11-16-2004

Synthesizing realistic spine motion using traditional rig controllers

Jabbar Raisani
Trinity University

Synthesizing Realistic Spine Motion Using Traditional Rig Controllers

Jabbar Raisani

A departmental senior thesis submitted to the

Department of Computer Science at Trinity University

In partial fulfillment of the requirements for graduation.

April 21, 2004

Thesis Advisor   Department Chair

Associate Vice President

For

Academic Affairs

Synthesizing Realistic Spine Motion Using Traditional Rig Controllers

Jabbar Raisani

Trinity University

Acknowledgements

First and foremost, I would like to thank Dr. John Howland for all of his assistance during the research and development phases of this thesis. I would also like to thank Dr. Maurice Eggen and Dr. Gerald Pitts for their additional assistance and time spent reviewing this thesis. Finally, I would like to thank Michael Donovan for giving me the opportunity to see my thesis used in an actual production environment.

Synthesizing Realistic Spine Motion Using Traditional Rig Controllers

Jabbar Raisani

Abstract:

This thesis presents a specific method for simulating realistic spine motion through the use of traditional rig control objects. The spine's motion and flexibility is based on a curve as opposed to the more traditional three bone spine. This allows for a more fluid animation more closely resembling that of a spinal column. However, this setup keeps the traditional three control rig setup in order to mimic the control system of a traditional three bone spine. Currently, three bone systems are well understood by animators and the motion is easy to control. The problem with this setup is that although easy to control, these rigs lack the fluid curving motion of a true spine.

The proposed method creates a solution to this problem by creating a curve that's length, shape, and twist is based on that of the three traditional bones. This solution allows traditional animators to quickly and effectively begin animating without having to alter the traditional control setup. This thesis describes how this animation pipeline can be set up using commercial software primarily focusing on Softimage XSI. Finally, a program was created using visual basic script that will allow other users to easily integrate the spine into future characters. The program is also based on the idea that the user is accustomed to the three bone spine set-up, and allows the user to create an advanced curve based spine rig by simply drawing the traditional three bone spine.

Contents:

List of Figures

Part I:

General Animation Background

1 Character Design:

The process for creating a realistic character is a long and arduous task. Before a character is even modeled in a graphics package a character must be designed. During the design process a distinct idea of the type of character you wish to create should be established. This process may simply involve a mental image of your character or an actual sketch of the character you wish to create. The body style of the character is important to determine during this phase because it will greatly affect the underlying skeleton and rig that will be required. Before the rigging can even begin, the character's proportions should be determined. When modeling a realistic character, an animator must be sure that the character's proportions are similar to those in reality. Although the animator can slightly alter the proportions, creating a character with proportions extremely different from that of reality will make creating realistic spine motion very difficult.

2 Character Modeling/Rigging:

Once the character has been modeled, based on the requirements stated by the design, a control skeleton and rig must be created. The skeleton developed should closely match that of the creature in reality. A character skeleton that does not mimic a real physical skeleton will make the animation process even more difficult. If the skeleton is drastically inaccurate it may be impossible to achieve realistic motion regardless of the amount time spent on animation.

Once an accurate skeleton has been created, a control rig must be developed to move the skeleton. A rig is basically a set of objects that do not render, which are used to control the skeleton that is attached to the character [8]. The rig should make the skeleton easier to manipulate than simply a skeleton on its own. Typically forward kinematics is best for arms and inverse kinematics should be used for the legs [1]. However, it is best to provide a method for switching between inverse and forward kinematics. Typically spines are created using a three bone forward kinematics system. However, as will be discussed later, a curve based skeleton can be used in order to achieve a more advanced spine motion. Although a rig is not absolutely necessary to move a skeleton, it can make the animation process much easier and faster.

3 Character Animation:

Following the creation of the rig, the animation process can begin. The best method for achieving realistic simulated motion is through the use of a motion capture system [2]. However, most systems are fairly expensive and not available to those who work on projects with small or no budget, as in the case of a student. Typically most animators in a low or no budget situation will turn to keyframe animation. Keyframing involves placing a character in one pose then moving in time and placing the character into another pose. The graphics software then interpolates the motion between the two poses. Regardless of whether the animation has been input through the use of motion capture or by an animator, the importance of the spine's initial setup is crucial.

If a spine is created using a traditional three bone setup, the accuracy of the animation will not be as close to reality as when a curved spine is used. This is because a spine is typically comprised of much more than three bones. For example the human spine contains thirty-three vertebrae, twelve thoracic vertebrae, five lumbar vertebrae, and five fused sacral vertebrae. All of these vertebrae protect the spinal cord, which travels throughout all of these bones. With the use of a three bone system the spinal chord is unable to follow a realistic path and unrealistic motion will most likely be the result.

Part II:

<u>Introduction to the Spine Based Rig:</u>

1 Introduction:

Creating realistic spine motion using computers continues to be one of the main challenges in computer graphics. One of the many challenges is that the character must comply with the rules of physics that exist in the world around us [6]. Even the actual laws of physics are not known. It is usually obvious when a character accidentally violates one of these laws. Spine motion is difficult to simulate because it is something that people are familiar with regardless of their knowledge of the laws of physics. Since birth, people observe how bodies move, balance, and react to changes in their surroundings. Humans are therefore very critical of simulated motion due to the great understanding of movement developed throughout our lives [5].

2 Research:

The solutions to the above problems with the three bone skeleton have been previously attempted [4]. The initial research and development involved looking into those solutions, which had already been attempted. After looking at the designs of several other curve-based skeleton rigs, the Isner Spine was selected as the best existing solution [3]. The Isner spine is the rig setup used in the default rig within Softimage XSI *(see **Figure 2.1**)*. This rig allows for very realistic spine movement through the use of a curve based spine. However, there are several problems with the existing Isner Spine [7].

