ABSTRACT

An Automated System for the Creation of Articulated Mechanical Parts.

(December 2009)

Christopher Ryan Wheeler, B.E.D., Texas A&M University

Co-Chairs of Advisory Committee: Prof. Tim McLaughlin
Prof. Philip Galanter

Proposes a new method to model the geometric form of articulated mechanical parts while simultaneously testing their range of motion in relation to other nearby parts. Utilizing a database of mechanical parts in virtual three-dimensional form, a software tool assists users in quickly building a complex high-level mechanical object which can be placed directly into a visual effects production pipeline. The tool creates a workflow that allows modeling and rigging problems to be solved concurrently within the same interface. Optimized animation controls are generated automatically to expedite the rigging process. A system of standardization provides a framework for each part's functionality within the hierarchy of each new assembly, while also guaranteeing re-usability and backwards compatibility with all other assemblies created with this tool. A prototype has been developed as a plug-in to existing commercial software to showcase the described methodology. This prototype provides a unique solution to common modeling and rigging problems in the field of visual effects and animation.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The visual effects industry is dependent on robust software tools to complete a wide range of tasks. The capacity of tools to reduce redundancy and increase productivity is a key factor in the success of companies in highly competitive markets. Computer graphics models are commonly broken into two categories based on the type of deformations applied to the surface geometry. The first of these, weighted deformations, use a weighted average at the vertex level to determine the final geometric form. Weighted deformations are commonly used for organic models such as people and animals, which have a single exterior surface made of skin. The second type, rigid deformations, are the focus of this thesis. Rigid deformations use simple matrix transformations to describe the translations, rotations, and scales applied to an object in either local or world space. Rigidly bound geometry is grouped hierarchically using parent-child relationships, and is typically found in mechanical assemblies such as robots, vehicles, and buildings. The process of describing how parts are hierarchically connected to one another and how they are controlled by the animator is called rigging. Rigging and modeling are usually handled in separate parts of the pipeline by two or more people and must be carefully coordinated to ensure that the geometry and rig are capable of handling the performance requirements of the final animation. This

---

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

separation of duties introduces inefficiencies and creates the potential for error when the initial process, modeling, fails to account for the range of motion that rigging must accommodate. In addition, the complexity of mechanical assemblies can quickly become overwhelming when hundreds of parts must work in unison, regardless of the workflow being utilized. Furthermore, if no system of standardization exists to control the connections between parts, changing the configuration of an assembly from its original design may introduce unnecessary complications, and increase the amount of time and money spent on a project. In order to alleviate these problems, this thesis introduces a new paradigm to bring modeling and rigging together in a unified, flexible environment that can be integrated with existing visual effects and animation workflows.

I.1. Significance

The visual effects and animation industries are continuously pushing the envelope to produce films that will captivate their audiences. This requires taking steps forward in complexity and scale, while simultaneously delivering these feats on shorter time frames with more limited budgets. Competition is fierce in this ever-expanding marketplace and any competitive advantage must be fully utilized for these companies to maintain profitability. Mechanical objects and characters continue to play an increasing role in movies and thus require their own unique solutions to integrate successfully with an already strained production pipeline. Existing commercial techniques rely on a heavily customized approach that requires all models and rigs to be built in a sequential process every time they are needed. Also, because of the likelihood of continuous

revisions throughout the design process, it is necessary to find an optimal method of combining rigid parts in a way that creates an accelerated, flexible workflow. In order to demonstrate this workflow, a tool has been created to extend the functionality of a common 3D software package to include a system dedicated to the simultaneous modeling and rigging of articulated mechanical objects.

CHAPTER II

PROBLEM

Creating articulated mechanical parts in a 3D environment can be a challenge, especially when all the parts must connect in a believable way while maintaining the ranges of motion needed for proper animation. This problem is compounded by the lack of re-usability of previously created geometry when no standards exist for the way they were modeled. Even after the models have been created, there is still the issue of articulation. Riggers must set up the animation controls for every character in the production and ensure that each part of a model is able to move in way that accommodates its performance requirements. When a model has parts that must collapse or fold together in a particular way, geometry intersections are a common problem to be addressed. Finally, highly complex models require the coordination of many modelers and riggers to finish the job. The rigging process cannot start in earnest until all geometry has been created, wasting valuable time in the initial stages of the pipeline. My proposed approach has alleviated these problems through the following techniques:

1. A database of parts is imported into the scene using file naming conventions to classify each part by identifier, type, and function.
2. A system of standardization governs the connections between parts. All parts must adhere to a set of rules to avoid possible conflicts within the object hierarchy.

3. A three-dimensional interface presents the parts and actions to the user directly within the viewport. The interface is simple and concise yet flexible enough to create models of considerable complexity.
4. An automated rigging system has been implemented. The rigging controls facilitate the process of combining the parts together and allow articulation to be explored to check for part conflicts within the range of motion.
5. Two complementary modeling workflows are supported:
    a. Modeling can be done on-the-fly within the active hierarchy to correct interpenetrations as they are identified. Each part is wrapped in a parent group to maintain its local coordinate system to facilitate quick and accurate vertex modifications.
    b. Modeling can also be carried out in separate files by any number of modelers. A swap function allows primitive building blocks to be used as placeholders while the modeling process is completed elsewhere. This allows articulation to be solved earlier in the pipeline, letting the final models be interchanged as they are completed. The standardized part descriptions reduce the chance of error and guarantee interoperability.
6. A working prototype has been successfully completed and demonstrated.

CHAPTER III

PRIOR WORK

This thesis builds on previous work in several fields including geometric modeling, database-driven component building, and shape grammars.

