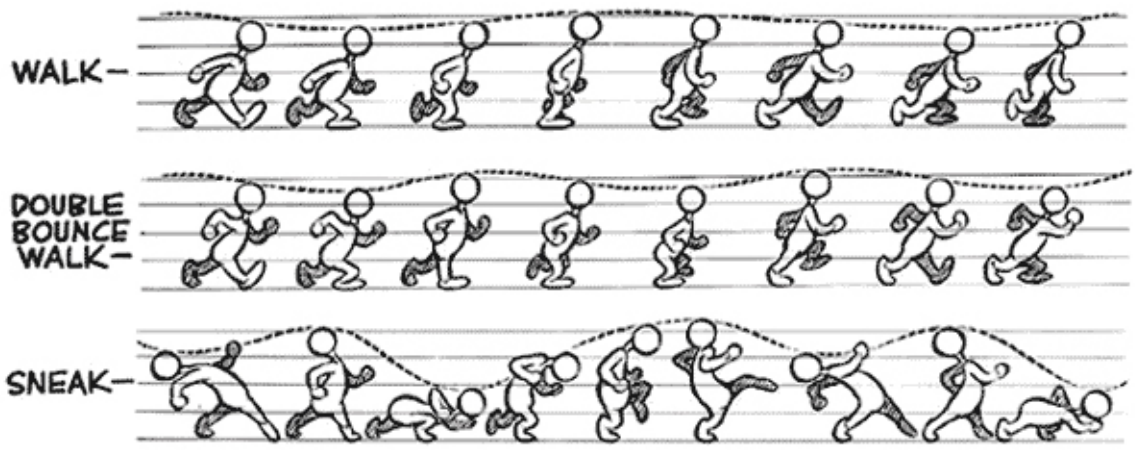


Figure 3: Double bounce walk and sneak compared to normal walk. [1]

The work presented in this thesis takes an animator's point of view of simplifying walk cycle development. The thesis describes the development of a tool that incorporates key frames into automatic gait generation. One significant function of my tool is letting the users see a sequence of consecutive key frames at the same time. To see different key frames, currently animators have to scrub the time line at the bottom of

the screen. With our tool, the animators can compare key poses at different time without jumping to different time segment of animation. This is closer to the way traditional animators have been working and makes it easier to see where to make changes for improvement.

Our experience in using this tool demonstrates this tool is a more effective walk cycle design tool for animators. Our program's interface is similar to the way animation textbooks display the examples. This leads us to believe my tool would be a good for education purpose as well.

2. BACKGROUND

Walk cycles have been studied in various fields such as psychology, computer graphics, robotics, biomechanics, and animation.

Psychological research findings verify a strong correlation between gait and other psychological elements such as personality and dominant feeling [11; 6; 4]. These justify the use of walk cycles as visual communication devices in traditional animation. Human perception on gaits is so remarkable that people can tell a person apart even when the animation was done only with a point light model.

As technology develops, psychologists started using computer program to examine human perception. Cutting's research [3] is one example. Based on the kinematical observation, Cutting [3] developed a system to automatically generate walk cycle animation using a simple point light model. He summarizes the characteristics of walk cycle with the ratio of the pelvis movement to the shoulder movement. By changing the ratio he generated different walk cycles. For the given point light model, he found out when the ratio is close to 4 or ¼, the walk cycle animation looks extremely masculine or feminine, respectively. His paper is a good introduction to the kinematic analysis of human walk cycles and the difference between feminine and masculine walk cycles. It also shows how to summarize motion characteristics using as few parameters as possible.

Researches on walk cycles in computer graphics have two general approaches: physically based or kinematic. Since physically based approach accounts for gravity,

muscle torques and the various surface reaction forces, it has obvious advantage of producing physically realistic motion in the resulting animation.

Influenced by research in robotics and biomechanics [5], dynamic analysis has been used in computer graphics research to achieve realistic and natural movement [4]. Although realistic, these methods require the animator to specify motion in terms of forces and torque, which is not very intuitive or easy. Physically based methods are computationally expensive as well.

To overcome the shortcomings, Bruderlin and Calvert [2] used a mixed approach of kinematics and dynamics, resulting in simplified computation and interface. By taking inputs such as velocity, step length, and step frequency, their system derives *step symmetry* and *state-phase-timings* from locomotion. These two concepts are illustrated in Figure 4 and Figure 5. Their system assumes step symmetry that is to say the angles between each leg and the vertical line are the same. In Figure 4, this means $\theta_1 = \theta_3$ in each step. Figure 5 shows state-phase-timings that specifies durations of stance and swing phases. As an outcome of this derivation, the forces and torques that drive the motion of legs are determined by numerical approximation techniques.

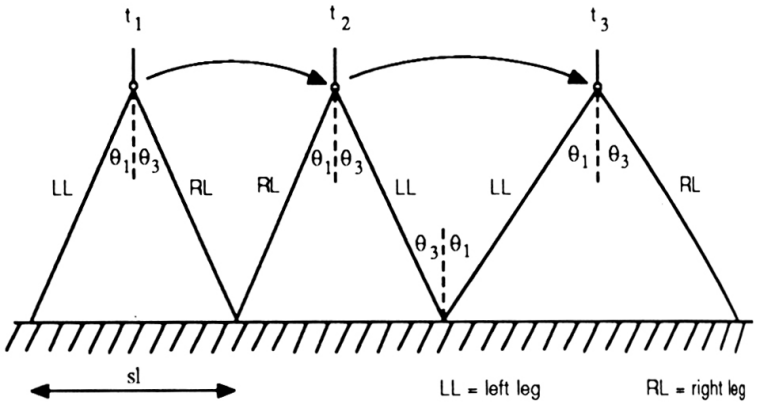


Figure 4: Symmetry of compass gait for different step lengths. [2]

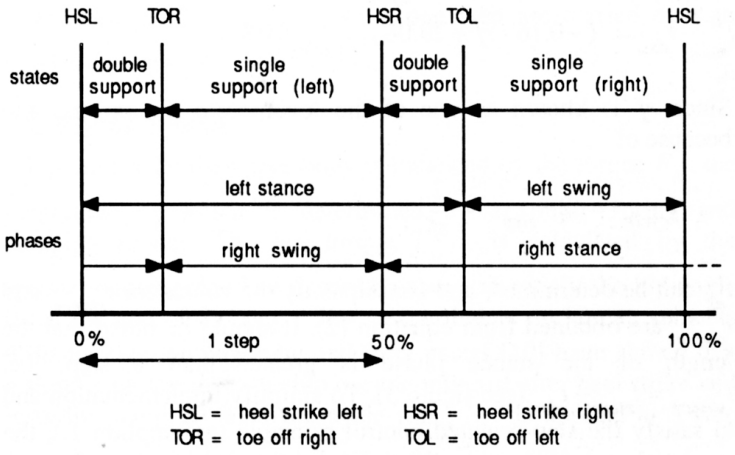


Figure 5: Locomotion cycle for bipedal walking. [2]

Sun and Metaxas [6] also used a mixed approach with different parameters when generating walk cycles. The walk followed a curved path on uneven terrain. To simplify computation, they used sagittal elevation angle motion representation. As in Figure 6, Sagittal elevation angle is the angle between the vertical axis and a limb segment's

projection onto sagittal plane, which is a vertical plane that bisects the figure into left and right halves. As illustrated in Figure 7, the calculation is simplified since the figure's heading direction describes sagittal plane's world orientation and the direction of walking. By changing orientation of sagittal plane over time, curved locomotion is achieved. This method greatly reduced the necessary number of data sets required for curved locomotion.

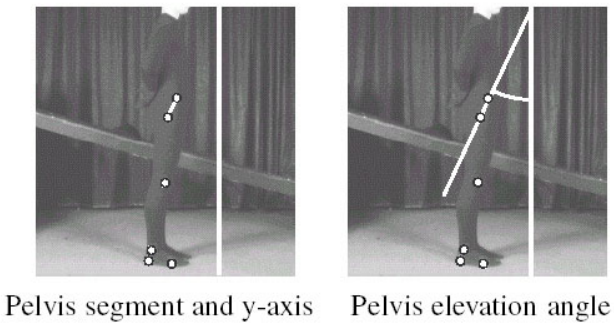


Figure 6: Measuring the pelvis sagittal elevation angle. [6]

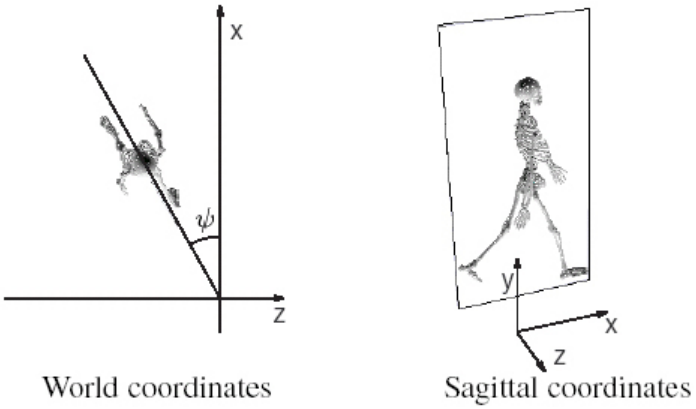


Figure 7: The sagittal plane. [6]

Although successful in achieving their goal, Sun and Metexas' work is geared toward usage in real time animation or automatic non-interactive motion generation. Unlike Bruderlin and Calvert [2], they do not allow the users to control the step length.