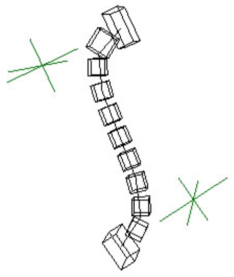

Figure 2.1: The Isner Spine

The first problem this rig presents is that it uses a non-conventional control setup. The rig has one hip controller and one chest controller. There are also two controllers that manipulate the overall shape of the curve. These manipulators determine which direction the curve bends in between the hip and chest controllers. There are several problems with this type of setup. The main problem is that it is difficult to determine exactly how the curve will react to the translation and rotation of the control objects *(see **Figure 2.2**)*. This is a significant issue because it is crucial that an animator understands exactly how the character will react to manipulations of the controllers.

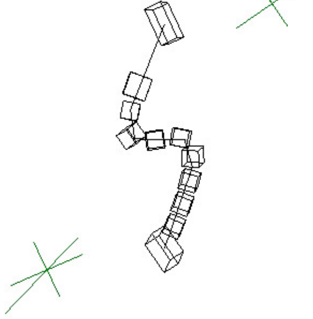

Figure 2.2: Undesired Rig Reaction

Another problem with the existing Isner rig is that the rig does not use a traditional control setup. Once again this rig uses two main controllers, a chest and a hip. There are also two secondary controllers to manipulate the shape of the curve *(see **Figure 2.3**)*. This setup is problematic because it is not similar to the existing rigs being used in production. The existing rigs use a three bone, three-controller setup. This setup is easy to use, and its motion is easily predictable and controllable. Since the Isner spine lacks this ease of control it becomes difficult for animators to use. It is essential for rigs to be easy to animate because projects often call for tight deadlines and efficient results. The additional learning times, as well as the general difficulty of use, make the Isner spine undesirable to many companies.

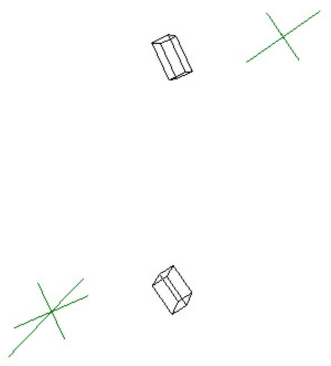

Figure 2.3: Non-Traditional Controllers

Yet another problem with the Isner spine is that it does not allow for exaggerated motion with a realistic curve based spine. When animating characters it is often necessary to exaggerate motions during the animation process. However, this type of motion often becomes impossible with the Isner spine. If the chest controller is rotated to extreme positions, such as an extreme backbend, it causes the upper body to flip or rotate anywhere from one-hundred-eighty to three-hundred-sixty degrees *(see **Figure 2.4**)*. This becomes a serious problem for animators because it does not allow for creative freedom to place a character into extreme positions.

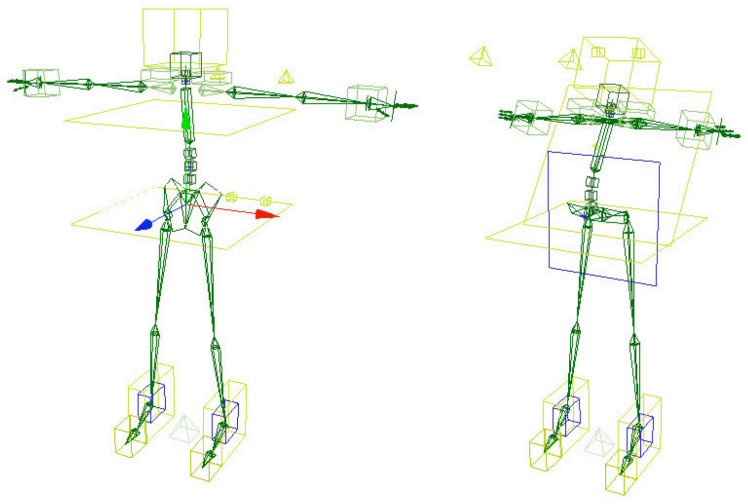
Figure 2.4: Isner Flip

The research concluded that although the Isner spine allows for realistic spine curvature, it is by no means the best solution to the spinal rig problem. The spine results in numerous problems thus preventing it from being the desired tool for most commercial production environments.

3 Solution Overview:

In order to allow the spinal chord to follow a realistic path, this thesis proposes that the spine follow the path of a defined curve as opposed to merely a three bone system. Initial research began by analyzing curve bone spine systems that already exist. The research was specifically focused the research of the pre-existing Softimage XSI default "Isner" rig because it had already been integrated into the pipelines of many smaller production studios and uses a curve based spine. The initial problem discovered with the default rig was that it does not use a traditional control setup that most animators are currently familiar with. In fact, the current control system is difficult for many animators to use. This is because the reaction of the rig to manipulations of the controllers can often be counterintuitive or simply unpredictable. The goal was to create a similar spine rig, while maintaining the traditional control objects for the animator. Thusly the setup would allow an animator to use traditional animation skills on a completely different type of underlying rig. Also, it was decided that the optimal set-up would allow for the spine to be integrated into the pipeline of others through the use of an automated spine creation process. This would allow other technical directors to create this advanced spine setup by simply specifying the location and shape of the spine desired. Thus alleviating others from having to go through the intense technical setup that this curve based spine setup requires.

Part III:

Final Set-Up Using XSI:

1 Spine Rig Creation:

First, the bones that will act as the underlying drivers for the spine motion should be drawn. This should be drawn in 2D, Y and Z space, so that it correctly aligns in with the rest of the skeletal rig *(see **Figure 3.1)***.

