Gravity Control System: Realistic Balanced Poses and Animations

Tobias Remmers
Nkp04trs
June 2007

Thesis, 10 points, C level
Computer Science

Creative Programming
Supervisor/Examiner: Sharon A Lazenby
Co-examiner: Torsten Jonsson
Gravity Control System: Realistic Balanced Poses and Animations

By

Tobias Remmers

Department of Mathematics, Natural and Computer Science

University of Gävle
S-801 76 Gävle, Sweden

E-mail:
nkp04trs@student.hig.se

Abstract

The Gravity Control for Maya will be extraordinary beneficial to an animator trying to create realistic animation, by calculating the center of gravity and area of balance. This control will provide the animator with the ability to rotate around the center of gravity and keep the character in a balanced pose. With that ability, the animator can easily create accurate poses and animation, such as mid-air flips. The system also supports a vast number of characters with different shapes, sizes and number of limbs.

Keywords: CG, Animation, 3D, GUI, Maya plug-ins, Mel, Mel Procedures, Custom Nodes and attributes, Rig Scripting, Constrain Scripting, Connection Scripting
# Table of Contents

1 PURPOSE ........................................................................................................................................... 4

2 QUESTIONS AND DEMANDS ............................................................................................................. 4

3 BACKGROUND AND TECHNOLOGY .................................................................................................. 4

3.1 Problem Description .......................................................................................................................... 4

3.2 Research and Reference Material .................................................................................................. 5

3.3 Current Research and Technology .................................................................................................. 5

3.4 Expected Result ................................................................................................................................. 5

3.5 Technical Explanation ....................................................................................................................... 6

3.4.1 MEL .................................................................................................................................................. 6

3.4.2 Procedure ......................................................................................................................................... 6

3.4.3 GUI .................................................................................................................................................. 6

3.4.4 Expression ......................................................................................................................................... 6

3.4.5 Nodes ............................................................................................................................................... 7

3.4.6 Node Connections ........................................................................................................................... 7

3.4.7 Constrains ......................................................................................................................................... 7

3.4.8 Group Nodes and Parenting ........................................................................................................... 7

3.4.9 Locator ............................................................................................................................................. 7

3.4.10 Attributes ....................................................................................................................................... 8

3.4.11 Variables and Arrays ...................................................................................................................... 8

3.4.12 Node-tree and Hierarchy ............................................................................................................... 8

3.4.13 Geometry ...................................................................................................................................... 8

3.4.14 Clusters ........................................................................................................................................... 8

4 DEVELOPMENT PROCESS ................................................................................................................ 9

4.1 GUI ..................................................................................................................................................... 9

4.1.1 Character List ............................................................................................................................... 9

4.1.2 Node List ......................................................................................................................................... 9

4.1.3 Add Characters ............................................................................................................................ 10

4.1.4 Remove Characters ....................................................................................................................... 11

4.1.5 Sorting Characters ......................................................................................................................... 11

4.1.6 Adding to the Node List ............................................................................................................... 11

4.1.7 Preparations for Writing Weight Node Procedures ......................................................................... 12

4.1.8 Adding Weight Nodes ................................................................................................................... 12

4.1.9 Removing Weight Nodes .............................................................................................................. 12

4.1.10 Sorting Weight Nodes ............................................................................................................... 13

4.2 System ............................................................................................................................................... 13
1 Purpose

The purpose of my research is to provide animators aid in making realistic looking animations by helping them keeping track of the center of gravity and balance of the characters. By checking that the center of gravity is within the area of balance, not only in the poses but in the interpolations as well, the system will be an immense help in making the animations look believable and correct. Also the system will provide the user with more possibilities to pose the character in a variety of poses and animated movements, which are not possible or very hard to achieve with an ordinary rig. Examples of poses and animations that would benefit would be handstands, flips and action movements.

2 Questions and Demands

The questions that are most obvious are: Can a system that dynamically calculates the center of gravity be created in Maya? Will the system be created where it supports more than one character? Can each character have different body shapes, looking at mass and number of limbs? For this system to work a method is needed to define the area between and under the limbs that touches the ground, either mathematically or via a deformable geometry. The center of gravity must be a point where location is calculated from the limbs and masses of the body. There must be a way to test if the center of gravity is above the balance area. This can be created by drawing a curve straight down from the center of gravity. The system can then use that curve to see when the center of gravity is above the area of balance.

Another way to get the same information is to use an expression to check the position of the center of gravity against the area of balance. Other questions that should be solved are: Can custom nodes be created in Maya without shape nodes or if the nodes within Maya can be used as they are? This would be useful for placing the attributes for characters and paths to other nodes. Is there a way to rotate a character around other points in space than the center of gravity and still keep the entire rig intact? This would be useful for such situations where a character swings between pipes or poles.

The final demand of the system would be can the calculation to mass be automated? Can I calculate the distance from a vertex to a control object and compare it to the distance to form that vertex to other control objects in the rig? I will try to include this in the system as a mass calculation. This might not be possible for a number of reasons, such as lack of time or skills. However, I would like to see if I can find a way to calculate volume from the control object to all vertexes of the geometry closes to it. This will then be used to figure out the mass in that area. If this can be solved, the weights would not need to be set up for each object manually, which will perhaps provide accurate results or at least lighten the work of setting up the rig.