3. METHOD

3.1 Overview

The Maya Embedded Language (MEL) was used for the development of a practical walk generating program for *Maya*(http://www.alias.com) animators. MEL extends *Maya*'s functionality by allowing usage of *Maya* functions and skipping its graphical user interface. A *Maya* animator will be able to generate gait animation with the following steps.

First, the model file has to be imported by referencing. The model has to be a biped character, facing in the positive z-axis direction, and rigged with a rigging system, *Final Rig*. It can be downloaded from Highend 3D website (http://www.highend3d.com) or the developer's website (http://www.radiantsquare.com). *Final Rig* is used to apply the walk generating system to different characters easily and quickly.

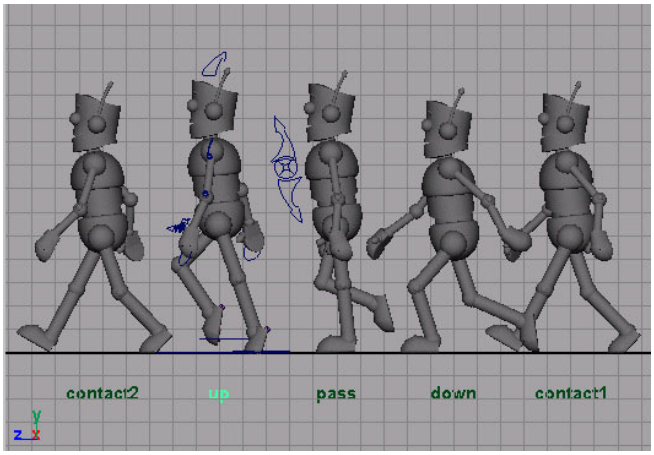


Figure 8: Interface of our program for key poses.

In the second stage, the user will set each key pose the interface of the walk generating program. When the program starts with a given model file, it shows five pose models for each pose as in Figure 8. Contact, down, pass, up, and next contact poses are loaded by default. To set a key pose, the user needs to click on a pose name. Clicking on a pose name activates the pose by moving the animation model to the given pose model location. After setting up a pose for one key pose, the user can click on another pose name to work on a different key pose. Notice that five key poses generate one step, that is only half the cycle since a full walk cycle includes two steps. Display modes such as sliding, toggle X ray, and "show only selected pose" are available in the walk generating program. Also, functions such as "copy pose from A to B" and "bake pose" were developed to help the animation process.

When all the key poses are set, the user can move on to the next stage, full walk cycle generation. Walk cycle usually refers to an animation clip that looks like the character walks in place without physically advancing in space when the clip is looped. The character looks as if it is walking on a treadmill. To make a full walk cycle, the second step in the walk cycle has to be made. 3D animators typically do this by copying and pasting key frames by going through each joint or each control object of the rig. For the second step, they also have to figure out which joint to copy from and which joint to paste to since the opposite arm and foot come forward and each joint with rotation needs to change its direction. This process is automated in the walk generating program so that a full walk cycle is generated with just one click.

Functions such as Arc and Shift are provided to improve walk cycle animation easily. Arc is a function that traces the path of an object or joint. Shift is a function that offsets keyframes of objects for smoother movements.

The last stage is generating a walk along a curved path. Using the walk cycle animation and a path the user provides, the program calculates where the character should be placed to walk along the path.

3.2 Rigging

Typically a rig contains joints, bones, and control objects. By investing time in rigging, time spent on the animation process can be saved significantly. A control object is a simple form of geometry that is constrained to the skeleton. Figure 9 shows a model with its control objects annotated. These objects are usually associated with a joint or a set of joints for preset movements. When animating, a lot of times it is difficult to select the joints under model's geometry surfaces. By having control objects placed just outside the model's surface, the user can easily select any joint.

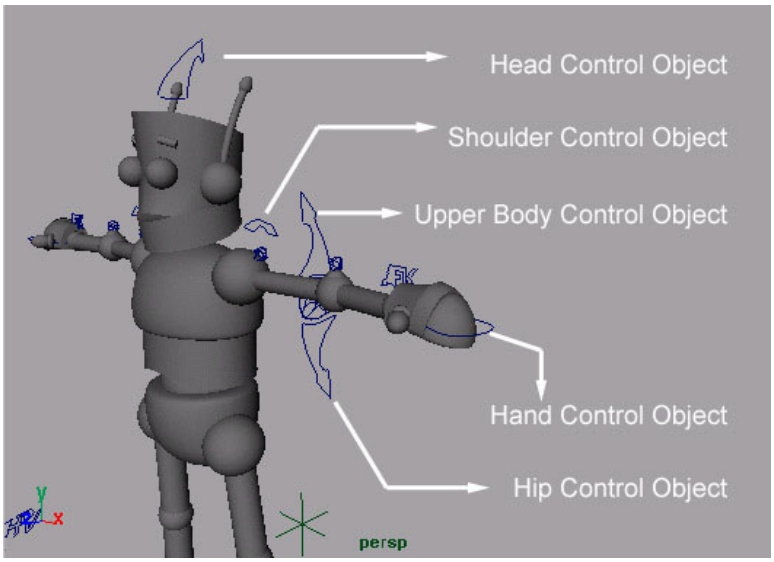


Figure 9: Examples of control objects.

Control objects often hold the parameters of preset movements as their attributes. For example, a control object associated with hands usually has a grasp or grab attribute, which is connected to all finger joints. Instead of selecting each joint for animating grab, all the user has to do is to set a value for the grab attribute of the hand control object. In many studios, animators are encouraged to set keyframes only to these control objects. That way animation data can be preserved even when some change such as the removal of joints in the skeleton is necessary later.

Final Rig allows the user to specify the joint location by moving provided ball markers. As seen in its graphical user interface, shown in Figure 10, it also allows control over the number of joints for fingers, spine, neck, and so on. Once a rig is created using *Final Rig*, the user can manipulate the joint either by clicking on the *Final Rig* user interface or by selecting the control objects in a view panel.

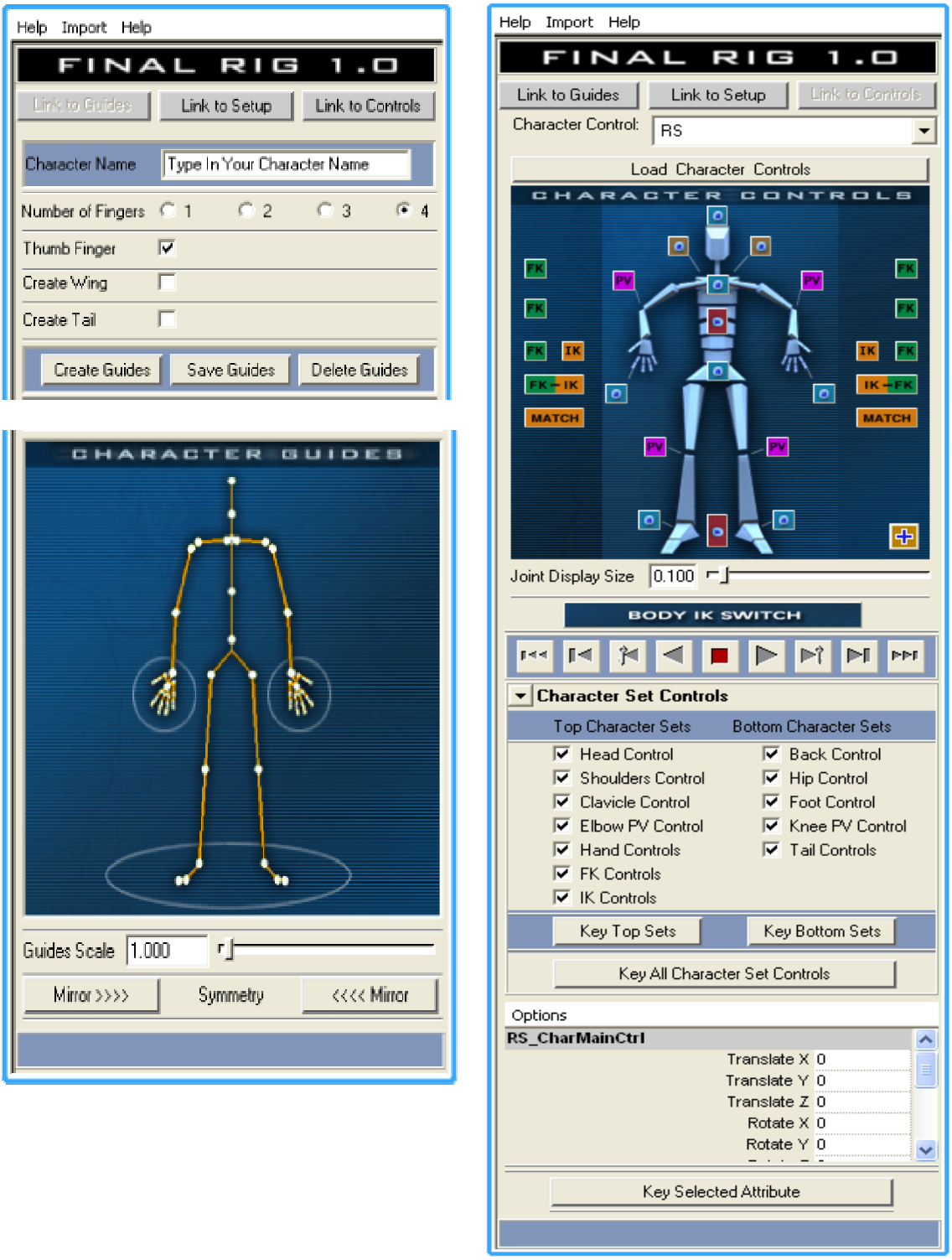


Figure 10: *Final Rig* interface.