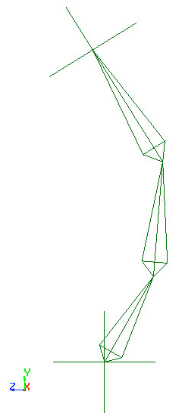

Figure 3.1: Traditional 3 Bone Spine

The next step in creating the spine rig is to draw the curve that will act as the spine. The curve should be drawn in 2D, Y and Z space. The curve should be constructed of five points *(see **Figure 3.2)***. The first point should be placed at the base of the spine, exactly on top of the root of the bottom bone. The next three points should be placed at the center of each of the three bones, thus following the curvature of the desired spine. The points are placed at the center of the bones to avoid the curve bending on the hinge of two bones in future steps. The final point should be placed at the effector of the top bone.

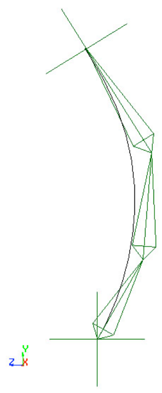

Figure 3.2: Spine Curve

Next, a chain based on the five-point spine curve was created *(see **Figure 3.3**)*. Within XSI, the create chain from curve function may be used to create this chain. This function will create a linear curve that will follow the original nurbs [9] (a type of spline) based curve, as well as the chain. Also, the "constrain chain to curve" option should be checked. This option forces the chain to follow the position of the curve at all times by rotating the bones in order to align as closely as possible to the curve. The amount of bones needed may vary from character to character. For a humanoid character of average height, eight to ten bones should be used.

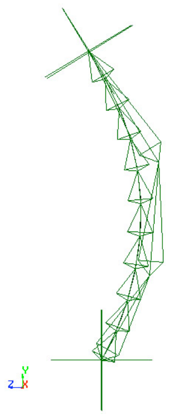

Figure 3.3: Spine Chain

The next step is to edit the linear curve that was created to have the same number of points as the initial nurbs-based curve. During the create chain from curve operation the number of bones, (NB), selected may exceed five. Since only five points exist along the nurbs curve, any additional points created due to the excess bones must be deleted from the linear curve. One point exists for each bone of the spine bones created during the create chain from curve step. Therefore, the equation for the number of points to delete is “(NB) – 5.”

Then, the remaining five points need to be placed in the same 3D space as the five points from the initial nurbs curve. Finally, the operator stack for the linear curve should be frozen in order to delete any dependencies upon the initial nurbs curve.

The linear curve is now ready to be enveloped to the three bone skeleton chain. Enveloping forces the curve to follow the curvature of the three bones as they are rotated. However, a problem results when the curve is enveloped to the three bone skeleton. The length of the curve slightly changes when the bones are rotated into extreme positions. This is a problem because the bones are not able to follow the curve if the chain is longer than the curve it is attempting to follow. In order to solve this problem, a compromise must be made. That is to adjust the length of the bones to collectively match that of the length of the curve. The next step is to create a set of ten nulls. These nulls should be path constrained to the initial nurbs curve *(see **Figure 3.4**)*. With all ten nulls selected, “L(0,100)” should be entered into the path percentage value. This forces all nulls to remain equidistant over the distance of the curve. This distance will dynamically change as the length of the curve changes due to manipulations of the original three bones.

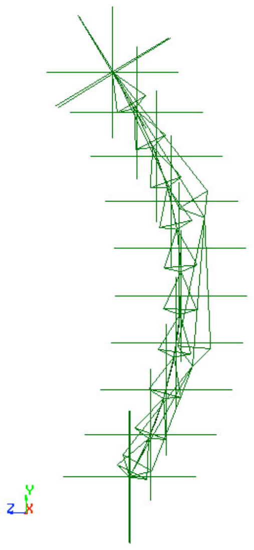

Figure 3.4: Length Nulls

Next, the length of the spine bones must be adjusted based on the length of the curve. To do this, expressions were created to control the length of the spine bones created by the curve to chain function. In order to know how long each bone should be, the length of the initial nurbs curve must be calculated. However, this resulted in another problem because XSI does not provide a function to calculate the length of a curve within an expression. Since the nulls are spread over the distance of the curve, this can be used to estimate the length of the curve. By multiplying the distance between the first two nulls (N1, N2) times the number of nulls (NN), and then dividing that distance by the number of bones (NB):

Ctr_Dist(N1, N2) * (NN - 1) / (NB)

the length of each bone can be calculated.

The next step in the process is to create the control objects that will be used by the animator to control the rig. Curve based squares were selected to serve as the control objects on this rig for two specific reasons. First, curves are non-rendering objects. Therefore, there will be no issues concerning undesired rendering objects during the rendering phase of the project. Secondly, it is fairly easy to tell what angle a square is rotated by analyzing the position of the four corners, as opposed to a circular control object. To aid in the visual representation of the angle of rotation, non-selectable curve based arrows have also been added to the squares. These arrows point forward when the control objects are oriented in the default position.

Curve based squares are primitive objects within XSI. Three curve based squares were created with the minimum allowed number of points in order to reduce clutter. These three squares will serve as the bottom, middle, and top spine controllers *(see **Figure 3.5**)*. The set-up of the spine bone controllers is an identical three step process. First, the position and rotation of the bottom controller must be matched to that of the bottom spine bone. Second, the rotation of the bottom bone needs to be constrained to that of the bottom bone controller. The position and orientation of the controller was matched to the bone first in order to avoid the bone changing position during the previous step. Third, a primitive null was created and its position and orientation were matched to that of the bottom spine controller. Then the null was made the parent of the bottom spine controller. By matching the position and orientation of the null to the controller and then making the null the parent, the local position and orientation of the controller were successfully changed to zero since the parent's global rotation and position are the same. These nulls are commonly referred to as "zero nulls." This step is useful because it allows an animator to easily reset the position or orientation of any axis of the controller by simply setting the desired values to zero. This three step process should be repeated on the middle and top controllers using the middle and top bones, respectively.