3 Background and Technology

3.1 Problem Description

When animating in a 3D application, such as Autodesk Maya, the normal way is to use keyframes, which is a process of posing the character on some frames and having the computer calculate the motion in-between the frames. When posing and moving the character, the model will have digital bones inside the geometry and the geometry will be bound to these bones. With this method, it is difficult to make believable animation, especially for beginners. The problems occur because the animator may be inexperienced and the character’s movement will look unnatural. If the character is unbalanced, the human eye will pick up on it directly. Since we have been watching how people move all our lives, subconsciously we can see when a characters movement is too stiff and unrealistic. When a character walks, it will sway from side to side, because its center of gravity must be over the area of balance, which is the area between all the limps that touches the ground. For example when you are standing on two feet, the area is under and between the feet. When standing on all four feet/hands, the area is under and between the front feet/hands and back feet. This may sound easy, which is because humans are symmetrical and proportional. If the character
is non-proportional where the right arm is ten times bigger than the left, there will be a huge impact on where the center of gravity is located, which will mean that the character must lean to the left to stand balanced over the feet. This can be difficult to see and correct manually especially in all poses when the character moves and the weight shifts from side to side.

The system which I plan to create in this research will also help in performing realistic movements. For example when a character is doing flips in mid-air, the method to create this movement in old rigs is to either have the character rotate around the main bone usually located in the hip, which will appear unnatural.

The second method is to create a center of gravity to rotate around, which will have to be placed and moved by estimation. Although this is much better than rotating around the hips, it is not very dynamic and has to be replaced if the mass changes or when the characters body is repositioned or deforms.

3.2 Research and Reference Material

I found a variety of information sources for writing this research paper, such as programming books, mainly “Complete Maya Programming” [8] and “Mel Scripting for Maya Animators” [11] but “Learning Maya 5 | Mel Fundamentals” [12] might prove useful as well. I also found some previous research papers called “Automating the rigging process for humanoid characters” [5], “Optimizing of the rigging process in Maya” [6] and “Building Controls for Simultaneous use of Keyframes and Expressions” [7]. These papers contain useful information on using Mel to setup expressions, constrains, and to setup rigs and solve rigging issues that might occur when building the system. A number of digital Encyclopedias like Wikipedia [9] and Encarta [10] will be also used, where I might find information on mathematical or physical aspects of gravity, weight and mass.

3.3 Current Research and Technology

When researching this topic, I noticed that there is no system currently in Maya to calculate and update the center of gravity for any purpose, especially animation, nor could I find any information on 3D forums, such as CGSociety [1], Omans3D [2] and HighEnd3D [3]. I was unable to locate any academic papers on this same type of system for which I wanted to develop on academic databases similar to ACM [4]. These papers were often written on subjects too far-off from my system to be of any use in my research. However, I did find a few papers written from former creative programming students “Automating the rigging process for humanoid characters” [5], “Optimizing of the rigging process in Maya” [6], and “Building Controls for Simultaneous use of Keyframes and Expressions” [7]. These papers did not have anything with controlling the position of center of gravity; however they used many of the same techniques, which I will have to use to setup this system via Mel.

3.4 Expected Result

I expect the outcome to be a plug-in made in Mel, which can setup gravity controls for as many rigs that need it on the same scene. In each rig, the user will be able to set an unlimited amount of control objects. Each object has a different amount of influence on the location on the gravity control. Some control objects will also influence the rig itself. All influence from control object will probably be created via expressions or constrains. The influence on the rig will first come from the feet to move the entire rig; therefore the rig movement can be controlled by only the feet if the animator wishes. Secondary influence takes place from the hands to move the spine. This will let the animator lean the character one way or the other by repositioning the hands. If time allows, there will be an option to correct the head to be upright at all times during the automated leaning.
3.5 Technical Explanation

3.4.1 MEL

Mel, Maya Embedded Language, is one of the two internal programming languages in Autodesk Maya. This language can be used to write the main scripts. In the scripts, it can provide the user with the possibility to create the expressions and node connections and attach them to the correct attributes on the correct nodes with the click of a button. It is also possible to use Mel to write scripts that creates windows to make GUIs, (see 3.4.3), where working and editing with the gravity system becomes much easier. Former owner of Maya, Alias|Wavefront explains it: "MEL stands for Maya Embedded Language. It is a language designed for giving instructions to Maya®. You can use it to automate tasks, customize Maya, create expressions or build scripts that will extend Maya’s existing functionality.”[12.a]

3.4.2 Procedure

A procedure is a piece of code that can be called at any point on the program. Some very small programs can be written in just one main procedure, however most uses a network of procedures to keep order in the program and to enable the computer to go back and forth through the program and reuse procedures as much as needed. When a procedure reaches its end, the program will stop if not called from within another procedure. Alias|Wavefront explains it: “A procedure is a grouping of MEL commands loaded into memory that can be executed by typing a single command” [12.b]. Whereas David A. D. Gould gives a slightly different explanation to the procedure: “A procedure can be thought of as a machine into which things are fed and which processes them to produce a result” [8.a].

3.4.3 GUI

GUI means Graphic User Interface and refers to a hierarchy of windows and window content. This can all be created and edited via Mel, GUI is a powerful way to make the system extremely easy to work with and obtain a response. GUI is one of the ways the system will show data to the user.