III.1. Geometric Modeling

Many tools exist that do a fantastic job of creating and editing geometric forms [1]. These models can be arranged and articulated in this software as well, but the process is usually complicated to set up correctly and the relationship between finished pieces is not clearly defined except within the mind of the artist who is creating the models. A program designed by Smith et al. [2] offers a solution to the problem of defining a relationship between the different pieces of geometry in complex structures. Using pin joints as a means of constraining and positioning parts, the resulting structures can be dynamically altered to optimize the geometry and mass as needed, maintaining correct physical properties in relation to the system as a whole. While this method is sufficient for engineering structures such as bridges, it does not provide a generalized framework that can easily be extended for work in other domains such as robotics. [D]eOung et al. [3] offer another approach where geometric constraints and model features are used to allow generalized model recombination using “Frontier”, a geometric constraint engine aimed at model assembly. This system successfully fit existing parts together to create unique models, but did nothing to address the way those

parts might move in relation to one another. Da Silviera et al. [4] have done interesting work regarding the modeling of complex large-scale environments. Lessons can be learned from their multi-level approach used to design and manage large sets of architectural models for use in virtual settings. However, being strictly architecturally focused limits the usefulness of their tool in other domains, especially in regards to articulation. Bush et al. [5] created a procedural approach to building geometry that used other objects as obstacles to avoid. This allowed each new part to fit into the system automatically while avoiding interpenetrations. While this may be a desirable initial condition, an ideal tool would help avoid interpenetrations brought about by the movement of the parts as well. Another procedural modeling tool was created by Morkel et al. [6] to control and vary models, add varying levels of detail, increase model complexity, and add base shape independence. These are great modeling-specific features but do little to address the needs of a properly rigged articulated assembly.

III.2. Data-driven Synthesis

Database-enhanced modeling techniques have been used with some success recently. Funkhouser et al. [7] have developed a system to take pieces of models from a library of geometry and combine them with pieces from other models using seaming and stitching techniques. An example given is one where the head of a cow is transferred onto the body of a dog. The automatic identification of similar part types offers a significant advantage over manual classification systems, and a scalable implementation of this feature would be necessary for wide-spread adoption of any truly universal

modeling tool. Corney et al. [8] have created a process in which rough stand-in models are used as a coarse filter to refine the selection of appropriate models from a database. This idea has been repurposed in this thesis to form a workflow in which placeholder geometry is used to solve basic articulation relationships while the final geometry is being completed.

III.3. Shape Grammars

Shape grammars can be defined as a means of describing form in a manner that can be communicated through a set of simple rules. Describing objects this way is of particular interest to the realm of architecture where facades can be broken into components that can be described with simple mathematical models. Using a progressive refinement strategy coupled with context sensitive shape rules, Muller et al. [9] added features to initially simple models to iteratively increase complexity until some terminal condition is met. A similar mode of operation is used in this thesis to progressively add parts down the hierarchy of an assembly until the user is satisfied with the result. Berndt et al. [10] have implemented a modeling language that takes parameterized primitives and combines them using a set of commands. These instructions form the building blocks for more complex shapes which are then converted to polygonal meshes. Another use of shape grammars by Birch et al. [11] was to create a system to rapidly build a range of historical architectural styles using a simple interface. The common features of each genre, such as roof type, window distribution, and exterior finishes, were used as the starting point for the building model. Similarly, this thesis

uses common mechanical part types and combines them in a logical way. A range of styles can be achieved by loading different sets of parts that adhere to a particular theme.

CHAPTER IV

METHODOLOGY

Because standardization is a key theme in this work, real-world mechanical objects were studied to determine commonalities and recurring themes in the means of articulation. After distilling these components into simple categories, five part types were discovered (see Fig. 1) that can be configured to provide a wide range of mechanical functionality. The parts are as follows:

1. Hinge joint: Allow rotation in a single axis perpendicular to the normal of the parent's surface (local Z axis.) A complete Hinge joint consists of a single HingeA and a corresponding HingeB.
2. Ball joint: Allow rotation in all three axes. A complete Ball joint consists of a single BallA and a corresponding BallB.
3. Screw joints: Allow rotation in a single axis parallel to the normal of the parent's surface (local X axis.) A complete Screw joint consists of a single ScrewA and a corresponding ScrewB.
4. Piston: Connect two rods on either side of a joint. A complete Piston consists of a pair of Hinge or Ball joints on opposing faces labeled PistonA1 and PistonA2, and the piston object itself consisting of a single PistonB1 and corresponding PistonB2. Pistons self-orient automatically and don't have manual controls.

5. Rod: Any intermediate geometry between joints or stand-alone geometry acting as an accessory. Rods are a single piece of geometry.

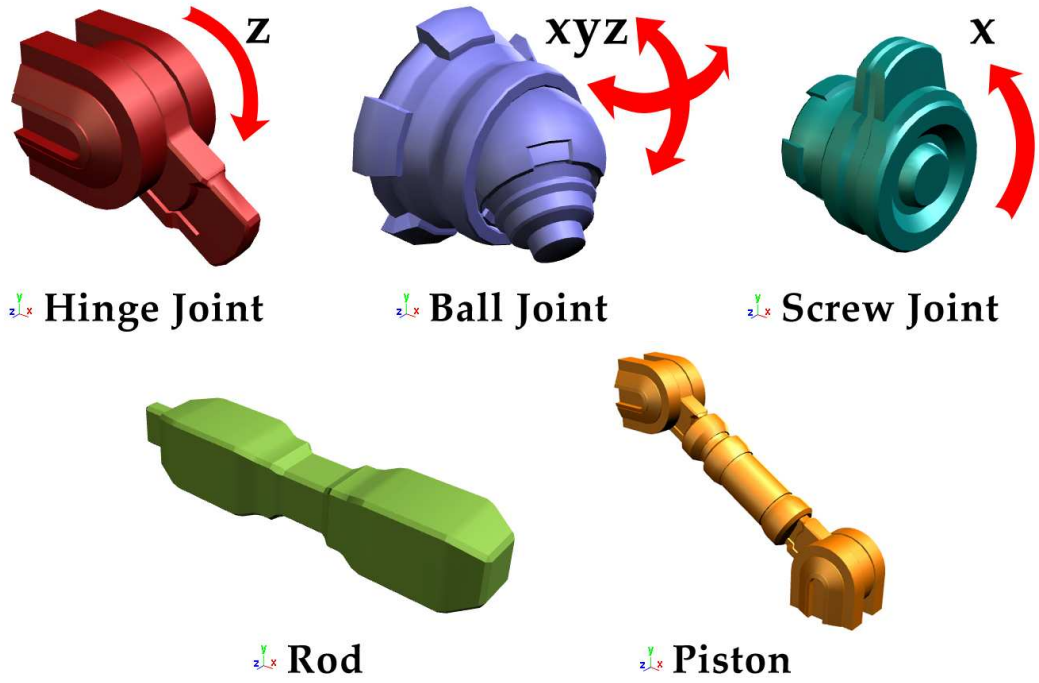

Fig. 1. The Five Part Types.