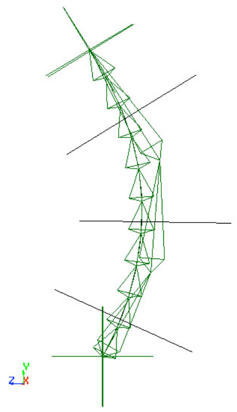

Figure 3.5: Controllers

The next step is to add control to the twisting motion of the spine. The twist of the spine should be based on the twist of the top bone controller. In order to achieve this, expressions must be placed on the roll of the bones created from the curve to chain function. As the top controller is rotated, those spine bones closer to the top of the chain should rotate more, and those towards the bottom should rotate less. This was actually fairly simple to set up since bones that are higher in the spine chain inherit the rotations of the bones underneath them. Therefore the X rotation of the top controller must be evenly distributed amongst the spine bones. Thusly the rotation (RX) must be divided by the number of bones (NB) in the spine:

$$(RX) / (NB)$$

Finally, the middle bone controller's zero null should be made a child of the bottom controller, and the top bone controller's zero null should be made a child of the middle controller. This will force the controllers to act similar to those of a traditional rig's bone controllers. For some final touches, the use should be kept from accessing certain aspects of the controllers that a user would not need to use. Specifically, the translation and scaling of all three control objects should be locked. This will prevent animators from accidentally manipulating undesired aspects of the control objects. The only aspect of the control objects that should able to be manipulated are the rotation values.

2 Rig Development: Problems and Solutions

During the development of the rig numerous problems were encountered. Some of these were foreseen, however numerous others were not. Initially, other curve based spine rigs were intensely examined in order to discover the abilities and problems that currently existed. Specific notes of these shortcomings were noted in an attempt to assure that the proposed rig would not follow any same pit-falls that might be discovered. The main focus of the spine pit-fall research fell upon the Isner-Spine.

The first and most extreme problem with the popular Isner spine is the tendency for the upper body to flip one hundred and eighty degrees when the chest controller is rotated to extreme positions. The proposed rig was extensively tested and its response to extreme rotations was found to be successful in that the rig did not rotate or flip *(see* ***Figure 3.6****)*.

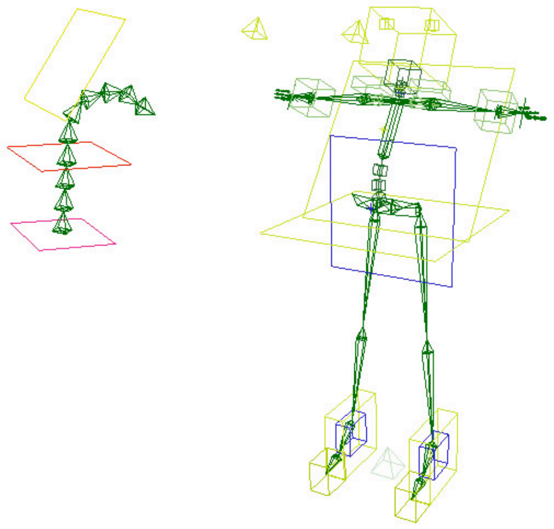

Figure 3.6: Isner Flip vs. Proposed Rig

However, one problem that was discovered during the testing phase was that the length of the nurbs curve changed during extreme rotations of the controllers. This resulted in the spine bones either kinking in an attempt to maintain the curve's shape, or simply extending past the curve *(see **Figure 3.7**)*. Although extending past the curve did not produce undesired deformations on the character, the kinking did. In order to avoid the kinking, research needed to be done in order to understand why the spine kinked. Essentially it was discovered that once a create chain from curve function was used, the length of the bones relative to the curve length was no longer analyzed.

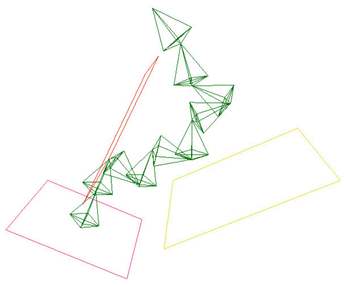

Figure 3.7: Spine Kinking

Therefore it was decided to dynamically update the length of each individual spine bone based on the length of the initial nurbs curve. However, this led us to another problem due to the fact that XSI does not allow for a curve length calculation through expressions. Several alternative solutions were then researched. One of which was to create a scripted operator that would update with each frame. Essentially, this is a program that would run each frame of playback and would use the curve length function to calculate the desired length of each bone. However, once the scripted operator was created to control the length of the bones several different problems arose. The biggest problem was that not all computers were able to update the length of the bones fast enough during playback. This resulted in different computers creating different animation although the same fcurves were being used as inputs. This was unacceptable behavior and it was therefore decided that a different method should be implemented.

The method chosen was to calculate the length of the curve only through functions that were available through expressions. The main function of focus was Ctr_Dist. The purpose of this function is to calculate the distance between the centers of two objects. Initially it was decided that the best method would be to constrain a null to each point on the curve and then calculate the distance between each consecutive null and then calculate the sum of these distances to find the length of the curve. However, this did not give a close enough estimate of the nurbs curve length. Instead, a series of ten nulls were constrained at equal distances apart along the curve. The distance between each of the nulls was then calculated. After testing, it was determined that this estimate gave acceptable results *(see **Figure 3.8**)*.