3.4.4 Expression

Expressions are small lines of Mel code that is assigned directly to an object’s attribute, for example pCube.tx = (time / 3). These can in some, not all, cases be more correct and easier to edit than constrains and faster to evaluate than node connections. Since they are Mel codes, they are the easiest to assemble in a script. David A. D. Gould explains expressions in this way in comparison to keyframes: “MEL commands are evaluated at each frame, and the result is stored in the attribute to which the expression are assigned. This technique is commonly known as procedural animation. This is when a program (MEL commands) commands the animation rather than it being controlled through keyframing. Expressions can be used to create complex animations with little or no manual intervention.” [8.b].
3.4.5 Nodes

Nodes are a collection of attributes. There are a vast number of nodes and all with very different uses. The most common nodes are shape and transform nodes, for example Polygon cubes or Nurbs cylinders. These objects appear in the scene and then have both a transform and a shape node. In the transform node, all position and shape data are stored, such as transform, scale, and rotate. The nodes that do not appear in the scene are used to manage data, for example multiply/divide node that is used to multiply or divide values or the file node, which is used to load a file into the scene that can be used for a texture. These nodes are often referred to as render nodes since they are often used in networks for making textures and changing other render related data, however these networks can also be used for many other applications.

3.4.6 Node Connections

Node connections are the simplest way to get attributes and nodes intergraded with each other. For example, if node A’s translate Y would be connected to node B’s translate Y then the two nodes will move at the same time. Special nodes can be added between, such as a multiply divide node. Then take node B’s translate Y, multiply that by -1 and feed that in to node A’s translate Y, which causes them to move in opposite directions from each other. In some cases, they are faster than expressions. You can easily follow the connections in the built-in editors in Maya. However, complex node networks might be slower than expression and hard to setup via scripts.

3.4.7 Constrains

A constrain can in most cases be explained as a more or less simple node connection or expression presets. There are many different kinds of constrains such as point, rotate, parent, aim, and pole vector. I will be using the first three, which will cause one object to be constrained to the position and/or the rotation of another object. Constrains have their usefulness because they are easy to setup and the fact that they offer simple solutions such as aim, pole vector and parent constrains. The same result would be very hard to re-create in expressions. The good thing about constrains is that they do not affect the hierarchy, therefore by using constrains, a relationship can be obtained between two or more nodes without changing how the scene is being managed. The affected nodes can also be located anywhere in the hierarchy, making the scene easier to organize. The downside is however that they can be hard to manage via MEL scripting.

3.4.8 Group Nodes and Parenting

A group node is the simplest node in Maya. It only has the basic attributes; although it is often very useful in sorting other nodes. Parenting is to make node-B a so called child of node-A. This will allow a user to move node-B five units in the Y-axes while node-A will not move. A child does not control the parent. If node-A is moved five units in Y-axes, then node-B will also move five units in the Y-axes. The parent is in control of the child. Groups can also be used for specific requirements other than order. They are created at the origin so all attributes are at 0. Also all objects that are parented under the group moves when the group does but individual of each other. If a child node is animated to move up and down in the Y-axes and its parent is animated to move along the Z-axes, this will provide a wave motion to the child node since the animation of the parent is added to the motion of the child. To have two objects put together under a group node is referred to as grouping them together.

3.4.9 Locator

A locator is basically a group node with a shape. Similar to a group node; it only has the basic attributes however it has a shape to represent them. A locator is shaped like a cross and, as the name implies, are often used to pin-point a specific location in 3D-space.
3.4.10 Attributes

Attributes is information associated with that node. The most common attributes that many of the
nodes in Maya has are translate, rotate, scale and visibility attributes. These attributes are used by
all nodes that will appear in the scene. A user can also setup custom attributes which are attributes
that the user has added manually to the node. The user can add as many attributes with any name,
set the data type, as well as limitations on the values.

3.4.11 Variables and Arrays

Variables are like a custom attributes for MEL scripts. The programmer chooses the data type and
name for the variable and put values in it to be used later in the script. There are many different
data types for variables such as the string that is used for text. The “int” is used for number such as
3, 45, and 114. The float is used for number such as 3.14, 64.11 and 0.2. An array is a group of
values like a wardrobe with infinite amount of space to save different values, however they all
have to be the same data type as the array.

3.4.12 Node-tree and Hierarchy

The node-tree is a name for how nodes are parented to each other. A skeleton is by its nature a
hierarchy of joints; lower arm is parented to upper arm which is parented to shoulder and so on up
to the root joint which often is in the hips. To keep order in the scene most artists, groups the
skeleton together with the geometry, which in some cases the geometry is many different nodes
grouped together with other nodes associated with that character such as animation icons. If it is a
large scene then the characters may be grouped together.

3.4.13 Geometry

Geometry is the name of the shape object and what kind of algorithms that is used to create the
shape in 3D-space. There are three different geometry types in Maya. The most common is the
polygon that often is the first geometry type associated with 3D-art. An object made from
polygons is made up of many different planes of different shapes and sizes. Each plane is a
polygon that is the area between different points in 3D-space. Each point on the polygon is called a
vertex, from one vertex to the next runs an edge that divides one polygon form the neighboring,
and the entire area is called a face. There is also NURBS which is a mathematical area between its
points. The points on a NURBS object does not connect to the surface, they are called a control
vertex or a control point. A row of points is called a hull and for every hull there is a line on the
surface called an isoparm. The last geometry is the Sub-D, which is often referred to as a
combination of polygons and NURBS. The Sub-D has a cage associated to the shape that can be
edited in a polygon workflow with the same tools and it also has points in the Sub-D mode that
works similar to points on a NURBS object.