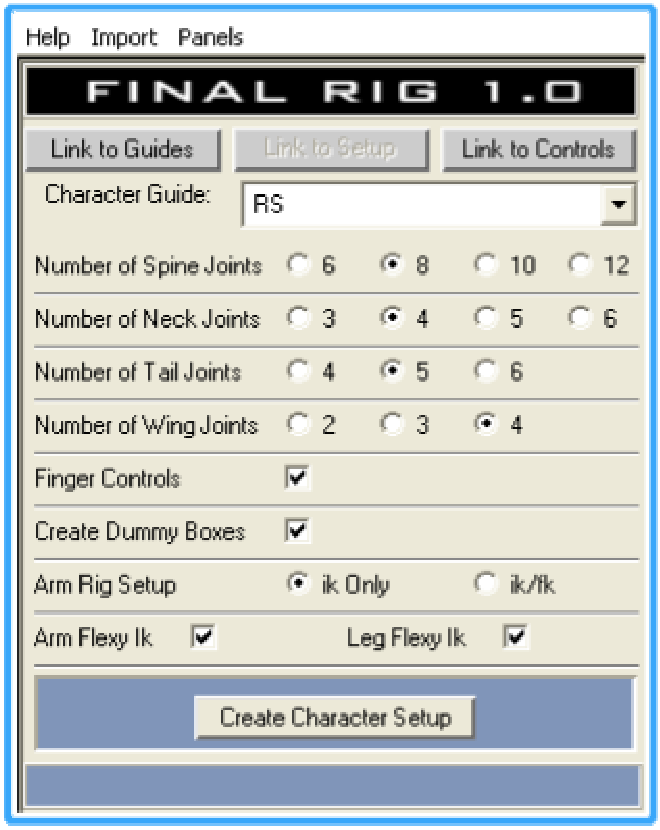


Figure 10: Continued.

Final Rig has a character setup that is common in the 3D computer graphics community. The root of the rig is located at the center of pelvis. It provides an inverse kinematics (IK) setup for the foot and an option for IK or Forward kinematics (FK) for the arms. A rig contains a hierarchy of joints, and a movement is defined by supplying rotation angles or translation values for the joints. Specifying movement by supplying rotation angle for each joint is called forward kinematics. In contrast, inverse kinematics

uses an effector that connects a joint with another joint to calculate the location of the joint in the middle.

A rig generating system not only allows for a quicker and easier setup for characters, but also provides a stable programming environment since the possibility of the user's mistake in rigging is unlikely. The rig system generated by *Final Rig* needs to be slightly hand modified to accommodate programming needs. These modifications are very easily done. They include enabling attributes that were hidden originally or inserting expressions for certain attributes of a node in the rigging system.

3.3 Setting Key Poses

The walk generation program has several features, some of which are borrowed from traditional animation methods. Most 3D software programs including *Maya* already support direct manipulation of a model to set a key frame instead of an interface full of parameter names and numeric input fields. 3D animators are accustomed to it. By utilizing this function, numeric inputs from the user have been minimized.

Direct manipulation of a model not only fits the current workflow of animators, but also it makes a much more efficient interface for visually oriented artists. Moreover, key poses constitute a concise visual representation of the complex numeric data required to describe a walk. When an animator sets a key frame, the necessary numeric data can be extracted from the key pose. The numeric data can be used to generate a gait along a path.

3.3.1 Interface

The fundamental purpose of our program is to provide an interface for easy comparison of the key poses. Traditional animators always compare key poses side by side or through the use of a light box. This practice is incorporated into the program by having an option for displaying all key poses at the same time. Figure 11 shows the result.

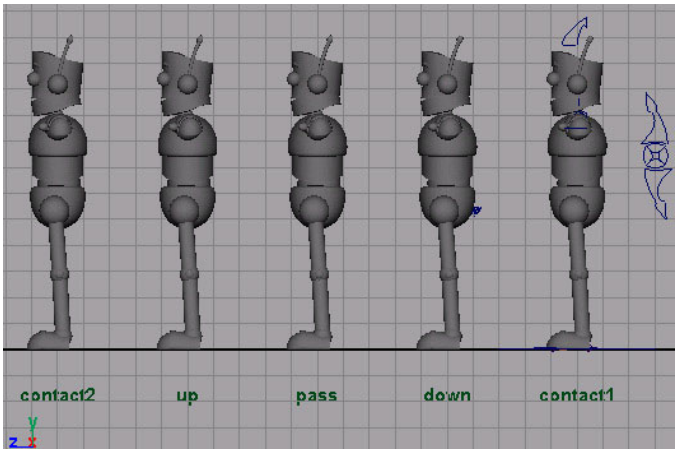


Figure 11: Walk generation program interface when the program is started.

When program starts, the user is prompted to select a file that contains a model to be animated. Then, the user needs to enter the number of frames per cycle in the GUI that looks like Figure 12. This value determines how fast the character walks. According to Williams [7], a normal walk takes 1 second per step. That is translated to walking on 12's in traditional animation, since the frame rate for cel animation is 24 frames per second (fps). For computer animators, normal walk speed can be translated to 24 frames

for film production or 32 frames for NTSC or television production. The default value is 24 fps.

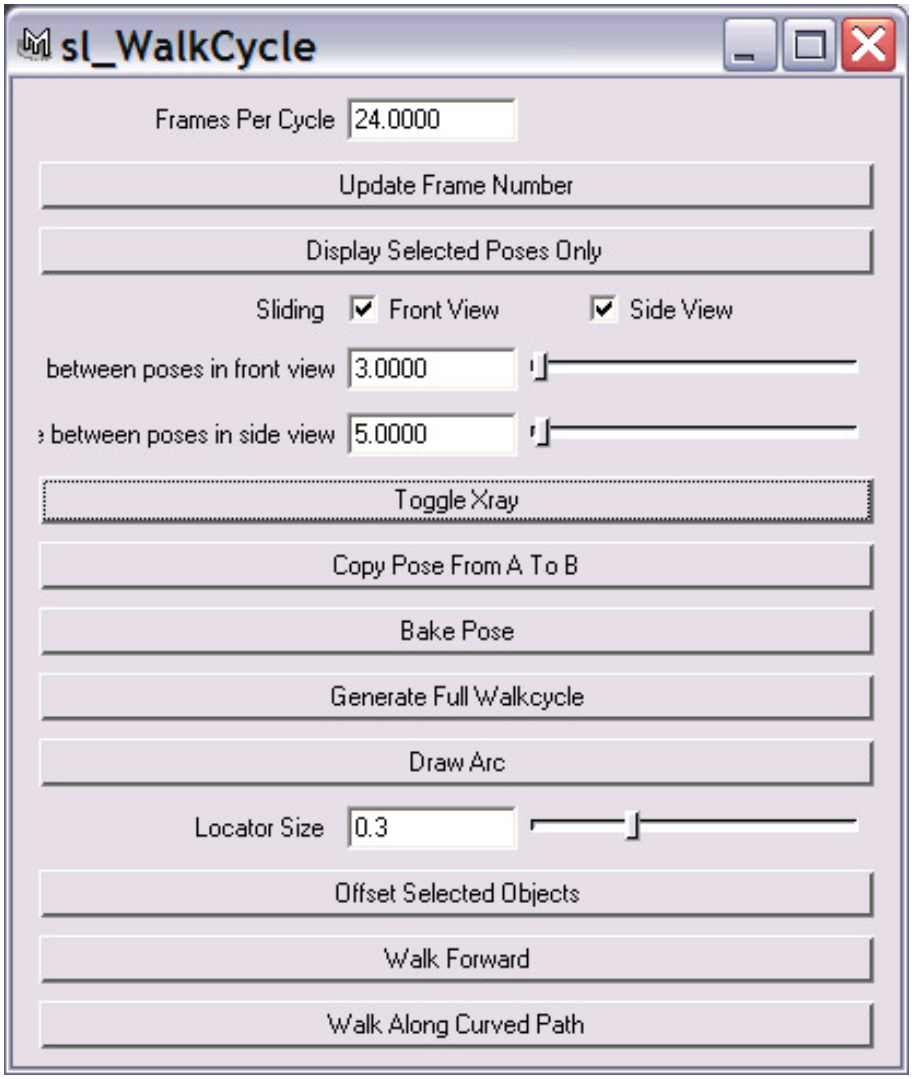


Figure 12: GUI of walk generator.