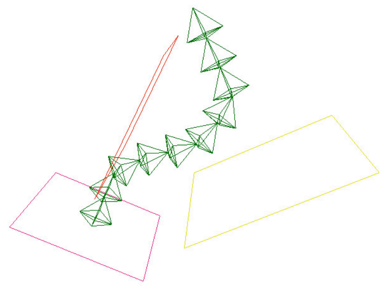

Figure 3.8: Spine Not Kinking

While attempting to reduce the number of function calls to Ctr_Dist it was discovered that each call to Ctr_Dist was in fact calculating the same value since the nulls were placed at equidistant intervals. Therefore, the number of nulls minus one multiplied by the Ctr_Dist between the first and second nulls, was instead used to calculate the length of the entire curve. Although the speed change due to a decrease in the number of calculations was not noticeable on one single character, this could potentially save a great deal of computation when dealing with scenes involving numerous characters or crowd simulation.

Another problem incurred during the development of the spine was the type of controllers to use for the rig. Initially, the idea was to use implicit objects to serve as control objects. However, this led to a couple of distinct problems. First, although implicit objects do not render, their names appear under the rendering objects list within pass setup. This could lead to confusion for artists attempting to set up render passes while using this rig. Secondly, implicit objects contain no points that can be actively manipulated by a user. This means that in order to change the size of a control object, an artist would need to change the actual scale of the implicit object. This could result numerous problems, mainly a change in the size of any objects that are children of the controller.

In order to avoid the problem of any control objects being listed in the rendering objects and to allow for easy resizing of the control objects, curves were used to create the control objects within the rig. Curves are the optimal choice because they are non-rendering, do not appear in the rendering objects list, and are easily re-scalable. In order to change the size of a curve based object, the animator must simply tag all of the points and then rescale the object. This does not result in a change in the scale of the center and thusly, will not change the size of the child objects.

Part IV:

Automation of Rig Set-Up:

1 Introduction:

After developing a solid spine rig set-up, it was decided that the optimal way to make the rig accessible to others in the industry was through automation of the rig creation process. Automation was chosen as opposed to a tutorial because automation of the process would not require the animator to know the underlying technicalities of the rig. There are several ways to do this within XSI, and the proposed automation program was implemented through the use of visual basic script (Appendix A). By writing the program using visual basic it was possible to use methods previously defined within XSI in order to aid in programming the automation. The easiest way for other artists, specifically technical directors, to integrate the spine into an existing pipeline was by making the automation closely resemble that of existing rig creation techniques. The process was specifically modeled after the method used to create the traditional three bone spine chain.

2 Step By Step:

First, the user must simply draw the traditional three bone spine setup and then run the program within XSI *(see Figure4.1)*.

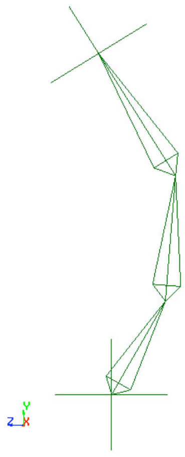

Figure 4.1: User Defined Spine

Next, the program asks the user to select the three bone setup and begins the automated process. The program initially analyzes the three bones that were drawn by the user and stores the translation and rotation of the bones in independent variables. The program locates the start and end points of each bone in 3D space.

The start and end point of the bones created by the user are then analyzed to locate the midpoint of each bone. This midpoint data, along with the location of the root of the initial bone drawn by the user, as well as the location of the effector of the last bone created by the user, are all stored into variables. The program then uses these variables along with the create nurbs curve function native to XSI in order to calculate the nurbs curve for the spine. Once the nurbs curve has been created, the program can convert this curve to a chain using XSI's curve to chain function.

Once the spine chain curve was created, the program then envelopes the linear, "curve to chain," curve to the three bones created by the user. Then ten nulls are created and constrained along the linear curve. These nulls are then spaced evenly along the curve. The expression for the length of each spine bone is then calculated using the Ctr_Dist function multiplied by the number of bones minus one, divided by the total number of spine bones.

Next, the program must create the control objects. The program uses the create curve function supplied by XSI. The program then matches the position and rotation of the bones using the previously stored variables from the user defined bones. After all control objects are in place, the three bones created by the user are then constrained to the control objects. The expression for the role of each bone is then set by the script to the Y rotation of the top bone controller divided by the number of bones. Finally, the program locks the translation and scaling of all of the control objects *(see **Figure 4.2**)*.

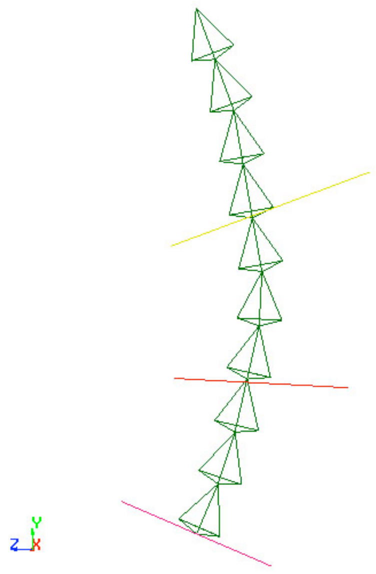

Figure 4.2: Final Spine Rig

Part V:

Conclusions and Future Work:

1 Conclusion:

In this paper an alternate method for creating a realistic spine for characters was presented. Although other curve-based spine rigs currently exist, this example is superior because it utilizes the traditional three control setup yet provides for more realistic spine motion. This setup allows for animators to easily make the transition from the old three bone skeleton to the more realistic spine curve skeleton because the advanced spine reacts similarly to manipulations in the traditional controllers. This will allow animators to easily begin using the advanced rig since they already know how to use the controls. This ease of transition can save companies time and money that would be necessary to teach the animators to use the other existing spine rigs. The most important development made for other companies was the automation of the rig creation process. Since the spine rig creation process has been automated the technical directors in charge of creating rigs are only required to know how to build a very rudimentary three bone chain spine. This will allow for artists with very little technical skill to create a custom curve based spine rig.