3.4.14 Clusters

A cluster is a small object in the scene that has no shape of its own only an icon. The cluster can be
associated to a number of objects much like a group or constrains. It can also have the same effect
on only parts of an object such as vertexes. This allows animation or deformation to easily be
created to that area of the object.
4 Development Process

4.1 GUI

4.1.1 Character List

First thing to create was sketches of how the GUI of the system would appear. Then, I could start typing in the code. The main window was the starting point, following my previously drawn sketches. I created a window containing one scrollable list and six buttons (Figure 1). The buttons are to be used for adding, removing, editing characters, and also to sort the characters in the list. I also added placeholder procedures for every button and other procedures, which I knew I would need them later, for example updateCharList. When the GUI of the main window was made, I started on the list where the characters would be listed. An empty Group node was created and prepared to be the root node of the system by locking all original attributes and adding custom attributes. These were a string array for the characters and an integer for the number of characters. Then, the procedure updateCharList was finished by extracting the information of the list on the system node and feeding it into the list on the main window.

![Figure 1. Main window of the gravity control system.](image)

4.1.2 Node List

The next GUI element was the node list. Following the same layout as the character list, a window was made with a text field to edit the name of the character and a scrollable list for the weight nodes. The window also has three buttons, a float slider and a float field to be able to edit, create, and remove weight nodes (Figure 2). These were connected to the main window by the edit button on the character list.
4.1.3 Add Characters

Next, the addChar procedure was needed, where the user is able to add a new character to the system. Every new character is parented into the root node of the system. At first when a new character was created, it would be named “characterX,” where X symbolizes the number of currently existing characters incremented by one. This was very problematic when a user creates three characters, removes the second one, and then creates one more. That would have caused an error. Then, a prompt dialog was chosen instead (Figure 3), where the user enters the name, which then is put through a series of tests to see if it is acceptable (Figure 4 and 5). Therefore, the script cannot continue if the name is not typed in and unique.

Later another test was added to see if the name began with a number since that would also mean problems for the system. The problem occurred when Maya removed all numbers in the beginning of the name. When the new group node for the character is created, named, and parented under the root node, the problem of adding it to the character list appears. A setAttr command was tried, however it does not take an array as an argument but after talking to Niklas Folkegård, who is a temporary teacher in MEL in the Computer Science Division, at HIG, and searching on an internet forum [1.a], I found that I could work around the problem. By using the arrayToString command, putting the whole setAttr command to a string that obtained feed to the Eval() command, which will evaluate the string as a MEL command, the problem could be resolved with only five lines of code. This was written in a different procedure, where it could be skipped if not needed by other procedures later in the program.
4.1.4 Remove Characters

To remove characters from the list was by comparison far easier. Since having the update character list in a separate procedure to remove the characters, the stringArrayRemove command was used to delete them from the character list on the system node and a simple for-loop to delete the actual nodes from the scene.

4.1.5 Sorting Characters

To finish up the Main window, a code was added for the three sorting buttons. The move up and down used the index number from the character list in the window, which was needed to move the strings in the array up or down. An extra procedure was added to reselect the names of the moved characters in the window, where they would not have to be manually reselected. I also wanted to add in the ability to sort the list alphabetically, which was more straightforward than expected. A function within MEL was found that was called simply sort(). It did exactly what I wanted, therefore on three lines, the list was retrieved from the gravity control, sorted and sent to the updateCharList procedure.

4.1.6 Adding to the Node List

The first item that needed to be fixed in the second window was the ability to change the name of the character; because it was listed among the procedures for the main window as a character related procedure rather than a weight node procedure. I also thought it would be simple, however it was not. To be able to change the name throughout the system and the scene, I had to query for both the new and the old name to be able to rename the node. By obtaining the index number of the selected character in the GUI character list, the procedure could locate and rename the character in the systems character list on the system node. This was necessary when the list in the GUI was not the same as the reference list on the system node, it did however have an identical setup with the same node name on the same place. To finish the procedure, the active character was updated to the new name.
4.1.7 Preparations for Writing Weight Node Procedures

Then, writing the procedures for the weight nodes began. I thought that many of the procedures for the weight nodes would be very similar to the character nodes. Though many of the procedures in the beginning might have seemed similar, I choose not to try to rewrite the character procedures where they could work on weight nodes as well. I feared that the differences in the needs of the procedures might appear later. The character procedures were used as a reference when writing the new procedures for the weight nodes. This was easier than having to go back and rewrite the old character procedure where they could be used with characters as well as the weight nodes. I ended up writing new procedures specifically for managing weight nodes. I also thought it might be easier to bug search and test run if they were kept separate.

The first realization was that there might be many weight nodes on one character; therefore I started to add features to the window where the user would be able to manage the nodes just like the characters. I added a text field to change the name of the node and move up, move down and sort buttons. After the new GUI elements were added with their placeholder procedures, I started working on adding the weight nodes. I also began adding the placeholder procedures for setting up the system that would be written once the GUI was finished.

4.1.8 Adding Weight Nodes

I initiated the weight node part of the GUI programming by adding the window for creating new weight nodes (Figure 6). It did not appear as planned, since I felt the need to add a text field, where the user could name the nodes on creation. To add the new nodes to a GUI list was a little bit tricky, since the list that was storing the names of the weight nodes was located on the character node. This added to the complexity of the commands and increased the opportunities for errors. I could easily build on my resent experience and code from the character list and did not need to search for help via internets and teachers, which was a lot quicker. I decided to list them following the pattern “Node Name – Constrain Weight – Node Behavior.” This enabled the user to obtain all information needed just from looking at the list. Even though it is not possible to change the behavior of the node after it has been created, it was important to add the information where it is easily viewed by the user.