Right after the program is loaded, its interface looks like Figure 11. It shows five key poses that make up one step. The contact, up, pass, down and the next contact poses

are shown at the same time. Since no key poses are set up yet, all poses look the same. They are displayed at the same time even though their order is sequential in the time line. In order to achieve this, some similar functions that are already available in *Maya* were researched. As a result of the functions' slow speed, a new solution was needed.

In the first attempt, the model file was referenced once for each pose. That is five times in total. Although it was straight forward to implement, having five fully rigged models was demanding on system resources, resulting in slow playback speed in the viewport. Slow display speed drastically undermines the intuitive process of animation.

The problem was solved by reducing the number of fully rigged models to two. One referenced model was used for animation the whole time and the second referenced model was used for the display of mirroring the contact pose in real time. Therefore, the second referenced model is displayed only when one of the contact poses is selected. For clarification, the first reference model is called the animation model, and the second one the contact reference model. Figure 13 shows these two models. The picture on top shows the animation model when up pose is activated. Notice it shows only one fully rigged model. The bottom picture shows both of the two rigged models when contact pose is activated. Contact1 pose shows the animation model and the contact2 pose shows the contact reference model.

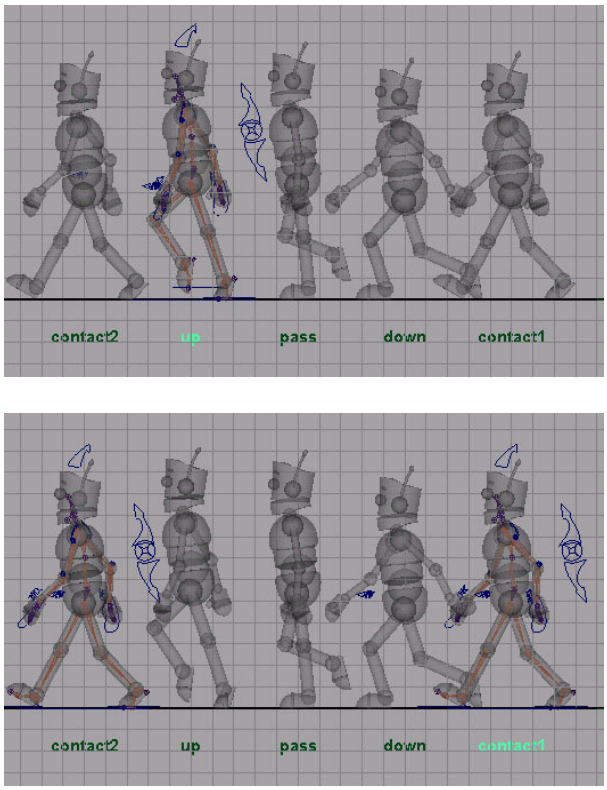


Figure 13: Animation model and contact reference model.

3.3.2 Storing Key Poses

When the model file was referenced as many times as the key poses, each key pose model held the data until full walk cycle generation. However, referencing model files twice does not allow it. Since the contact reference model was purely reference only, the storage problem of the key pose data was solved by setting keyframes to the animation model from the beginning. However, keyframes set sequentially in time in one character, does not yield the display of key poses all at once. To work around that,

geometry was “baked” for each pose. When a pose is activated, the “baked” model for the activated pose was replaced with the animation model.

3.3.3 Activating a Pose

In Figure 13, notice each model has a pose name written at the bottom. Made of *Maya*’s annotation node, these names work as buttons for selecting a pose. The annotation node also holds the frame number as an attribute, so that the corresponding frame number can be retrieved from this node when a key pose is activated.

To “activate” a key pose, the user has to select or click on one of these pose names. This is implemented with *Maya*’s scriptJob command, which allows running a script whenever a specified event occurs. The following code in Figure 14 shows the line scriptJob command that was used in walk generating program. When the user selects something, the SelectionChanges state is met. Then, scriptJob command runs the switchToAnimationModel function. Pseudo code for the switchToAnimationModel procedure is also explained in Figure 15.

```
scriptJob -event "SelectionChanged"  "switchToAnimationModel";
```

Figure 14: ScripJob command.

< switchToAnimationModel Pseudo Code >

1. Check if the selection is one of the pose names.
2. If it is not, do nothing. If it is, do the following.
3. If the previous pose name is the same as the selected pose name, do nothing. If it is not, do the following.
4. Set a key frame for the animation model at the previous pose.
5. "Bake geometry" for the previous pose.
6. Move the animation model to the selected pose by changing the current time to the frame number of the selected key pose.
7. Delete backed model of the selected pose.
8. Set the selected pose as the previous pose.

Figure 15: Pseudo code for the switchToAnimationModel function.

Selecting a key pose name means going to a different frame, so the animated character appears at the specified key pose location. After switchToAnimationModel is run, the user can start setting up a key pose for the selected pose.

3.3.4 Mirroring of Contact Pose

As shown in Figure 13, our interface provides two contact poses to help animators, one at the beginning and one at the end of the sequence. This way the animators can compare poses that are closer to the second contact pose. These two

contact poses are exactly the same except for the fact that they are mirrored. The opposite side's arm and foot are forward at the second contact pose. When the user changes the contact pose, the change is reflected in the other contact pose in real time. As it was stated earlier, the contact reference model was referenced to accommodate this, and it only appears in the viewport when one of the contact poses is activated.

Our program requires the character model in the model file to face the positive z direction, and the world up vector to be y axis. That means the character's motion should be mirrored about the yz plane. At first, mirroring was needed only for the paired control objects such as hands and feet. However, after linking attributes of the contact reference model's control objects to those of the animation model's opposite side control objects, mirroring was not done properly. Careful inspection revealed that, besides switching sides, entering negative values for some of the attributes is necessary. Also even if the control object is not paired, it is still affected by mirroring.

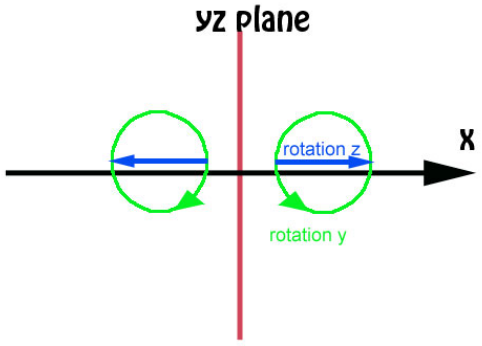


Figure 16: Mirroring rotation direction.

Figure 16 shows how rotation should be mirrored. Viewing from the top, the red vertical line corresponds to the z axis. The positive y-axis is coming out from the screen. Figuring out how to mirror translate values was relatively easy since only translate x value was affected. Translate x attribute of the contact reference model can be calculated by negating the translate x attribute of the animation model's control objects. As depicted in Figure 16, the rotation directions should be countered when mirrored. Negating also accomplishes countering rotation. In summary, to mirror motion about the yz plane, translate x, rotation y, and rotation z should all be negated.

Real time updates of the contact reference model were done through expressions. An expression is a segment of run-time code that uses mathematical functions to relate object attributes. The multiplication code was inserted in every affected attribute of the control objects in the contact reference model. Switching the side of control objects was also done if the control object was paired.

3.3.5 Display Modes

To help animators set up key poses more easily, display modes such as spread out, sliding, toggle X ray, and "show only selected pose" are available in the walk generating program.

Spread Out

A function for spreading out key poses was implemented as well to avoid overlapping of one key pose with the next one for clear comparison, so that the visual

comparison of key poses, as in Figure 1 of the introduction, is also possible in a 3D software program. Figure 17 shows the pose models spread out with different distances between the poses. A simple expression that calculates multiple of the distance between poses is inserted into the parent node of each key pose. The spread out function is available for both the front view and the side view, and the distance between poses can be set through the GUI that is shown in Figure 12. Since it is implemented with expressions, real time run code, changing distances between poses using slider in GUI updates the scene in real time.

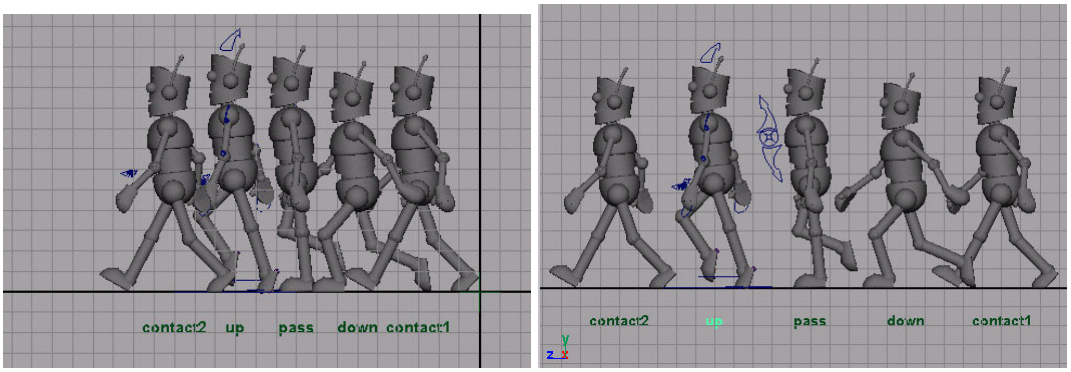


Figure 17: Spreading of key poses. Distance between poses is 3 for the left picture and 5 for the right.