An initial set of these parts exist as separate part files to assist the novice user in learning the system. The part files follow a specific set of rules to guarantee interoperability. Each part file contains the geometry needed for that part plus two null objects that determine the beginning and end locations for insertion into the final assembly. The name of the part matches the file name, all transformations are set to zero, and the first null object is located at the world origin. These rules streamline the

process of integrating more parts with the system and make it possible for independent geometry builders to integrate their work efficiently.

IV.1. User Interface

In order to display the pre-built parts to the user in an easy to understand interface, a novel technique is used within the viewport of the 3D software. Upon loading the tool from a freshly opened file, a quick search is done within the parts directory to query the currently available parts. As new parts are added to the part folder, they are automatically added to the interface the next time the tool is loaded. Once the tool is loaded, the parts are displayed as three-dimensional objects in a series of three arcs in the top-left corner of the viewport (See Fig. 2). The outermost arc contains the editing tools and interface modification buttons. These functions include Copy, Mirror, Pin, and Swap. The Copy command duplicates a part and all its associated children. Mirror creates a copy and flips the entire assembly along a specified axis. Pin updates the transformation matrices of a part that has been moved from its initial insertion point. Swap removes the current part and replaces it with another. The arrow in the corner allows the user to minimize the interface to maximize screen space. Working inward, the next arc is the navigation section and it toggles between the different types of parts available to the user. For Hinges, Balls, Screws, and Pistons, the third arc is reserved for type A parts and the fourth arc for type B. Since Rods do not have A and B components, the third and fourth arcs just allow more Rods to be displayed. When a navigation button is pressed, a short animation creates a seamless

transition between sets of parts to inform the user of this change. A small scaling animation identifies a part insertion operation.

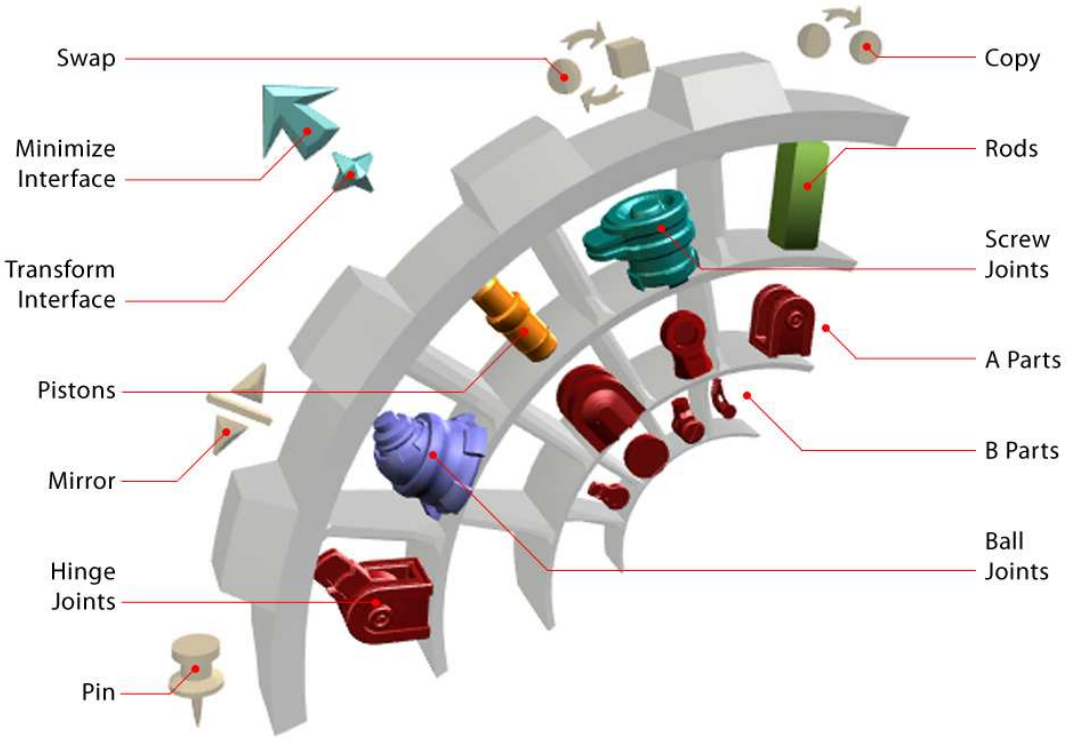

Fig. 2. The Interface.

IV.2. Using the Tool

As parts are inserted into the scene, the tool identifies the intent of the user based on the current selection. If no objects are selected, the new part is placed at the world origin facing down the x axis. If one object is selected, the new part is inserted at the location of the selected object's default attachment point, and the new part's X axis is aligned with the selected object's X axis. If an object's face is selected, the new part is

inserted at the center of the face, pointing in the direction of that face's normal (see Fig. 3).