**Figure 6. Add new weight node window.**

4.1.9 Removing Weight Nodes

When writing the procedure to remove weight nodes, it became very clear that there was a problem with how the weight nodes name was added to the GUI. Since I wanted the GUI list to have its items portrayed “Node Name – Constrain Weight – Node Behavior” when getting the strings to identify the nodes, they all had this extra information that was added to the name. I could not use the strings from the list to identify the node as I did with the characters. To fix this, there were a few options: I could have one extra array with just the node names, but that would mean considerable more work in the remove weight node procedure than needed. Therefore, I added an attribute to the node that said how many characters of the string in the array that was the node name. When I experimented with this, Maya renamed the nodes that my program created if there was a node somewhere in the scene with the same name, no matter where in the node tree it was located. That behavior did not appear when manually renaming the nodes. I decided to add the
character’s name to the node name. This corrected the name problem however it did not correct the problem with the names in the list.

Then, I tried a different approach to change the procedure for updating the GUI list where the program only adds the information of weight and node behavior just before adding the string to the GUI list and not have the information added to the weight node list on the character. By using this method, I could get the index numbers of the nodes selected in the GUI list and cross-reference them with the names in the weight node list on the character and thereby get the names of the nodes in the scene that are to be deleted. For every node deleted, the name temporarily gets saved aside to an array which, similar to the characters, is used to delete the posts from the weight node list using stringArrayRemove. Then, the GUI is updated and the procedure is completed.

4.1.10 Sorting Weight Nodes
To complete the last of the GUI, I created the move and the sort procedures for weight nodes. This was easier than I thought, since there were problems converting the character procedures to work on weight nodes. However, this time it did not take more than changing some procedure names and adding a few rows of code to get to the nodes that are at a deeper level in the node structure. After fixing some typos in the GUI and adding to the comments in the code, I was ready to start working on the system, setting up connections and expressions.

4.2 System

4.2.1 New Systems
To start off the systems part of the development, I began at the most obvious place, the procedure for creating new gravity control setups. Whenever a new character is created, this procedure will run and add the center of gravity, area of balance, as well as the clusters that deform the area of balance. It will also create attributes that will make it easier to locate the clusters when needed.

4.2.2 Constraining the Center of Gravity
Once the center of gravity was established, it needed to be integrated with the character. When a new weight node is created, the center of gravity obtains a point-constrained to that node. This of course means that when the first node is created, the Center of gravity will be right on top of it. When this node is positioned in the right place and the second node is created, the center of gravity will be somewhere in between depending on the weight settings of each node.

4.2.3 Area of Balance
A polygon plane was added to each character which will act as a reference to the system, where the character can stand balanced. The plane is a one faced, four cornered polygon also known as a quad. Since it has only four vertexes, it will be easy to manipulate. Each of the vertexes of the plane obtains a cluster associated to it (Figure 7), where the plane can be manipulated into a shape that covers the area under the character and between the limbs on the ground.
4.2.4 Warning Sphere

Now, I was soon able to obtain usage from the system. I added a NURBS sphere to each character. This acted as a warning light that showed the user if the character is off balance. The sphere is point constrained to the center of gravity but with an offset of ten units in the Y-axes (Figure 8), which can later be edited by selecting the sphere and changing the offset on the point constrain. This was all done in the first procedure of the system, the same procedure which creates the root node, as well as the foundation for the system. I also add functionality to create a bright red shader which is mapped to the color of all the warning spheres that is created. I named the shader simply GravityControlSystemWaringSpheresM.
4.2.5 Warning Sphere Setup Expression

To make the sphere turn on and off its visibility, depending on if the center of gravity was above the area of balance or not, my first thought was to use an expression. This was a challenge; of course, I could not test if the area of balance’s X- and Z-attributes were the same as the center of gravity. This would only be true if the pivot points of the two objects are on top of each other. I needed to test against the polygon’s entire area. I experimented using the X- and Z-attributes of the clusters that were on the vertexes. I took the highest and lowest X- and Z-value and compared it to the values of the center of gravity X- and Z-attribute.

There were two reasons why that did not provide acceptable results. The first was that all of the attributes on the cluster were set to 0 when it was created. It gave an offset of the distance between the origin and the vertex where the cluster was created. This made the area mathematically smaller in the calculation and if the area of balance would be as small as or even smaller than the size it was created. There would be no area where the expression would calculate the character as balanced. This could be helped some by using the xform command to query the cluster for its actual position in the 3D-space and not the local X- Z-attributes used before. I obtained this idea from a post on the internet forum CGSociety [1.b]. This provided an offset in the opposite direction since the cluster was not perfectly centered over the vertex. But as I said before there was one more major problem with the expression. It did not take into account the shape of the area of balance and assumed that the area was a square; that it fully stretches from its highest to its lowest point in both X- and Z-axes. This is of course not the case if the character is for example an ant is standing on its right middle leg and on the left it stands on all three; this would give a triangular area of balance.