Sliding

The animation model stays at the origin by default. Sometimes it is useful to see the animation model going through each key pose or to place the animation model between two key poses as in Figure 18. The user can see how the animation model is

interpolated between the key poses. A sliding option has been provided to enable animators to do this. Sliding is implemented by adding an expression to the translate attribute of the animation model's root. The expression is shown in Figure 19. The code has been edited for easier read.

Figure 18: Sliding of the animation model.

```
expression

-string (
  "if (sliding){
            $root.translateZ = $distance_btwn_poses + $distance_btwn_poses / ($halfFramesPerCycle / 4) *
             frame;
    } else {
            $root.translateZ = 0;
    }")

-name "slideBetweenPoses";
```

Figure 19: Expression for sliding.

When the sliding option is on, the animation model slides through the different key poses. When the contact pose is at frame 0 and the down pose is at 3, with the distance between poses at a value of 3, the root of the geometry model for these two poses are placed at (0, 0, 3) and (0, 0, 6), respectively. Between frame 0 and 3, with sliding on, the root of the animated character will be moving from (0, 0, 3) to (0, 0, 6). The resulting animation looks as if the animated model is sliding between key pose geometries.

Toggle X Ray

Sometimes having five copies of the model geometries can be too crowded in the viewport. To make the animation model stand out, inactive poses can be displayed in X ray mode. X ray mode is a shading mode that makes the geometry transparent as in Figure 20. X ray mode can be applied to a selected object by displaySurface command. Also it can be toggled on and off.

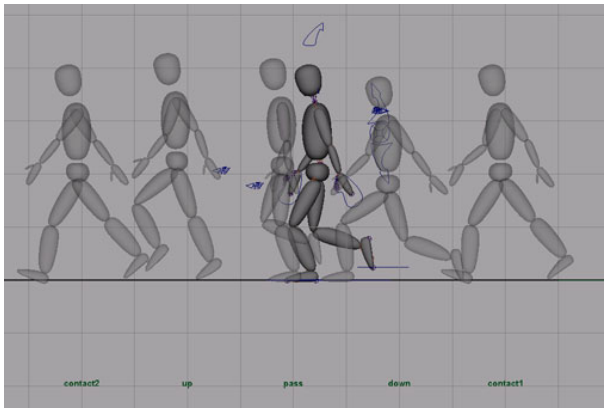


Figure 20: X ray mode for inactive poses enabled.

3.3.6 Animation Functions

Two functions that are related to setting key poses are available. One is the copyPoseFromTo and the other is the Bake Pose function. When setting key frames, it saves a lot of time to start from the key pose that is already set instead of starting from scratch. The copyPoseFromTo script allows that functionality. As input, the copyPoseFromTo function requires two key pose names; the first key pose is where the attribute values are copied from and the other pose is where copied values are to be pasted. The script goes through every attribute while copying and pasting. Copying key poses gets extremely fast with this script.

Baking the pose is essentially copying or duplicating geometries of the character at the given time. The SwitchToAnimationModel function has an internal call to this function, so the user does not have to bake a pose each time there are changes to different key poses. However, the scriptJob command that enables the call to switchToAnimationModel is not excuted when the animation is played. Sometimes, that leaves one of the key poses missing in the viewport. Therefore, there is manual access to this function. It is recommended to run this function before playing the animation.

3.4 Generating Full Walk Cycle

Since key poses in the previous stage only had half of the walk cycle, the other half needs to be generated. Since the other half of the walk cycle consists of key poses

that are mirrored from the poses in the first half of the walk cycle, each key pose is mirrored in the way the contact reference model was mirrored from the animation model, and then key framed in the appropriate frames. However, this is not done through the expression. Mirroring takes place in the background by manipulating keys. Since this is implemented with one function call, the users can go back to setting the key pose stage by undoing the "generateFullWalkCycle" command. Some things are only detectable by looking at actual animation. The users can play back the full walk cycle, and if they spot room for improvement, they can go back and rework the key poses.

3.4.1 Draw Arc

To ensure fluid movement, traditional animators use arcs for visual reference. They are also used to design an object's path on a projected plane, which helps to effectively communicate the action or the story. This practice is continued among 3D animators in the industry by tracing the movement of an object frame by frame on the computer monitor with erasable markers.

Williams [7] illustrates the use of arcs for walk cycles in Figure 21. Two distinct arc shapes are present in a walk cycle. One is for the wavelike motion of the arm and the head. The other is for the path of the heel and the toes which looks like a figure eight from the side view.

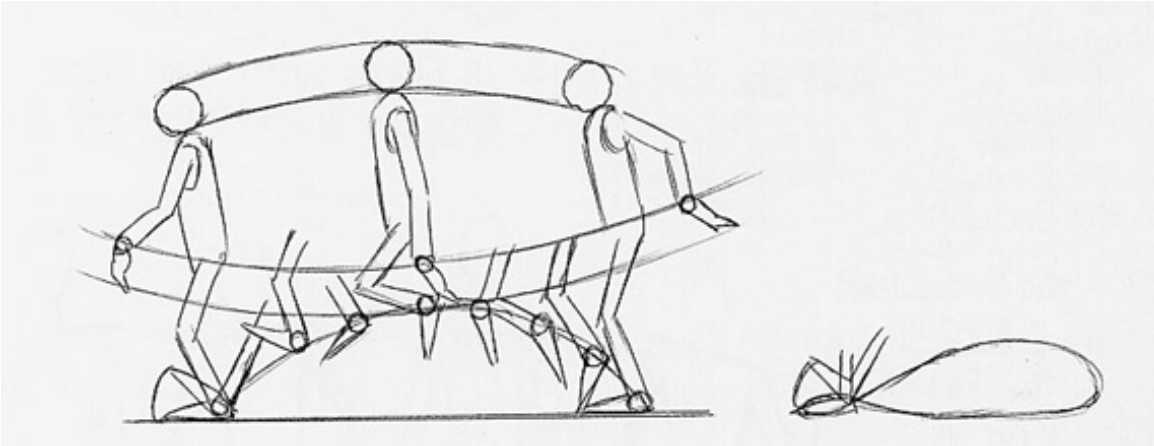


Figure 21: Use of arcs in a walk cycle. Williams[7]

Arcs are implemented by tracing a selected joint in every frame. Our program generates an arc using the traced joint location as control vertices of a curve.

3.4.2 Shifting (Offsetting)

Somehow CG characters produce a robotic feeling in their movement when every body part is hitting a pose at the same time. We can make the body parts flow into a pose by offsetting key frames of different body parts. This gives a smoother motion and more organic feeling.

To run the shift script, the user needs to select the objects in the order they are offset. We let the user specify the offset amount for each object through a dialogue box. The dialogue box had to be made on the fly since there is no way of knowing which

objects will be selected. The dialogue box looks like Figure 22. In the figure, the offset chain of the arm control objects is shown. The selected objects are clavicle, shoulder and elbow controls, which are selected in the typical order for offsetting.

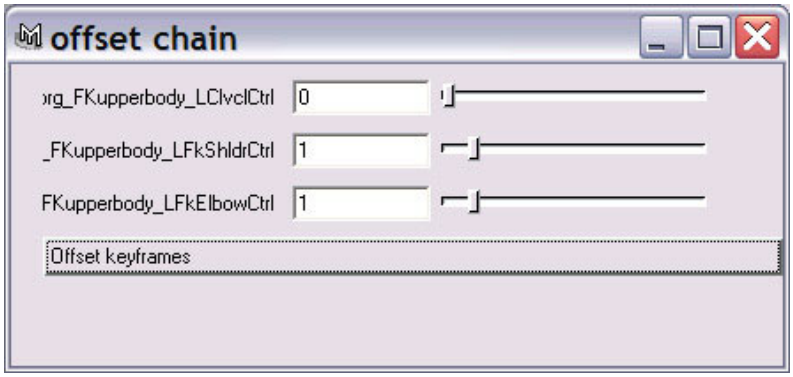
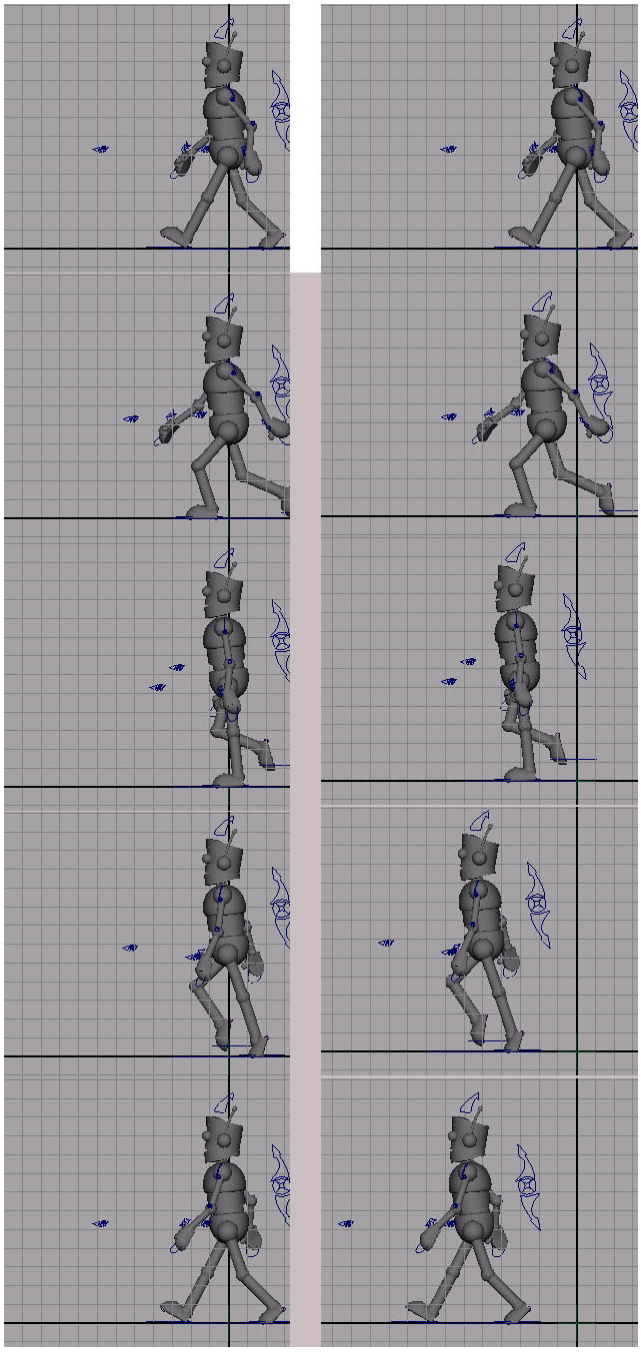