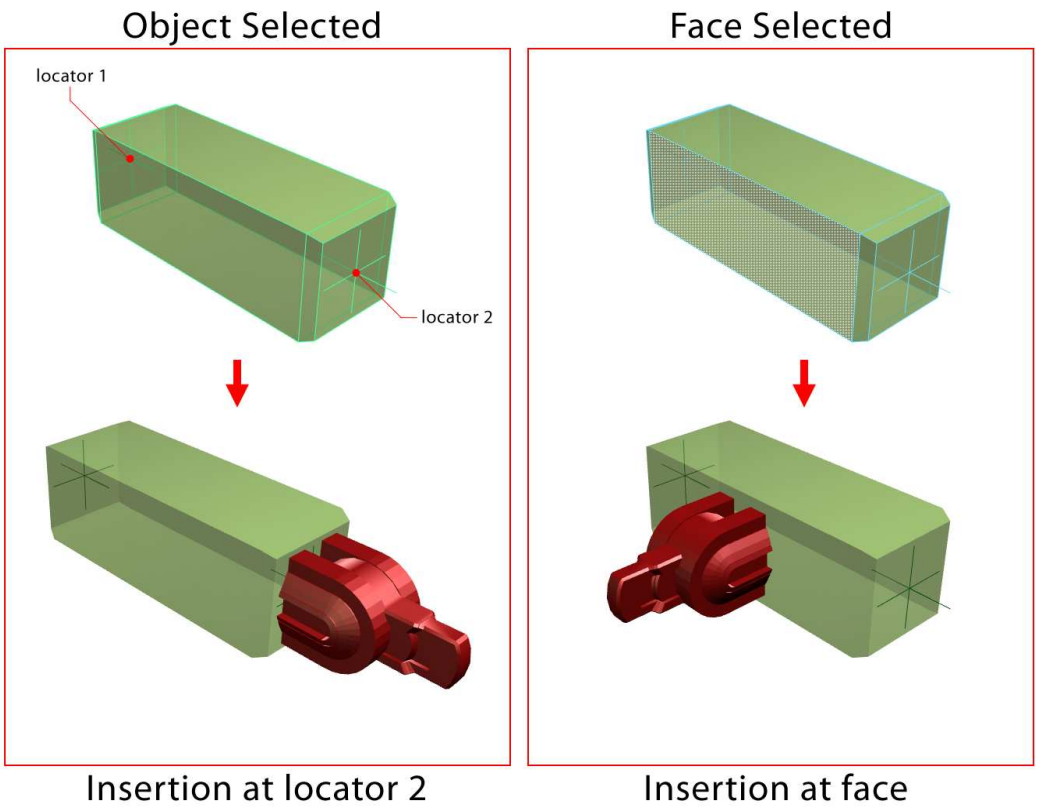

Fig. 3. Inserting New Parts. The two methods of inserting a new part onto an existing part.

Once a part has been inserted, a controller is also imported and bound to the new part. Each part type has a unique controller to help identify its function. For instance, a HingeA part gets a controller that looks like a set of bolts to emphasize that the connection to its parent is rigid. Likewise, HingeB parts get a circular, gear-like control to clarify that only one-dimensional rotations are allowed (see Fig. 4).

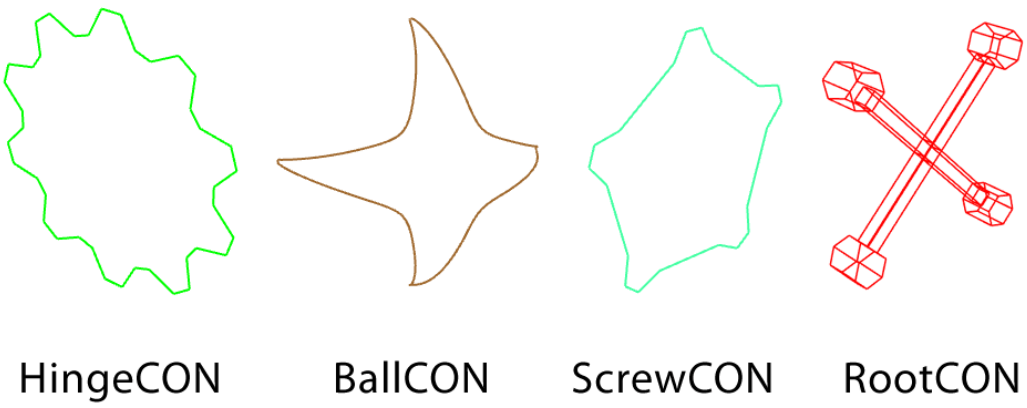

Fig. 4. The Four Types of Controls.

The primary function of the controls is to allow the user to manipulate the imported parts through translations, rotations, and scales without affecting the local coordinate system of the part. A unique attribute of the controller is that it will maintain its relative position to its parent as the parent is scaled without inheriting the scale factor itself. This gives the effect of the part being “pinned” to the surface of its parent and allows for each part to have independent transformation matrices. This is where the need for the Pin command becomes apparent. If a part is moved from its initial insertion point to another location on its parent, the child will appear to be pinned to its old location unless the Pin command is executed, resetting its local transformation matrices to zero at the new location. Under this paradigm, scaling a control object along its local X axis changes the distance between the two insertion points (null objects) of each part.

A typical session with the tool might go something like this. The user runs the initialization script in a newly opened file to import the interface and activate the

commands. A Rod is inserted at the origin to act as the highest parent in the hierarchy (most likely the hip geometry for an anthropomorphic robot). A face is selected where the shoulder joint would be and a Ball joint A and B would be imported here. Since the last object to be imported is left selected, a single click will insert a Rod at the default insertion point of BallB. Another two clicks and a HingeA and B are placed at the end of the Rod. Another Rod and then another BallA and B finished by one last Rod completes the arm assembly. The same sequence of steps could be used to create a leg assembly or the Copy command can be used as a shortcut. By scaling the controls, the length, width, and depth of each part can be worked out until the desired condition is met. At this point, the Mirror command can be used to create mirrored copies on the opposing side of the first Rod inserted. This completes the four appendages needed for a biped or quadruped robot and the last part, the head, can be inserted as a Rod at the front of the first Rod. Additionally, pistons may be inserted between any two faces as desired. The joints can be run through their degrees of freedom to locate interpenetrations and guarantee that the entire assembly meets the animation specifications. The final phase of the process is to swap out each part for its corresponding finished, high resolution geometry if needed. The use of proper naming conventions is critical to the successful application of the swapping operation. An external text file holds the swappable name pairs to allow ease of access by all team members and to facilitate updates to the file outside of the 3D software. As the swapped parts are imported, they acquire the length (distance between null objects) of the placeholder part but not its width or height to avoid skewing the high resolution geometry. Once the Swap operation has taken place,

additional translation, rotation, or scaling may be necessary to finesse any differences between placeholder and final parts. Each joint can be taken through its range of motion to check for part interpenetrations which can then be fixed at the object or vertex level using standard modeling techniques.

Because of the explicit generality in the types of parts used, it should be obvious that one could think of a situation where the tool might fail to create a level of complexity suitable for some models. The process of rigging usually requires a highly customized approach because of the unique features found in a group of characters and these need to be addressed on a case by case basis. The focus of this tool is to be lightweight and intuitive enough to accelerate the initial character setup process without burdening the rigger with unwanted complexity.