4.2.6 Warning Sphere Setup Node Network

To fix the problem previously mentioned, I had to continue as originally planned and use the polygon rather than its clusters. I removed the expression and used a node network instead (Figure 9). In an old tutorial movie from Alias|Wavefront [13], the tutorial talked about the closestPointOnSurface node for NURBS surfaces. After reading the Maya help file, I found that a similar node existed for polygons. I used the area of balance and the center of gravity and connected them to a closestPointOnMesh node, which would output the point on the area of balance that in 3D-space is closest to the center of gravity. I could then take that point, test the distance in only the X- and Z-axes to the center of gravity and if this distance was higher than 0, the center of gravity is not above the area of balance. I could then via a condition node set the visibility of the warning sphere to turn on and off (Figure 10).

![Figure 9. Node network for the warning sphere.](image-url)
4.2.7 Foot and Hand Weight Nodes

The hands and feet nodes are different than the normal weight node even though they all have effect on the placement of center of gravity. The difference is the area of balance since both hands and feet are supporting the body if they touch ground. The area of balance needs to deform to match the area between the hands and feet nodes. To make this work, I decided to add four corner nodes to the hands- and feet-nodes (Figure 11). Then, the clusters on the area of balance could be constrained to the appropriate corner node. This means that the upper right corner of the area of balance is constrained to the upper right corner node on the hands and feet.

4.2.8 Deforming the Area of Balance First Attempt Using Expressions

Once the clusters were constrained to the corner nodes, I needed to write up a code that could decide which of the feet or hands that had the best corner node to deform the area of balance. To be able to test the position, I used an xform command that I read about on CGSociety [1.b]. This means that if a character has four legs or is standing on its hands and feet; the upper right corner of area of balance should be constrained to the node in the upper right corner of the hands or feet located further to the right and highest up. In other words, it is the foot or hand node which has the highest X value and lowest Z value; if the character is facing positive Z and is viewed from the front.
First, I decided to set a weight value on the hands and feet nodes as well as the corner nodes surrounding the main node. Via an expression, I multiply these together and set the result as the weight of the point-constrain to the cluster. This part of the setup proved to be much harder than I had predicted. My first attempt was creating an expression that saves the position of the tested corner node to an attribute on the character, which would indicate the placement of the best node. If the next tested node is better placed, it would be used instead and replace the old node as the best node for the rest of the tests.

This did not work for a number of reasons. First off, Maya was throwing errors when the second foot was created since the attribute on the character holding the placement of the best node was associated with the expression from the first foot, because an attribute can only have one expression at the time. Secondly, the attribute for the placement of the best node was never reset. This result concluded if a node was in the best position and those attributes was saved and the node then moved back, towards the origin, to a position less desired that position would be tested against the saved numbers even if there was no node in that position.

The third problem was similar to the second in that the weight values were never reset back to zero. This was problematic if the first node tested was in a good position and was set to deform the area of balance, but the second node was in a better position. The first was not reset to zero so both nodes had a weight value of one and then gave very unwanted results. I tried to solve this problem by putting the attributes in arrays so that they would be kept individually, thus being accessible for the testing. This proved impossible when I could not add arrays of the right data type as node attributes; therefore I had to take a different approach.

4.2.9 Deforming the Area of Balance Second Attempt, Expressions and Mel Hybrid

To proceed around the problems that I encountered with my last attempt, I tried to use Mel procedures to test manage the attributes on the nodes. First, I started by writing up two procedures, one for testing the position and one for resetting the weight value. They operate by looping through all weight nodes and if the node has hand or foot behavior, the process continues and goes in to its corner nodes and resets or tests position for the corner node of the right type. Unfortunately, this approach did not work either.

The first problem was in the expression; it said that if the node had the same name as what the test procedure had returned as the most suitable node to use, the weight value would be set to 1 and make the weight value associated with an expression, where the reset procedure could not reset it. This was solved by adding the setting of the weight value of the right node to a part of the reset procedure, although this meant that no value was associated with the expression that was calling the procedures. This was a major problem since an expression with no associated values will not execute. To make this work, I needed a method to call the procedures automatically and often enough, where it will move with the node when the user is working. This meant that they need to execute even when the user is working in the scene and not just when the animation is running, therefore to have it run by changes in the global time or frame attribute would not be enough.

4.2.10 Deforming the Area of Balance Third Attempt Using Only Mel Code

After the failure of the expression and Mel code hybrid, I stumbled over a command in the Maya help files, while searching for a possible way to create what I needed. The command was scriptJob, and it would allow me to create a listener that reacts on given condition and then calls a procedure. I created two script job for every hand or foot node, one that listened for changes in the nodes x value and one for the z value. When any changes occurred the script job would run the procedure for setting up weights.

This procedure caused several problems before I could make it work. I could basically use the code from before, but since that code never worked properly, it took a lot of rewriting before getting it to behave as I needed it to. The final version of the procedures was a setup of four procedures layered after one another. The script jobs calls the first procedure that resets all nodes of the character to a weight of 0, and then it asks the second procedure for which nodes that should have
a weight of 1. The second procedure checks the position of every node on the character compared
to the center of gravity. The procedure can then categorize the node based on the relation to the
center of gravity into three categories; Best, Good and Worst. It is created so only the nodes in the
best possible position will be used in the next step. This is where the nodes progress to either of the
last two procedures. These procedures will then test which one of the remaining nodes is in the
absolute best position for being used to deform the area of balance. The procedure that manages
the good and the worst group of nodes check the position of the node and distance from center of
gravity and compare it to the rest of the nodes in the same group. There are some small differences
between testing for the worst and the good nodes, but not big enough to need two procedures.