Figure 22: Offset dialogue box.

3.5 Generating Gait Along a Curved Path

Once a walk cycle animation for a character is done, the character looks like a person walking on a treadmill as seen in Figure 23(a). Making it walk forward can be done by careful attention to the stance foot's movement. We can make the character walk forward as in Figure 23 (b) by moving the root of the character in a way that the stance foot does not slip. Therefore, in the resulting animation, the stance foot stays in one place during each step.



(a) not moving (b) moving

Figure 23: Frames of a walk cycle with and without the root moving.

In Figure 23, notice that between the contact pose and the down pose, the stance foot rotates about its heel. From the down pose until the pass pose, the stance foot does not rotate. It stays put, but after the pass pose, it gets lifted. This time the stance foot rotates about the ball joint until it reaches the next contact pose.

To make it simpler, we can say the stance foot rotates about its heel during the first half of the step with a rotation value 0 between the down pose and the pass pose. During the second half of the step, it rotates about its ball. Therefore, to make the stance foot stay at one place, the heel joint has to stay at the same place for the first half of the step, and then the ball joint for the next half of the step. Figure 24 shows the code for calculating the translation values for the root to accomplish this.

The script assumes the rigged model is facing the positive z direction, and the translate attributes of the root, that were hidden originally, are activated. That way, walking forward means moving toward the positive z direction. We keep track with only translate z values of the heel and ball joints. Since the joints' translate values are always in the local coordinate system, the xform command was used to determine the translate z values of the joint in world space.

```
//Code for walking forward: edited for readibility
{
 float $prevPivotLocation[], $curPivotLocation[], $pathPivotLocation;

 float $endFrame = `playbackOptions -q -animationEndTime`;

 for ($i = 0; $i< $framesPerFullCycle; $i= $i+1){
       if ( 0 == $i){
             currentTime $i;
             $prevPivotLocation = `xform -q -ws -rp $stanceFootHeel`;
             $pathPivotLocation = 0.0;
             setKeyframe -time 0 -value $pathPivotLocation CharMainCtrl.tz;
       }
       else if (($i >0) && ($i <= $framesPerHalfCycle)){
             currentTime $i;
             $curPivotLocation =`xform -q -ws -rp $stanceFootHeel`;
             $pathPivotLocation = $pathPivotLocation + $prevPivotLocation[2] - $curPivotLocation[2];
             setKeyframe -time $i -value $pathPivotLocation CharMainCtrl.tz;

             $prevPivotLocation = $curPivotLocation;
             if ($framesPerHalfCycle==$i){
                         $prevPivotLocation = `xform -q -ws -rp $stanceFootBall`;
             }
       }
       else if ($i >$framesPerHalfCycle){
             currentTime $i;
             $curPivotLocation =`xform -q -ws -rp $stanceFootBall`;
             $pathPivotLocation = $pathPivotLocation + $prevPivotLocation[2] - $curPivotLocation[2];
             setKeyframe -time $i -value $pathPivotLocation CharMainCtrl.tz;

             $prevPivotLocation = $curPivotLocation;
       }
 }

  float $offset=0.0;

 for ($i =0; $i < $endFrame; $i = $i + $framesPerFullCycle){

       if($i == 0){
            copyKey -time "0:$framesPerFullCycle" CharMainCtrl.tz;
            currentTime -update no $framesPerFullCycle;

            $offset = 0;
             float $temp[]= `keyframe -query -eval CharMainCtrl.tz`;
             $offset=$temp[0];
       }
       else{
            currentTime $i;
             pasteKey -time $i -option insert -valueOffset $offset CharMainCtrl.tz;
            $offset += $offset;
       }
 }
}
```

Figure 24: Calculating translation values.

The script above looks at the translate z value at the previous time step and the current time step and compensates for the difference in the translate z values of the heel or ball joint by adding the difference to the translate z value of the root. In this case, the root is a control object named CharMainCtrl. The compensated values are stored as keyframes for CharMainCtrl.translateZ. Once the script has calculated the appropriate translateZ values for CharMainCtrl for one step, these values are repeatedly accumulated for the rest of the animation until it hits the specified end frame for the character to walk forward continuously. The translate z values for a given time increment are stored to be used as a basis for the curved walk.

Generating a gait along a straight line that is not parallel to z direction can be done the following way. The gait is thought of as being rotated by theta from the gait along z axis. As illustrated in Figure 25, the root location *(x, z)* can be calculated with simple trigonometry.

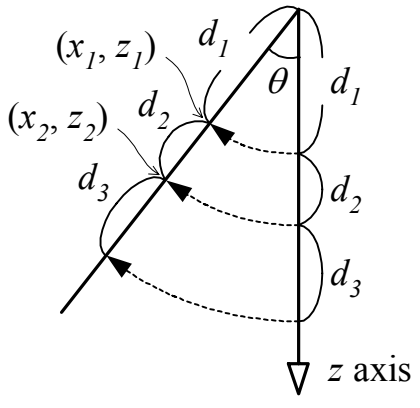


Figure 25: Root coordinate calculation.

1. Calculate the angle between the straight line path and the z axis.
2. Calculate tx and tz coordinates with trigonometry

Let's say d_1, d_2, d_3 and so on are the amounts the root has translated in a given time segment. Since the path is being treated as a rotation from the z axis, the line segments' lengths stay the same.

Then, $x_1 = d_1 \sin \theta$

$z_1 = d_1 \cos \theta$

and

$x_2 = x_1 - d_2 \sin \theta$

$z_2 = z_1 + d_2 \cos \theta$

Therefore,

$x_n = x_{n-1} - d_n \sin \theta$

$z_n = z_{n-1} + d_n \cos \theta$

The algorithm for following a curved path develops further from here. For the given curved path, a linear approximation of the path is calculated first. The step is treated as walking on a straight line within each line segment. Just like Sun and Meteaxas' [6] use of sagittal plane, only the rotation Y attribute of the root was used to make the character turn.

Since each line segment can be treated as a straight walk rotated from the z axis, the coordinate of the root can be calculated the following way.

$x_n = x_{n-1} - d_n \sin \theta_n$

$z_n = z_{n-1} + d_n \cos \theta_n$

,where θ_n can be easily calculated using coordinates of the two end points of the line segment. As the character moves, the rotation value was updated with θ_n. For smoother transition of the values, keyframes for θ_n were not set for the first half of the n*th* step and the second half of the *(n-1)th* step.

As rotation Y attribute of the root changes for turning, the whole character rotates. This means the stance foot gets rotated as well. To make the stance foot stay in the place it should be, a constraint was used.

4. RESULTS AND EVALUATION

The evaluation method used in this thesis is informal. Gaits were animated with and without the developed tool. Based on personal experience and records kept during the process, the two animation processes are compared. The gaits generated by hand-animation and by the developed tool look almost identical in the results, but the process was very different.

Figure 26 shows the Maya workspace as it was configured for developing the walkcycle. The graph editor fills the bottom half of the window and the timeline is the strip of frame numbers. To move to different frames, the user needs to scrub in the timeline.

With hand-animating, some planning is needed before starting to animate. After deciding the pace of the walk cycle, the animator needs to keep track of the frame numbers for each key pose. If the walk is normal speed, for example, 24 frames are needed for the full cycle. Each pose is keyframed equally apart in time. The contact, down, pass, up, and the second contact poses are placed at frame 0, 3, 6, 9, and 12, respectively. Frame 12 is the mirrored pose of frame 0. Mirrored poses of frames 3, 6, and 9 was used to fill frames 15, 18, and 21. Also, the first contact pose needs to be copied to frame 24 for proper interpolation and continuity back to frame 0. When cycling the animation, either frame 0 or frame 24 has to be excluded. Otherwise, the resulting animation does not have smooth motion around the contact pose.