In the exploration of this thesis, a technique using joints (or bones in some software packages) eventually proved fruitless. The assumption was that inverse kinematics (IK) would be beneficial to the process of modeling and rigging a mechanical object, and joints are necessary for the application of IK. The benefit of IK is mainly apparent during the animation phase of the pipeline and the additional overhead of keeping a matching joint for each part created unnecessary complexity in all aspects of the tool. A simple workaround is to only generate joints as needed for the special case of IK to keep the files as clean and readable as possible.

CHAPTER V

IMPLEMENTATION

The tool was created as a plug-in to Autodesk Maya 2008 using Python as the primary scripting language. Maya is used frequently in the visual effects industry and provides all of the necessary functions required for modeling and rigging complex characters and mechanical assemblies. The geometric models created for this thesis were composed of polygons, although NURBS or SubD surfaces could be used just as easily. Maya also provides a way to link objects together non-hierarchically using constraint nodes, which bypasses some of the standard matrix inheritance issues associated with simple parent-child relationships.

To begin the process of testing the tool, many geometric parts have been modeled in a format compatible with Maya. The geometric structures are composed of vertices, edges and polygons, and techniques such as extrusions and bevels were used to add detail. Each part resides in its own unique Maya file (.ma) and contains only the geometry of the part along with two locator objects named locator1 and locator2. Locator1 sits at the world origin of the file and represents the location this part will "snap" to when inserted. Locator2 provides the default position to which future children parts may attach. A naming convention is used to determine the function, identity, and type of the part using this format: FunctionIdentityTypeIndex. The Function prefix can be either HingeA, HingeB, BallA, BallB, ScrewA, ScrewB, PistonA, PistonB, or Rod. The Identity part of the name is unique for each part and becomes crucial during the

swap phase of the process. Type can either be CON or GEO, representing the terms controller or geometry, respectively. Index is used to differentiate separate instances of the same part in the same file. A typical example of this naming convention in use might be HingeAmetalGEO12. Both the geometry and the file share the same Function and Identity prefix (in this case HingAmetal.ma) to assist the process of importing.

V.1. User Interface

To create the interface, additional files were created for each component to allow updates or modifications individually without affecting the rest. The interface is unique in that it is composed completely of three dimensional geometry and the parts displayed as buttons are the actual parts that will be imported when pressed. As the interface is built, the folder containing the Maya files is scanned to include the most recently updated parts available. Because the interface is made from polygonal surfaces, a solution needed to be found to avoid intersections between the geometry of the interface and that of the geometry in the scene. The fix is to create a group under the tool's camera object that has been scaled down to an extremely small value under which all interface objects reside. By placing this group very close to the camera, it is unlikely that a user will be able to create a condition where the interface intersects other geometry in the scene. A three-point lighting setup is also grouped under the tool's camera ensuring proper lighting of the interface and working model as the camera navigates the scene.

V.2. Real-time Animation

Using the Python scripting language, it is possible to intercept the signal representing a change in the current selection. When an interface object is selected, a check is made against all known interface object names to determine if there is a match. If it is a valid name, animation routines are called to let the user know a button has been pressed and the importing procedure begins. These animations are a necessary addition to the usability and simplicity of the interface. However, real-time animation is not a feature that is normally allowed outside of the built-in timeline. An interesting workaround is to increment a time function within the Python script while manually changing an attribute of the animated object at each time step. By forcing a screen refresh at the end of each step, custom animations can be displayed using any framerate desired. When an interface navigation button is pressed, the currently active group of parts is swept off the screen and replaced with the correct group of parts waiting off-screen. This transition minimizes the screen space necessary to display all possible parts and notifies the user that a change has occurred in the menu structure. While initializing the tool, all part groups are minimized off the screen except for the Rods which tend to be used first in a new file.

V.3. Importing Objects

After a new file is created and the program is loaded, the user will likely begin by pressing one of the Rod icons in the interface to create a new part to serve as a base. This object becomes the default top level parent for all parts to come. In order to create

a parent above this one, a check is run every time a new part is imported onto the top level object, and a dialog asks the user if they would like to make the newly imported part the top level parent. However, the normal operation for all parts other than the top level parent is for the newly imported object to automatically snap to the locator2 object under the currently selected part. The first step of the operation is to call Maya's import command to bring the new part geometry (GEO) from its file into the current one at the world origin. A new controller (CON) is imported from its respective file and bound to the new geometry and renamed to match the part. If the part is a Rod, Piston, HingeA, BallA, or ScrewA, a RootCON is used to control it. HingeB, BallB, and ScrewB parts have unique controllers that help identify their function (see Fig. 4).

A vector is created between locator1 of the new part and locator2 of the selected part. The new controller is translated along this vector, pulling the new geometry along for the ride, until the final position is reached. The controller is aligned in the direction of its new parent by looking at the difference between their respective direction vectors in world space. All of the transforms of the new controller are set to zero in its new location to make this its default position and orientation. A similar sequence of events occurs when a face is selected. The import command is called and the controller is connected. This time, the average position of all the vertices connected to the selected face is obtained and a new temporary locator is created in this location. A vector is created between locator1 of the new part and the temporary locator to move the new controller along. The orientation of the new part is obtained from the normal of the selected face by comparing it to its current orientation (see Fig. 3).

Once the part and controller are in their final position and orientation, constraints are used to connect all the parts together to obtain the desired behavior.

One might assume that a condition might be created where children parts are connected cyclically back to their parent or grandparent parts, freezing the articulation controls in place. Within the framework of the tool, constraints are only created upon the initial insertion of a part, avoiding this cyclical condition entirely. If a user chooses to manually connect parts in this way, Maya will warn them that unexpected behavior is likely to occur.

In order to keep the scale of each part localized from its children, groups are needed to buffer the effects of inherited transforms. Each part geometry and controller is grouped with itself one time and all constraints are created at this level of the hierarchy. Each controller's group is parent constrained (translation and orientation) to the geometry of its parent's control with the exception of the top level parent control (whose implied parent is the world). This allows the selectable geometry of the controller to move and rotate its children by transforming the group above them, keeping their local transforms at zero. The part geometry (GEO) has a slightly different set of constraints. The group above each piece of part geometry is scale constrained to its corresponding controller to obtain localized scaling. This group is also parent constrained to the same controller so it will inherit translation and orientation as well. The overall effect is to keep the geometry "pinned" to the surface of its parent as the parent is scaled. This produces beneficial results for the end user because scaling operations no longer become a destructive force further down the hierarchy of children (see Fig. 5).