During the development of this spine creation process a visual effects firm in New York was in fact searching for the exact type of solution the spine problem. The curve-based spine previously discussed was offered to them as a solution, and they decided to test the effectiveness of the setup. After testing, they decided that this was the spine setup they wanted to use. The rigging and animation went smoothly during the project, and the commercial was a success. The fact that the rig held up, while being used in the commercial visual effects industry, displays that this is an effective solution to the curve based spine rig problem.

Future Work:

In the future, the goal is to eliminate the current problems that method shares with the other existing spine rigs. One of which is the tendency for the arms to inadvertently rotate a small number of degrees during extreme rotations of the chest controller. The research done was unable to successfully pinpoint the cause of this error. It is believed that the twisting is somehow related to the direction of the normal of the nurbs spine curve changing when the top control is manipulated. This could possibly be achieved by placing some additional constraints on the arms, or with some alternative hierarchical setup of the arms and chest. One alternative hierarchy could be making the arms a child of the top controller as opposed to being the child of the effector of the curve to chain bones.

The program created allows for this spine setup to be easily integrated into the existing pipelines of companies without requiring an extensive knowledge of the underlying technical aspects of the rig. However, more non-technical functionality would allow the user to have more options when using the automated spine rig creation program. One of the main options that could be included is to allow the user to select the hip and chest bones of their existing rig. The program would then automatically integrate the spine rig into the existing rig by applying the correct constraints and parenting. Another feature would allow the user to input names for the controllers as well as the bones that are created by the program. This would allow the user to use a specific naming convention as opposed to the names that were predetermined within the program.

Another change that could be made to the automated spine creation process is to have the spine rig's length and shape retain a dependency on the user-defined chain. Currently the curve based spine is independent of the user's chain as soon as the three user bones are substituted with those of the program. If the dependency remained, it would allow the user to modify the original bones defined in order to adjust the curve based spine. Currently, the user would need to modify the bones and then rerun the program, thus creating a new curve spine. By dynamically updating the spine curve based on changes to the original bones, a user would not be required to reintegrate the curve spine into the character rig.

The final change that could be made to the program would be a conversion from using visual basic to using C++. Both programming languages are supported by XSI. Visual basic was selected because it is possible to develop the code from within XSI. C++ however must be developed within a third party software such as Microsoft Visual Studio and then converted to a plug-in. The program would then have to be converted to a .DLL add-on and installed within XSI, which is yet another problem in itself. The coding itself is also much more difficult because it requires the use the XSI API as well as XSI's COM functions. Although the coding of the program would be much more difficult using C++, it would allow for the creation of classes, libraries, and structures.

Bibliography

[1] Funkhouser, Thomas. 2000. *Kinematics and Dynamics..*
http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/kinematics/sld001.htm

[2] Hodgins, Jessica K. 1996. *Simulating Human Motion*.
http://www.cc.gatech.edu/gvu/animation/Areas/humanMotion/humanMotion.html

[3] Isner, Michael, 2002. *Introducing the Isner Spine.*
http://www.isner.com/isnerspine/spine_introduction.htm

[4] Krotki, Andreas, 2003. *Human thoracic spine accurate bio-mechanical movements.*
http://www.maxforums.org/forum/forum.asp?t=221006&r=24

[5] Lin, John; Wu, Ying, Huang, TS. 2000. *Modeling the Constraints of Human Hand Motion.*
http://csdl.computer.org/comp/proceedings/humo/2000/0939/00/09390121abs.htm

[6] Liu, Karen; Popovic, Zoran. 2002. *Synthesis of Complex Dynamic Character Motion From Simple Animations*. http://www.cs.washington.edu/homes/karenliu/sig2002/paper.pdf

[7] Sale, Adam. 2002. *Constructing a Character Rig in XSI.*
http://www.joncrow.com/tutorials/xsi_tuts/Character_rig/character_rig.htm

[8] Swain, Mark. 2001. *Character Rigging, Poser.*
http://www.techtv.com/screensavers/answerstips/jump/0,24331,3358209,00.html

[9] Zorin, Denis. 2002. *Nurbs, Spline Surfaces and Blossoming in Two-Dimensions.*
http://www.techtv.com/screensavers/answerstips/jump/0,24331,3358209,00.html