The whole setup was made so that the program only tested as few nodes as possible, if there were
any nodes in the area considered best those would be the only ones tested. If no nodes were found
then the program would use the nodes found in the area considered good instead. Only if the “best”
area and the “good” area were empty on usable nodes the test would run for all nodes, ergo the
nodes located in the area considered worst. This means that if there are any nodes at all in the best
category the test will only run for them and none of the other nodes. The best category’s test is
closer, since it only tests for the distance from the center of gravity. The name of the node in the
most appropriate position is returned to the first procedure that sets the weight on its point
constrain to the cluster in the area of balance, which then can deform to match the nodes better.
Figure 13. Hierarchy of a gravity system with two characters showing two weight nodes.
5 End Result

5.1 System Parts

5.1.1 Weight Nodes

The weight nodes are locators that will be controlling the center of gravity. They can be point or parent constrained to the joints of a character, but then they cannot be parented into the node hierarchy. They can have individual weight values and can be moved, rotated, and scaled to match the user’s needs. The weight nodes have some extra information that is used by the system such as the weight value of the point constrain, as well as a number to identify which of the point constrains on the center of gravity is associated with this weight node. It also has a string indicating what type of node it is, i.e. body hand or foot node. The hand and foot nodes are identical but still have different names to ease the work for the user.

5.1.2 Center of Gravity

The center of gravity is represented by a locator that is point-constrained to each of the weight nodes in the rig. The weight of each node can be edited to the users liking so that the center of gravity is in the middle of the model. The center of gravity is a locked node, meaning that it cannot be moved, scaled, or rotated. A warning sphere is point constrained to the center of gravity with an offset in the Y-axes.

5.1.3 Warning Sphere

The warning sphere is an invisible bright red sphere, located above the center of gravity. Via a render-node network, it will turn its visibility on if the center of gravity is outside the area of balance. The node network is setup to figure out the closest point on the area of balance to the center of gravity and calculating the distance between them in the X- and Z-axes, not counting the Y-axes. If this distance is not less than the value set on the character, the sphere is then visible to warn the animator that the character is not balanced and might need correcting. The sphere is locked but the offset from center of gravity can be edited in all three axes.

5.1.4 Area of Balance

The area of balance is a polygon plane that is deformed to the hand and foot nodes that are below the ground height. Area of balance is locked and cannot be edited by the user, instead it has clusters at each of its four vertexes that changes its shape via point constrains and script jobs.

5.1.5 Ground Height

The ground height is a NURBS plane that indicates the height of the ground in the calculations. The hand and feet nodes that are above the ground height are considered to be in the air and do not affect the area of balance. It is only the Y-axes that are considered in the calculations so the shape and X and Z position are irrelevant. It can although be repositioned and scaled to suit the users preferences.
5.1.6 Characters

Characters are just group nodes and not actually visible in the scene. Inside the character are all the nodes associated with that character; warning sphere, weight nodes, center of gravity and area of balance, as well as the clusters controlling the area of balance. The character nodes hold a vast number of data for the system to find and integrate with the character correctly. The data consists of a list with the weight nodes associated with the character and a string indicating which node is currently selected in the GUI. It also holds variables indicating how far from the center of gravity a node must be to be considered in the calculations and an accuracy setting for the calculation of distance between the area of balance and center of gravity. Finally, it holds a reference to the cluster handles on the area of balance to be used when a new weight node is created.

5.1.7 Gravity Control System

Just similar to characters, the Gravity Control System node is a group node. It acts as the base of the system and holds the list of characters, as well as an indicator to which character is currently active. This group node holds all the scene nodes needed for the system to work.

5.2 Usage

5.2.1 Launching the Script

Place the MEL file in a folder Maya recognizes, i.e. “C:\Documents and Settings\user name\My documents\maya\scripts” or source the script via the script editor inside Maya in the script editor or the Command line execute, the command GravityControl(). This will launch the script and the system will start up. If the system has not been started before in the scene, it will create the system node and its attribute, the warning spheres’ shader and the ground height. Lastly, it will open the gravity control window. It is important to note that the user must run the script once when opening a saved scene that has a gravity system in it where the script jobs that is used to deform the area of balance is setup correctly.

5.2.2 Adding Characters

In the first window of the gravity control system, also known as the character list, the user can add characters by clicking on the “add” button, a dialog box opens and the user is asked for a name. This name will be tested against a number of demands, for the name to be accepted it must be unique for the scene. It must also begin with a letter and not be an empty string.

5.2.3 Handling the Character List

When a character is created, it is added to the list on the left side of the window. The characters in the list can be move up and down in the list and also deleted by using the appropriate buttons. The list can be sorted alphabetically in descending order. The character can be renamed by clicking on the edit button. This will open the weight node list window and at the top there is a field to rename the character.

5.2.4 Adding Weight Nodes

In the weight node list window, weight nodes can be added to the character via the “add” button. This will open another window where the user can specify the name, weight value, and the behavior of the weight node. The name of the weight node goes through the same test as the characters. The only difference is that the weight nodes name must only be unique among the weight nodes on that character. It might be a good idea to begin with the weight nodes for the feet.
5.2.5 Weight Node Behaviors

There are three behaviors to choose from but the hand and foot behaviors are identical. They are split up so the user can easily identify which nodes belong to the characters hands and which belongs to its feet. These nodes will have an effect on the area of balance when moved below the ground height. This is not true for body type nodes, however all three kinds of nodes will have a constant effect on center of gravity and the power of that effect will be determined by the weight value.