V.4. Editing Operations

A number of editing operations are available to the user to modify parts and groups of parts that have already been imported. They include Copy, Mirror, Pin, and Swap. The Copy command duplicates a part along with all of its children while correctly maintaining all constraints. The user selects the control of the part to copy

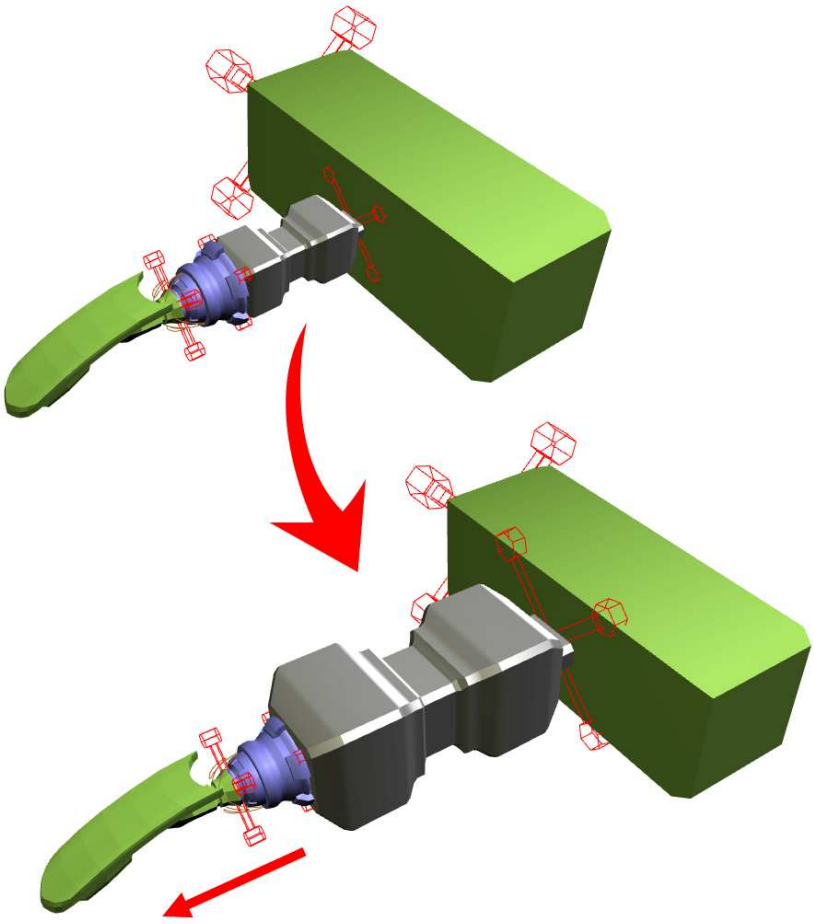

Fig. 5. Scaling Within the Hierarchy. Scaling the grey part in the center slides the blue part and its children forward.

and presses the Copy button. A recursive algorithm begins to search through the constraint hierarchy to find all children associated with the part. Once all the children have been selected, Maya's duplicate command is called with special parameters to maintain the constraints. One drawback to this approach is that Maya's handling of object names is not comprehensive; thus, requiring all newly copied object's names to be verified and fixed after the copy operation is complete. The newly created top level controller becomes the current selection so that a single mouse drag can efficiently place the copied group in a new location (see Fig. 6).

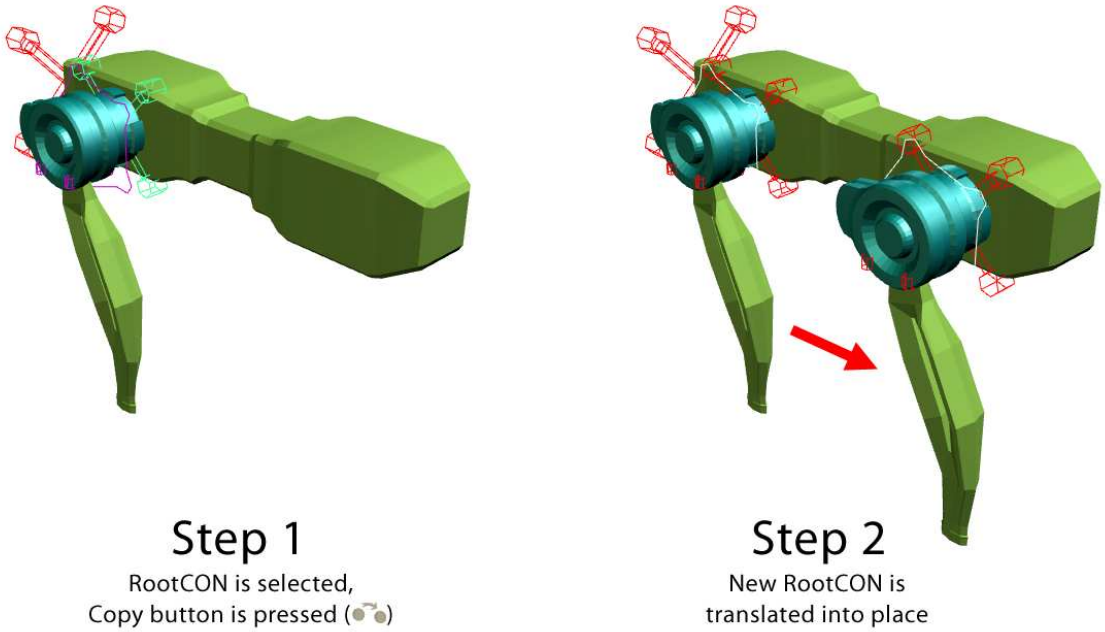

Fig. 6. The Copy Command.