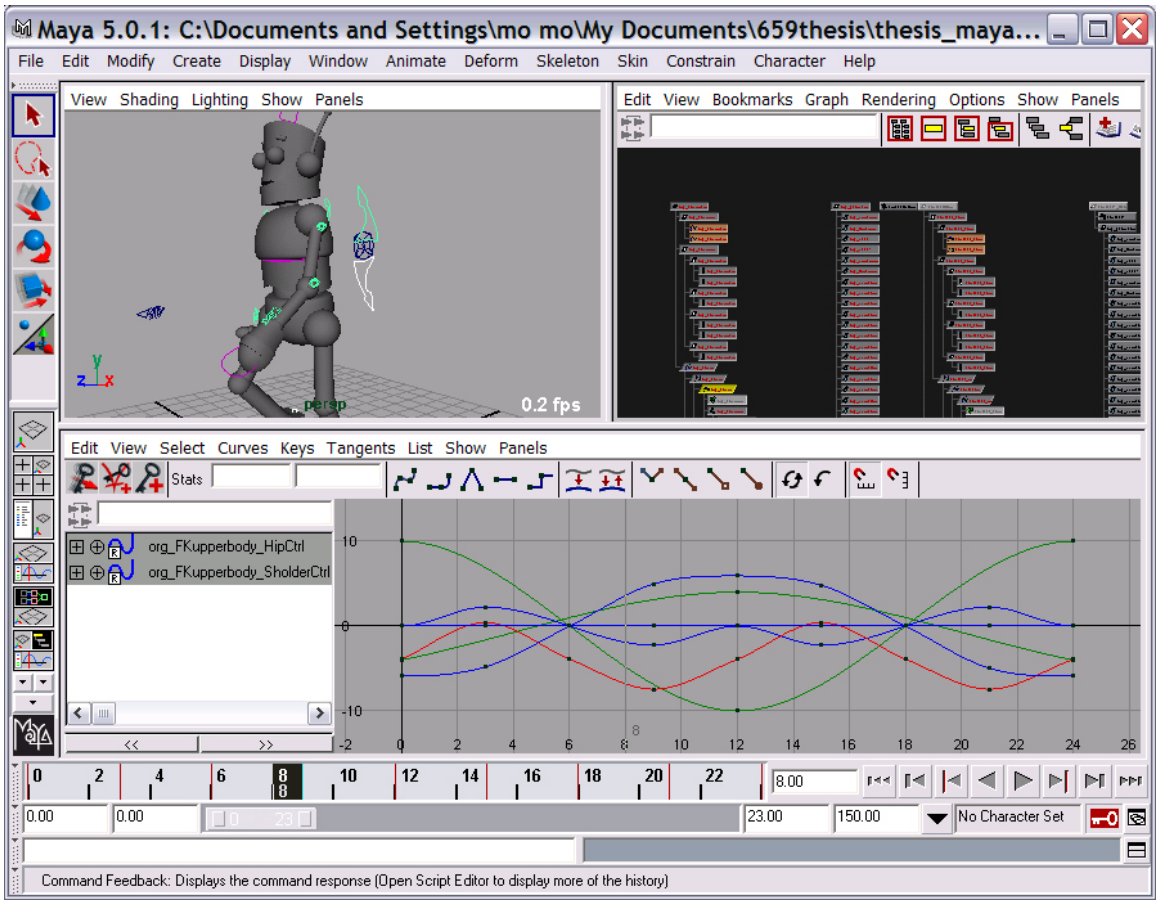


Figure 26: *Maya* interface showing graph editor and time line across bottom.

Mirroring a pose can be done either by scaling in the graph editor in *Maya* or entering negated values for each attribute of control objects. These two methods have shortcomings. The first method lacks accuracy and the second method is cumbersome. Although there are some time saving tips such as “copy and paste” in the timeline after selecting all control objects, careful attention and planning were still needed.

When setting up key poses, it was necessary to frequently scrub in the time line. Since it was difficult to remember how the previous pose looked, scrubbing had to be done to compare poses. It was tedious and distracting.

Using the tool described in this thesis, the *Maya* setup looks exactly the same as Figure 26 except the fact that the model viewport looks like Figure 27. This interface enables keeping track of the frame numbers for each pose. The program also takes care of mirroring automatically. Being able to concentrate only on setting key poses expedited the process.

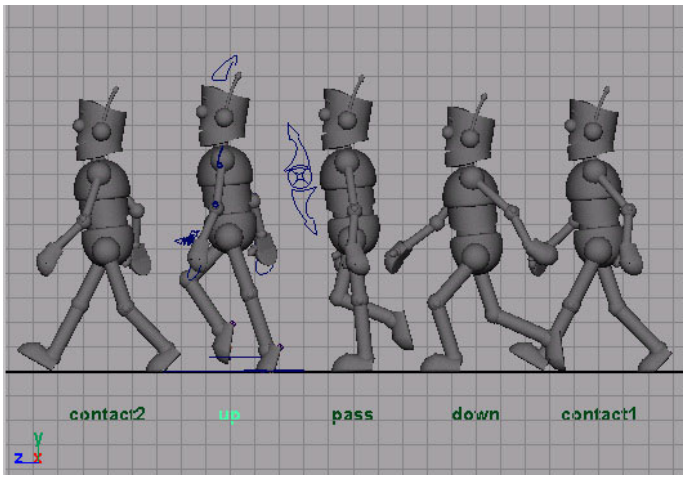
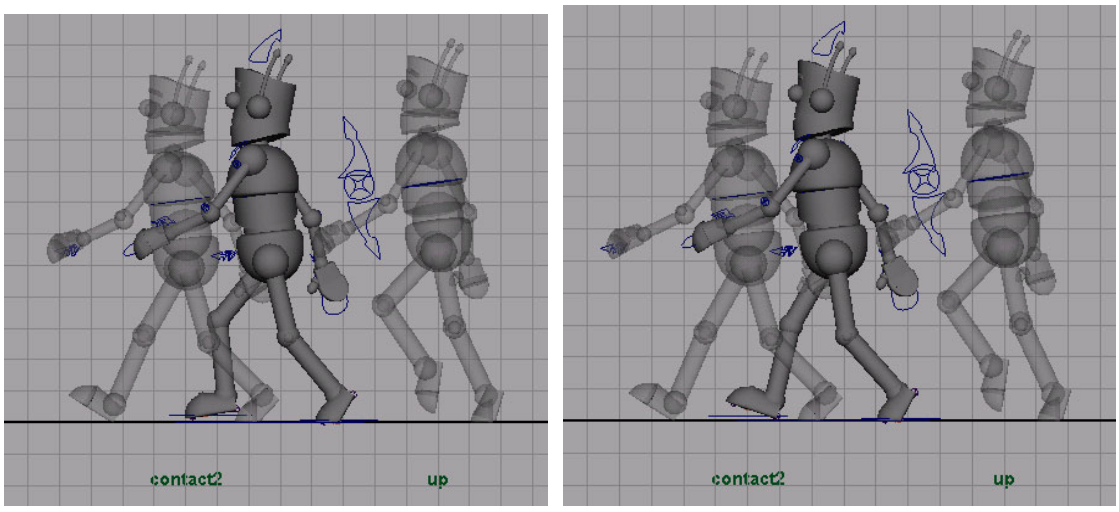


Figure 27: Interface of the developed program.

Comparison of the poses was very convenient with the developed program. Since all key poses are shown, not only the comparison between the previous pose and the current one but also the comparison between all poses and the current one was possible. That made it easier to determine the proportionate amount of movement needed.

For example, when setting up down pose, it would be nice if the pelvis does not move down too much compared to the amount the pelvis moves up at the up pose. Since the up pose does not come immediately after down pose, this kind of comparison is difficult with hand-animating.

Manual refinement for movement was possible throughout the entire process. For example, in-between poses can be set with our interface. In-between pose is literally a pose that is needed in between key poses. It does not define the movement, but it is needed to clean up jerkiness or to smooth the transition in the motion between key poses. Figure 28 shows an example of in-between pose. The front foot had jerkiness between the second contact pose and up pose. With the sliding option on, the animation model is placed in between the second contact pose and up pose so that the front leg can be compared. In this case, the animation model is placed at the frame immediately before the seoncd contact pose. From the comparison, it was discovered that the motion was jerky because the knee moved too much around the second contact pose. In addition, the interpolated rotation angle of the foot was wrong. The front leg motion was fixed by matching the leg of the animation model closely to that of the second contact pose. Least amount of scrubbing was needed to complete this task.

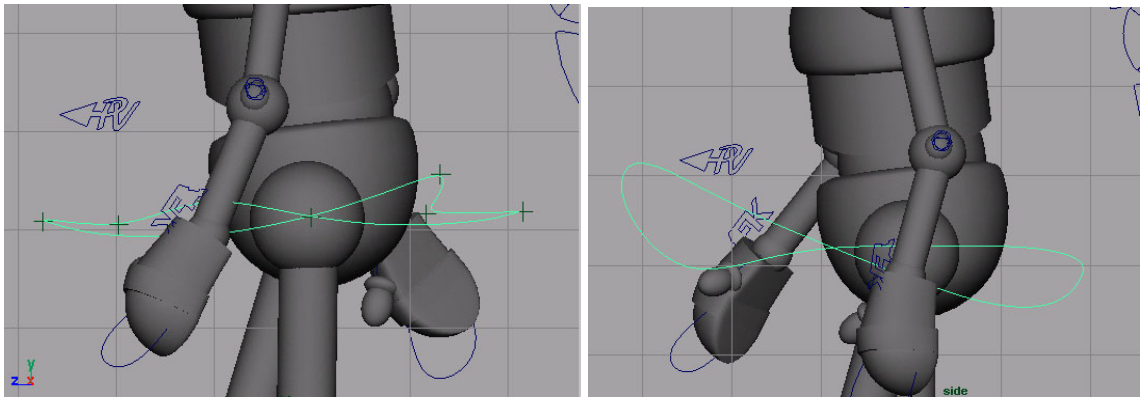


(a) Before (b) After

Figure 28: Setting up an in-between pose with the interface.