Appendix A

Source Code

```
'****************************************************
'CurveSpine
'Version: 1.0
'Author: Jabbar Raisani
'Date: 3-5-04
'Purpose: Creates a Curve based spine from 3 bone chain
'Usage: Draw a 3 bone chain then run
'the program.  Select the root of the chain
'****************************************************

'Explicit in order to force variables to be predeclared

option explicit

main

'****************************************************
'main
'Purpose: call other methods
'****************************************************


function main

    dim oRoot

    DeselectAll
    oRoot = SelectChain

    'Determine if User Cancelled the Spine Creation
    if oRoot = "Cancelled" then
        exit function
    end if

    DrawChain oRoot

    SetUpChain

    CreateControllers
```

```
	SpineExp

	CleanUp

end function


'***************************************************
'SelectChain
'Purpose: Have user select chain root that will serve as
'the basis for the spine
'***************************************************

function SelectChain



	dim oRoot, oPressed

	'Have User Select the Chain Root
	PickElement "chain_element", "Select Root", "Select Root", oRoot, oPressed

	if oPressed = 0 then
		logmessage "Cancelled by User"
		oRoot = "Cancelled"
	end if

	'Returen Root
	SelectChain = oRoot

end function


'***************************************************
'DrawChain
'Purpose: Draws Spine Curve and Chain
'***************************************************

function DrawChain(oRoot)

	dim oP2x, oP2y, oP2z, oP3x, oP3y, oP3z, oP4x, oP4y, oP4z, oP5x, oP5y, oP5z
	dim oRx, oRy, oRz
	dim oMid1, oMid2, oMid3


	'Get X,Y,Z point values for all 5 Spine Curve Points
```

```
oRx = GetValue(oRoot&".kine.local.posx")
oRy = GetValue(oRoot&".kine.local.posy")
oRz = GetValue(oRoot&".kine.local.posz")

GetPrim "Null", "Mid1"
ApplyCns "TwoPoints", "Mid1", "bone,bone1"

oP2x = GetValue("Mid1.kine.local.posx")
oP2y = GetValue("Mid1.kine.local.posy")
oP2z = GetValue("Mid1.kine.local.posz")

GetPrim "Null", "Mid2"
ApplyCns "TwoPoints", "Mid2", "bone1,bone2"

oP3x = GetValue("Mid2.kine.local.posx")
oP3y = GetValue("Mid2.kine.local.posy")
oP3z = GetValue("Mid2.kine.local.posz")

GetPrim "Null", "Mid3"
ApplyCns "TwoPoints", "Mid3", "bone2,eff"

oP4x = GetValue("Mid3.kine.local.posx")
oP4y = GetValue("Mid3.kine.local.posy")
oP4z = GetValue("Mid3.kine.local.posz")

GetPrim "Null", "Mid4"
ApplyCns "Position", "Mid4", "eff"

oP5x = GetValue("Mid4.kine.local.posx")
oP5y = GetValue("Mid4.kine.local.posy")
oP5z = GetValue("Mid4.kine.local.posz")



'Create a cubic interpolating curve called SpineCurve
SICreateCurve "SpineCurve", 3, 0
SIAddPointOnCurveAtEnd "SpineCurve", oRx, oRy, oRz, False
SIAddPointOnCurveAtEnd "SpineCurve", oP2x, oP2y, oP2z, False
SIAddPointOnCurveAtEnd "SpineCurve", oP3x, oP3y, oP3z, False
SIAddPointOnCurveAtEnd "SpineCurve", oP4x, oP4y, oP4z, False
SIAddPointOnCurveAtEnd "SpineCurve", oP5x, oP5y, oP5z, False

'Create Chain From Spine

FreezeObj "SpineCurve"
```

```
	CreateChainfromCurve "SpineCurve", 10, True
	FreezeObj "SpineCurve_crv"
	DeleteObj "SpineCurve"

	'Match new curve to old curve

	ApplyTopoOp "NurbsCrvDeletePoint", "SpineCurve_crv.pnt[2-LAST]",
siUnspecified, siPersistentOperation

	SIAddPointOnCurveAtEnd "SpineCurve_crv", oP2x, oP2y, oP2z, False
	SIAddPointOnCurveAtEnd "SpineCurve_crv", oP3x, oP3y, oP3z, False
	SIAddPointOnCurveAtEnd "SpineCurve_crv", oP4x, oP4y, oP4z, False
	SIAddPointOnCurveAtEnd "SpineCurve_crv", oP5x, oP5y, oP5z, False

	ApplyTopoOp "NurbsCrvDeletePoint", "SpineCurve_crv.pnt[1]", siUnspecified,
siPersistentOperation

end function


'****************************************************
'SetUpChain
'Purpose: Envelopes New Curve and sets up Expressions
'****************************************************

function SetUpChain

	'freeze curve stack and rename curve

	FreezeObj "SpineCurve_crv"
	SetValue "SpineCurve_crv.Name", "SpineCurve"

	'envelope curve to original chain

	ApplyFlexEnv "SpineCurve;bone,bone1,bone2,eff", False

	'Create Nulls to be used for Length Calculations

	GetPrim "Null", "Len1"
	GetPrim "Null", "Len2"
	GetPrim "Null", "Len3"
	GetPrim "Null", "Len4"
	GetPrim "Null", "Len5"
	GetPrim "Null", "Len6"
	GetPrim "Null", "Len7"
	GetPrim "Null", "Len8"
	GetPrim "Null", "Len9"
	GetPrim "Null", "Len10"


	'Parent Length Nulls under same Parent

	GetPrim "Null", "Len_Par"

	CopyPaste "Len1", , "Len_Par", 1
	CopyPaste "Len2", , "Len_Par", 1
	CopyPaste "Len3", , "Len_Par", 1
	CopyPaste "Len4", , "Len_Par", 1
	CopyPaste "Len5", , "Len_Par", 1
	CopyPaste "Len6", , "Len_Par", 1
	CopyPaste "Len7", , "Len_Par", 1
	CopyPaste "Len8", , "Len_Par", 1
	CopyPaste "Len9", , "Len_Par", 1
	CopyPaste "Len10", , "Len_Par", 1


	'Equally Distribute Length Nulls about curve

	ApplyCns "Path", "Len1,Len2,Len3,Len4,Len5,Len6,Len7,Len8,Len9,Len10",
"SpineCurve"
	SetValue
"Len1.kine.pathcns.perc,Len2.kine.pathcns.perc,Len3.kine.pathcns.perc,Len4.kine.pathc
ns.perc,Len5.kine.pathcns.perc,Len6.kine.pathcns.perc,Len7.kine.pathcns.perc,Len8.kine