The Mirror command takes a similar, but more involved, approach. Again, the desired part's controller is selected and the Mirror button is pressed. A dialog appears

querying the user for information about which axis (relative to the parent) to mirror around. The same recursive algorithm is called to find the children of the base object and a copy is made as above. A data structure is created to map the information about which parts are connected with constraints and then all constraints are deleted. The entire set of newly copied parts is collected under one group and the group's appropriate scale axis (selected earlier in the dialog box) is set to -1. All transformations for the group and its children are collapsed to zero and then the group is deleted. The recursive data structure is then read to re-establish the previous constraints. This method allows the mirrored state to become the base state without switching the direction of the rotation axes (see Fig. 7).

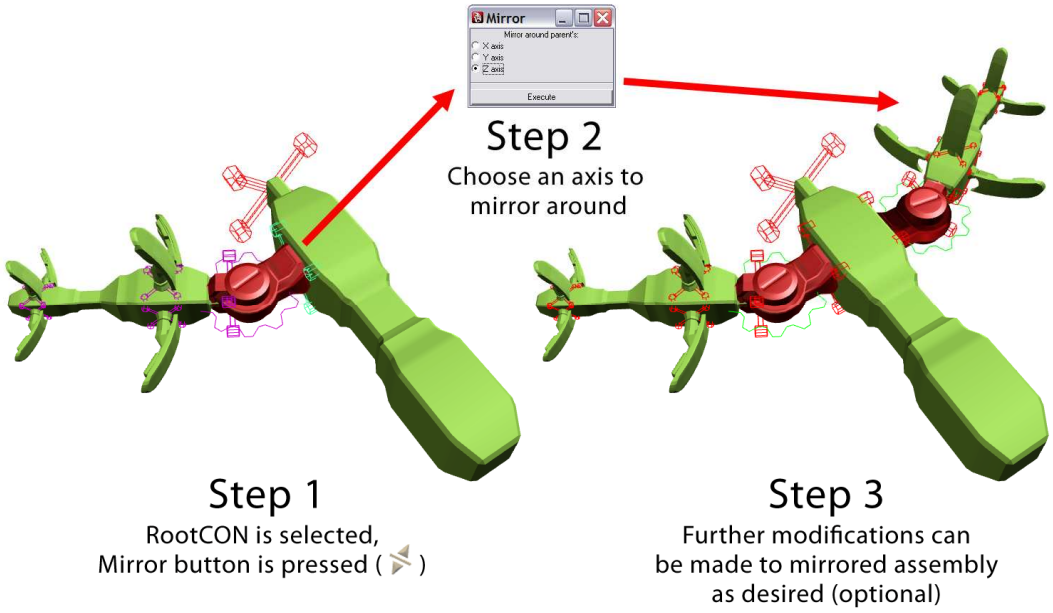

Fig. 7. The Mirror Command.

The Pin command is necessary when parts have been moved from their initial imported position. When a part is imported, the constraints that are created use the current object transformation matrices to determine a default home state for the constraint. Also, the pivot location of the group above the part's control is placed in the same location as the part itself. When a control is translated, scaled or rotated after the initial import, there now exists an offset between the transformation values of the part and its group. To make this new condition the default home state, the Pin command deletes the current constraints from the part to be pinned, moves the pivot points of the parent group to match the new location, sets the transformation matrices to zero, and creates new constraints to match the previous condition. Leaving out this step creates unpredictable behavior within the hierarchy, especially since the pivot locations are generally hidden from the user (see Fig. 8).

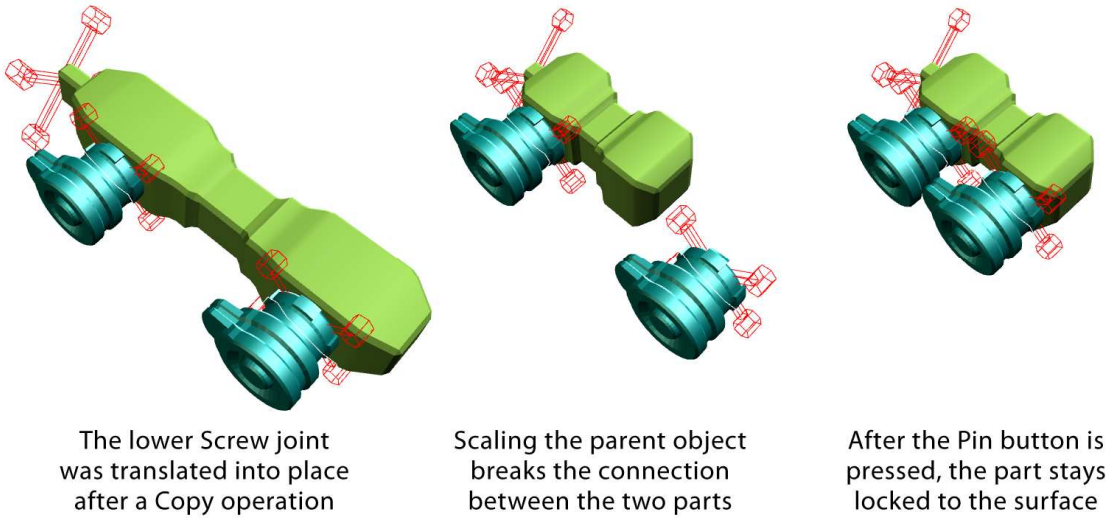

Fig. 8. The Pin Command.

The Swap command allows parts to be freely interchanged with one another and has several uses. During the modeling process, Swap facilitates the switching of function or form of any part in the assembly. For instance, a user may decide to change a Rod into a series of HingeA and HingeB parts to add joint functionality to a previously rigid region of the assembly. A dialog appears during the swapping process to query the user for their intent. The number of parts may be specified, with a value of two or more splitting the distance between the locators of the old part into equal segments. An external file may be specified that contains a list of all parts in the assembly, each with a corresponding replacement part. This facilitates a workflow tailored to the demands of modern visual effects pipelines by letting the modeling and rigging processes take place simultaneously. Consequently, there is no need to wait for the final geometric components to begin solving the problem of articulation. Scaling is used to create a rough approximation of the final geometry, since the relative proportions between different parts are generally all that is needed. As the final parts are swapped into place, there may be slight variations between them and the part they are replacing due to refinements in the final geometry. By default, the lengths of the articulated schematic model take precedent over the imported geometry to avoid unwanted shifting (although this can easily be modified.) Uniform scaling is used to change the length of the imported geometry without distorting it. Some final modifications may be necessary to fix minor issues once everything is in place (see Fig. 9).