5.2.6 Handling the Weight Node List

Just like the character list the user can move, delete and sort the weight nodes with the buttons on the right hand side of the window. Unlike the character list the weight node list contains more than just the name of the node. It also has the weight and behavior type of the node.

5.2.7 Using the System

Once the user has set up the character, created the nodes, and positioned them on the character as needed, the weight nodes should be parent-constrained to the joints of the skeleton. Then, the process of setting up the weight can begin in the weight node list window where the user can use the weight slider and weight field to set up the weight. A good work flow might be to have the character curl up and then the center of gravity should be at the center of mass. When the weight is setup to the user preference, then the system can be used. On the characters, the user can set the accuracy for the calculation between the area of balance and the center of gravity. The user can also find settings for how far from the center of gravity a node must be to be calculated into the deformation of the area of balance.

By using the attributes Min Node Distance the user can set the distance different for the four corners of the center of gravity for the selected character. The letter combinations to the left of the attribute name indicate which corner is affected. P stands for positive and N for negative, Z for Z-axes and X for X-axes. I.e. the PZNX means the corner in the positive Z and negative X direction from the center of gravity. Therefore, any corner node with a higher Z value and lower X value than the center of gravity and whose distance from the center of gravity is lower than the character’s Min Node Distance PZNX will not be considered in the calculations. Another control the user has on this calculation is moving the ground height object up or down in Y to define when a weight node is to be used in the calculation or when it is to be considered to be elevated off of the ground.

5.3 Troubleshooting and Known Error

5.3.1 Area of Balance Constrains Calculations

The calculations do not recognize the number of clusters constrained to one weight node, which might lead to unexpected result for the user. This will be represented as three clusters on one weight node and one cluster points out to another weight node, because of the weight shift in the character. If the weight shifts towards the foot with one cluster, another will soon jump over, because the distance from the center of gravity plays a high part in the calculations. This will simulate very well as the character shifts its weight from side to side, because the weight gets concentrated on the leg closest to the body.

5.3.2 Area of Balance and Center of Gravity’s Distance Calculations

The calculation network that is used to check the center of gravity against the area of balance is sometimes off by about 0.3 units. I do not know why this occurs or how to prevent it. That is why
an accuracy setting was added on the characters. This posed another problem since it did not recognize 0.000 to be equal to 0.000, therefore the accuracy is set by default to 0.001.

5.4 Discussion

5.4.1 GUI

I was very satisfied with how the GUI turned out. The only item that I wished to have changed is in the window where the user creates new weight nodes; it is necessary to press the “ok” button to make the script continue. If the user only had to press enter on the keyboard to accept the inputted data such as when creating a new character, it would have added to the functionality of the script.

5.4.2 System

I am content with how the system turned out especially since I managed to obtain a working system in the end; still I wished I would have had the time to add the other features that I wanted to have in the system. As it turned out, I did not even have the time to accomplish enough testing to conclude if it was possible to make a complete setup as I envisioned in the planning stage of this thesis project.

6 Future Researches

6.1 Rotation Around Center of Gravity and Hand Type Weight Nodes

The test I did gave two results. The first was that the skeleton did not rotate around the center of gravity but only took the center of gravity’s rotation and rotated the same amount around itself. The second was a rotation around the center of gravity but when rotating the skeleton around this center the weight nodes constrained to the skeleton moved and shifted the center of gravity. This gave an uncontrollable double translation and the character disappeared in the distance.

A method of working around this might be to parent constrain the root joint of the skeleton to a second node which pivot point is always moved to the position of the center of gravity. I did not have enough time to do tests on the possibility of rotating the rig around its hands either, but I believe that it will work if the root joint is parent constrained to a separate node which is moved to the position of the hand, similar to that mentioned above. Via the weight values on the parent constrains the effect can be turned on and off, however only one node may have the control over the skeleton’s rotation at one time and this will be controlled by the weight values.

6.2 Dynamic Shape Changes for the Area of Balance

Something, I reflected on looking back in this thesis project was that if the area of balance, instead of having four points, it could have a minimum of four to an infinite number of points. In place of moving the clusters of the area of balance to the four best corner nodes on the character’s weight nodes, every weight node under the ground height gains one or two points. This means that the script jobs would have to add and delete points from the area of balance depending on the number of weight nodes that is located under the ground height.

Having a setup similar to this would allow more accurate results from characters such as spiders with more than four legs. One problem however with this method is that not all nodes can gain points on the area of balance. If a six-legged character uses five legs to shape the area of balance to a pentagon and the sixth leg is placed in the middle, only the outer five would be the ones to deform the area of balance and the last leg must be excluded from the algorithm.
6.3 Auto Calculate the Weight Values

To make this area work, I think a good approach would be to test each weight node against all vertexes or control vertices in the mesh and compare the results among the weight nodes. This will show which locator each point is closest to and then the combined distance from the weight node to each of its associated points will be that weight nodes mass. By comparing the mass of each locator, the system can then determine the appropriate weight for each weight node compared to the other weight nodes. This can simply be created by dividing the mass of each weight node with the highest mass any node has on that character.
References


[8.a] Procedures pp.96
[8.b] Expressions pp.245


[12.b] Procedures pp.169