Work style changes as animation evolves. Due to the way it is programmed, the user has to click on the pose name whenever changing key poses. Since there is no need to keep track of the frame numbers, the initial set up of key poses was convenient and intuitive. However, when setting up key poses are complete in large part and a lot of tweaking is needed, clicking on a pose name and then clicking on a control object became cumbersome because of the frequent change of the poses. For tweaking, having a control object selected all the time and going back and forth between the key poses while scrubbing in the timeline was more convenient. When the user concentrates on the motion of one body part, changing key poses by scrubbing in the time line and "baking a pose" after tweaking seem to be a better use of the tool.

Another big help was the use of the arc function in the developed tool. Playing the walk cycle animation in a loop helps detecting unnaturalness of the motion. However, sometimes it is very difficult to determine the cause of the strangeness when the character is constantly moving. The use of arc was helpful in that situation, since it shows the trajectory of an object. Figure 29 shows how the arc was used. The arc in the pictures traced the wrist joint with the key frames coming from the wrist joint control object. The crosses on the arc tell where the key frames are set, revealing which key poses need tweaking. The arc of the wrist should appear like the number 8 in shape, and not curved like in Figure 29 (a). Figure 29 (b) shows the trajectory after some changes. The arc looks like an 8 and the jerkiness in the arm's movement disappeared.

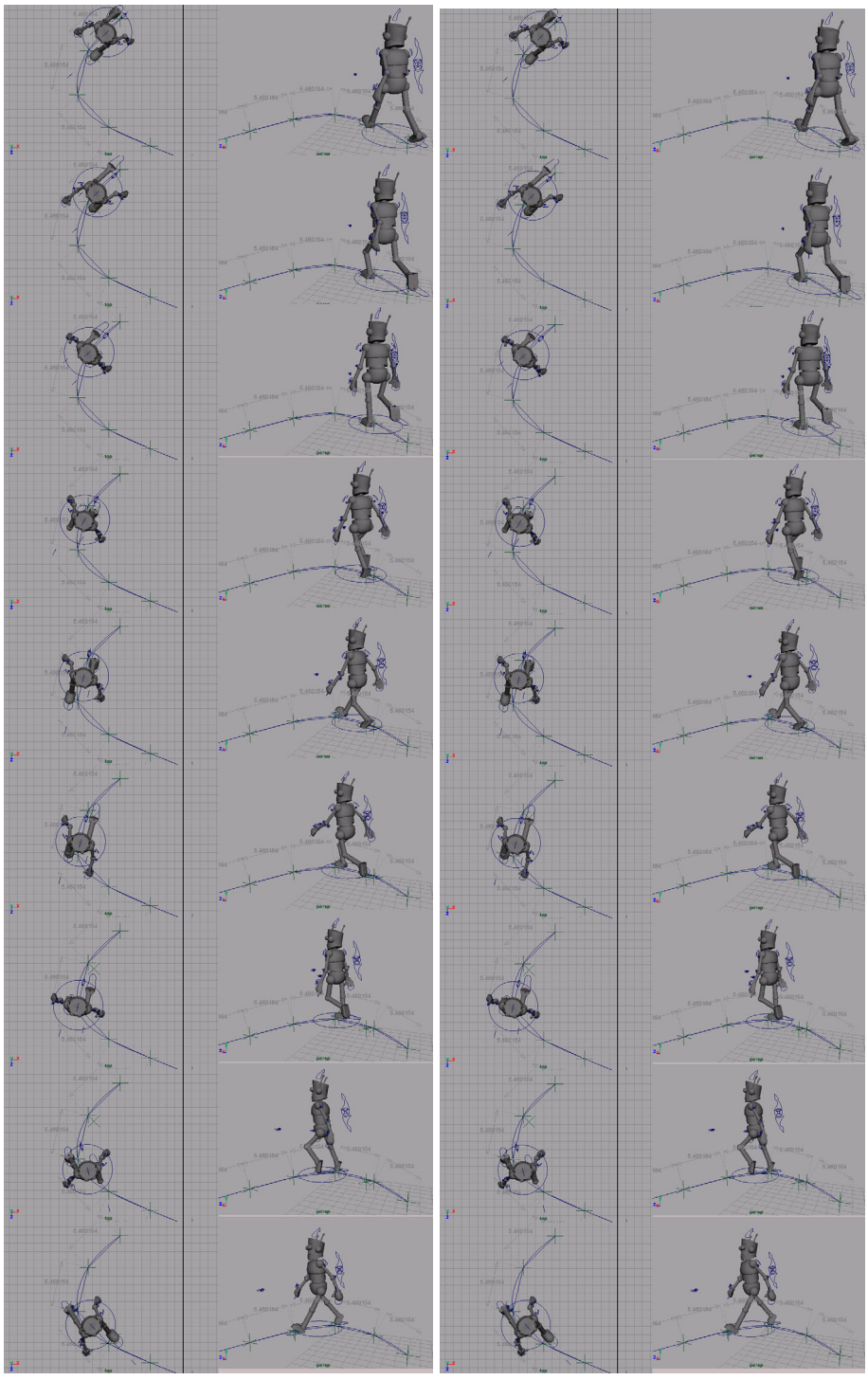


(a) Arc shape when there is jerkiness in motion. (b) Arc shape when the jerkiness is gone.

Figure 29: Example use of arc.

Although the ease of comparing key poses is the strong point of our tool, considerable time was saved in other parts of the program such as automatically

generating the full walk cycle and gait generation along a curved path. When done without the developed tool, it took nearly five hours to animate eight steps along a curved path. Since stance foot was calculated automatically with our tool, it took the least amount of time to have the character walk along a curved path. As shown in Figure 30, the results are indistinguishable.



(a) hand-animated (b) animated with the developed tool

Figure 30: The gaits along a curved path.

5. CONCLUSION

A gait generating program for traditional animators was developed and tested. Traditional animation practices such as the display of all key poses for comparison and the use of arc were incorporated in the development of the tool, as well as other helpful functions. Repetitive and tedious tasks are automated, leaving the creative decision to the animator.

Ease of comparison of the key poses helped the animation process. Automating features such as generating a full walk cycle and generating gaits along a curved line reduced the time needed for walk animation significantly. Moreover, the tool developed is not restrictive thus its use can be easily combined with the current workflow of 3D animators. This can lead to a better and faster result.

From the comparison of the gaits that are animated by hand and that are animated with the developed tool, it is shown that using the walk generating program it is possible to produce animation of quality similar to that produced using standard methods, but more efficiently.

Although very convenient, our tool works only with the rig generated by *Final Rig*. It would be a good future work to expand the usage of the program with other rigs as well. This could be achieved by letting the user specify the control objects. To determine how to mirror the paired control objects, a function that tests how they are setup might be useful. With some modification and improvement of the program, the use of the program can be expanded to animating runs.

The quality of the animated outcome with the walk generating program, heavily depends on an animator's creative decision. If the animator is an absolute beginner, the outcome will not be wonderful, but it will be better than the hand animation by the same animator. That leads us to think that the developed program can be effectively utilized as a teaching tool. Since animation is usually taught traditionally with hand drawing, it could be taught using 3D tools. That allows a beginning animator to concentrate on learning motion not acquiring meticulous drawing skills.

REFERENCES

[1] P. Blair. *Cartoon Animation*, Walter Foster Publishing, 1994.

[2] A. Bruderlin and T. Calvert. Goal-Directed Dynamic Animation of Human Walking. *Proceedings of ACM SIGGRAPH 89*, pages 233-242, July 1989.

[3] J. Cutting. Generation of synthetic male and female walkers through manipulation of a biomechanical invariant. *Perception 7,* pages 393-405*,* 1978.

[4] M. McKenna and D. Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. *Computer Graphics SIGGRAPH 90 Conference Proceedings*, pages 29-38, September 1990.

[5] M. H. Raibert. Legged Robbots. *Communications of the ACM, 29*, 6, pages 499-514, June 1986.

[6] H. Sun and D. Metaxas. Automating Gait Generation. *Proceedings of ACM SIGGRAPH 2001*, pages 261-269, August 2001.

[7] R. Williams. *The Animator's Survival Kit*, Faber and Faber Limited, 2001.

VITA

Sung Hwa Lee was born in Busan, Korea and came to the United States for her college education. She earned her B.S. in computer science and mathematics from the University of Illinois at Urbana-Champaign in 1998. She can be contacted through:

Visualization Laboratory

C418 Langford Center

Texas A&M University

3137 TAMU

College Station, Texas 77843-3137.