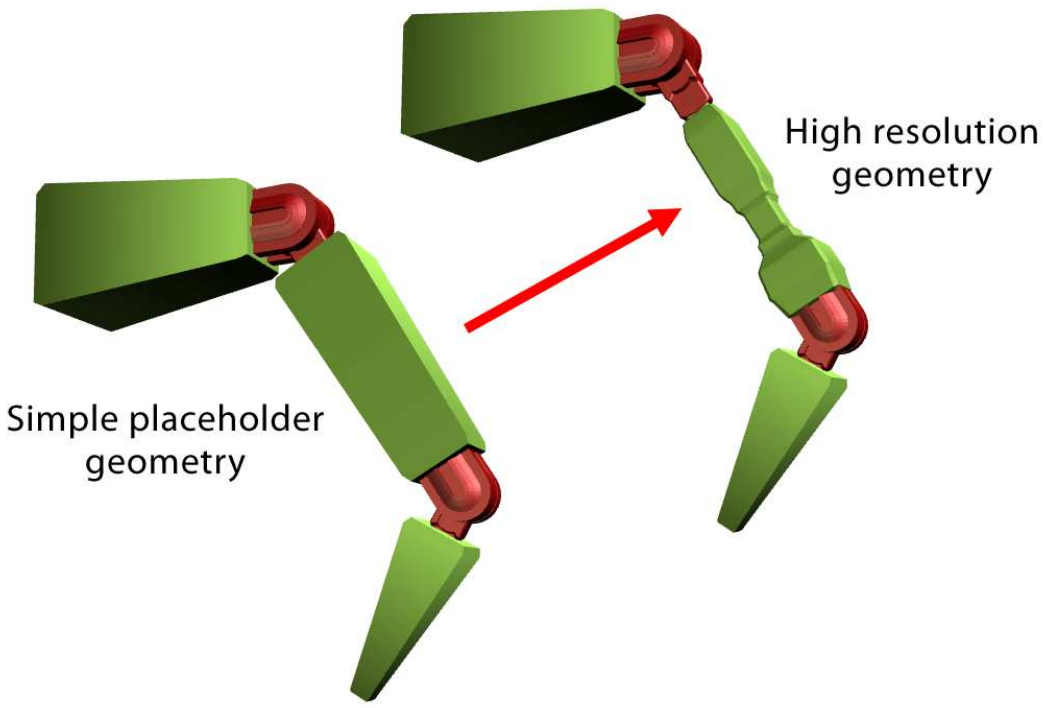

Fig. 9. The Swap Command.

CHAPTER VI

FUTURE WORK

The field of geometric modeling and rigging will continue to grow for years to come and further advances must be made to realize the full creative potential of the entertainment, robotic and related industries. One potential area of research is in the creation of an open-exchange database through which standardized parts can be shared among an international community. Users would be able to create new parts and govern the standards set forth by the system much like wiki pages are used for information or open-source software development organizations are governed. Another way the system could be extended is through the development of specialized rigs for specific object configurations. A study could be used to determine common high-level assemblies of parts and a new library of these rigs could be developed and integrated into the system. Another obvious extension to this thesis is to include organic geometry in a manner similar to Funkhouser et al. [7]. Combining organic elements into a single organic object is an interesting and much more difficult problem, primarily because the pieces must be joined into a single mesh upon completion. Issues such as topology and surface continuity become the most challenging aspects of this medium, however, many of the solutions offered by this work are still applicable to the rest of the process.

database to quickly test the compatibility and feasibility of different types of assemblies. Finally, the ability to share standardized databases of parts allows disparate communities of developers the option of connecting on a global, cross-discipline platform.

Fig. 10. Case Study: Bumblebee Process.

REFERENCES

[1] Autodesk Maya 2008, http://usa.autodesk.com, 2009.

[2] J. Smith, J. Hodgins, I. Oppenheim, and A. Witkin, "Creating Models of Truss Structures with Optimization," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 295-301, 2002.

[3] J. Oung, M. Sitharam, B. Moro, and A. Arbree, "Frontier: Fully Enabling Geometric Constraints for Feature-Based Modeling and Assembly," *Proceedings of the Sixth Symposium on Solid Modeling and Applications*, pp. 307-308, 2001.

[4] L.G. Da Silveira Jr. and S.R. Musse, "Real-Time Generation of Populated Virtual Cities," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 155-164, 2006.

[5] Roger Bush and Carlo Sequin, "Synthesis of Bent Sheet Metal Parts from Design Features," *Proceedings of the Fifth Symposium on Solid Modeling and Applications*, vol. 5, pp. 119-129, 1999.

[6] C. Morkel and S. Bangay, "Procedural Modeling Facilities for Hierarchical Object Generation," *ACM International Conference on Computer Graphics, Virtual Reality and Visualization in Africa*, pp. 145-154, 2006.

[7] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, et al., "Modeling by Example," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 652-663, 2004.

[8] J. Corney, H. Rea, D. Clark, J. Pritchard, M. Breaks, and R. MacLeod, "Coarse filters for shape matching," *IEEE Computer Graphics and Applications*, vol. 22, no. 3, pp. 65-74, 2002.

[9] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural Modeling of Buildings," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 614-623, 2006.

[10] R. Berndt, D.W. Fellner, and S. Havemann, "Generative 3D Models: A Key to More Information Within Less Bandwidth at Higher Quality," *Web3D Symposium Proceedings*, pp. 111-122, 2005.

[11] P.J. Birch, S.P. Browne, V.J. Jennings, A.M. Day, and D.B. Arnold, "Rapid Procedural-Modeling of Architectural Structures," *Proceedings VAST 2001 Virtual Reality, Archeology, and Cultural Heritage*, pp. 187-196, 2001.

VITA

Name: Christopher Ryan Wheeler

Address: Texas A&M University
C108 Langford Center
3137 TAMU
College Station, TX 77843-3137
c/o Tim McLaughlin or Philip Galanter

Email Address: c9ine@yahoo.com

Education: B.E.D., Environmental Design, Texas A&M University, 2007
M.S., Visualization Science, Texas A&M University, 2009