.pathcns.perc,Len9.kine.pathcns.perc,Len10.kine.pathcns.perc", Array(0, 11.111, 22.222,
33.333, 44.444, 55.556, 66.667, 77.778, 88.889, 100)



end function


'*****************************************************
'CreateControllers
'Purpose: Creates Control Objects For Spine
'*****************************************************

function CreateControllers

'CREATE LOWER CONTROLLER

CreatePrim "Square", "NurbsCurve", "Lower_Ctrl"
SetValue "Lower_Ctrl.crvlist.geom.subdivu", 3
SetValue "Lower_Ctrl.square.length", GetValue("bone.bone.length")

'Match Controller to bone
ApplyCns "Position", "Lower_Ctrl", "bone"
ApplyCns "Orientation", "Lower_Ctrl", "bone"

SelectGeometryComponents "Lower_Ctrl.pnt[*]"
Rotate , 0, 90, 0, siRelative, siAdd, siObj, siXYZ

'get zero null
GetPrim "Null", "zero_Lwr"
MatchTransform "zero_Lwr", "Lower_Ctrl", siSRT

'Make bone follow ctrl
RemoveAllCns "Lower_Ctrl"
ApplyCns "Orientation", "bone", "Lower_Ctrl"

'zero out ctrl
CopyPaste "Lower_Ctrl", , "zero_Lwr", 1


'CREATE MIDDLE CONTROLLER

CreatePrim "Square", "NurbsCurve", "Middle_Ctrl"
SetValue "Middle_Ctrl.crvlist.geom.subdivu", 3
SetValue "Middle_Ctrl.square.length", GetValue("bone1.bone.length")

'Match Controller to bone
ApplyCns "Position", "Middle_Ctrl", "bone1"
ApplyCns "Orientation", "Middle_Ctrl", "bone1"

SelectGeometryComponents "Middle_Ctrl.pnt[*]"
Rotate , 0, 90, 0, siRelative, siAdd, siObj, siXYZ

'get zero null
GetPrim "Null", "zero_Mid"
MatchTransform "zero_Mid", "Middle_Ctrl", siSRT

'Make bone follow ctrl
RemoveAllCns "Middle_Ctrl"
ApplyCns "Orientation", "bone1", "Middle_Ctrl"

'zero out ctrl
CopyPaste "Middle_Ctrl", , "zero_Mid", 1
```

```
'CREATE TOP CONTROLLER

CreatePrim "Square", "NurbsCurve", "Top_Ctrl"
SetValue "Top_Ctrl.crvlist.geom.subdivu", 3
SetValue "Top_Ctrl.square.length", GetValue("bone2.bone.length")

'Match Controller to bone
ApplyCns "Position", "Top_Ctrl", "bone2"
ApplyCns "Orientation", "Top_Ctrl", "bone2"

SelectGeometryComponents "Top_Ctrl.pnt[*]"
Rotate , 0, 90, 0, siRelative, siAdd, siObj, siXYZ

'get zero null
GetPrim "Null", "zero_Top"
MatchTransform "zero_Top", "Top_Ctrl", siSRT

'Make bone follow ctrl
RemoveAllCns "Top_Ctrl"
ApplyCns "Orientation", "bone2", "Top_Ctrl"

'zero out ctrl
CopyPaste "Top_Ctrl", , "zero_Top", 1


end function


'*****************************************************
'SpineExp
'Purpose: Setup Expressions For Spine Bones
'*****************************************************

function SpineExp

    'setup roll expresstions

    SetExpr "bone3.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone4.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone5.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone6.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone7.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone8.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone9.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone10.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
    SetExpr "bone11.joint.roll", "Top_Ctrl.kine.local.rotx / 10"
```

```
	SetExpr "bone12.joint.roll", "Top_Ctrl.kine.local.rotx / 10"

	'Setup Length Expressions

	SetExpr "bone3.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone4.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone5.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone6.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone7.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone8.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone9.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone10.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone11.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"
	SetExpr "bone12.bone.length", "ctr_dist( Len1. , Len2. ) * 9 / 10"

end function

'*****************************************************
'CleanUp
'Purpose: Finishing Touches, Deleting Objects, Parenting
'*****************************************************


function CleanUp

	'Create Group and hide technical spine elements

	DeleteObj "Mid1,Mid2,Mid3,Mid4"
	CreateGroup "Hidden", "Len_Par,
Len1,Len2,Len3,Len4,Len5,Len6,Len7,Len8,Len9,Len10, SpineCurve, zero_Lwr,
zero_Mid, zero_top, bone,bone1,bone2,root,eff,root1,eff1"
	SetValue "Hidden.viewvis", 0
	SetValue "Hidden.rendvis", 0

	'Setup Parenting for Controls

	CopyPaste "zero_Mid", , "Lower_Ctrl", 1
	CopyPaste "zero_Top", , "Middle_Ctrl", 1

	'Create Transform Setups for additive rotation

	AddProp "Transform Setup", "Lower_Ctrl, Middle_Ctrl, Top_Ctrl"
	SetValue "Lower_Ctrl.transformsetup.tool", 3
	SetValue "Middle_Ctrl.transformsetup.tool", 3
	SetValue "Top_Ctrl.transformsetup.tool", 3
```

```
    'Change Color of Controllers

    MakeLocal "Lower_Ctrl.display", siNodePropagation
    MakeLocal "Middle_Ctrl.display", siNodePropagation
    MakeLocal "Top_Ctrl.display", siNodePropagation

    SetValue "Lower_Ctrl.display.wirecol", 527
    SetValue "Middle_Ctrl.display.wirecol", 15
    SetValue "Top_Ctrl.display.wirecol", 126

end function
```