Creation of a Maya Tool Based on the Paper Character Articulation through Profile Curves

By

Vanessa Stotz s5602665

MSc Computer Animation and Visual Effects
at Bournemouth University, UK

August 2024

ABSTRACT

Rigging still relies heavily on corrective blend shapes and skeletal deformation, which makes it a cumbersome process. This work presents the development of a tool in Autodesk Maya based on the new deformation technique for character articulation introduced by Pixar Animation Studios. The tool provides a framework for drawing and editing the curves, benefiting from Maya's native functions. Resulting in an extra layer added to the rig, enhancing the skeletal deformation.

Contents

List of Figures

Introduction

The animation of a character stands and falls with the quality of the rig. The animation should be able to show a wide range of motions and emotions. Therefore, the character must be pushed into extreme poses while maintaining a strong silhouette. With good skin deformation, a rig can easily provide all of this. It should not only be able to not break during motions but also maintain the volume of the character. Although there are established methods of skinning, corrective blend shapes are still required. They are often sculpted to enhance specific poses. Another problem with these techniques is their dependency on the mesh. Therefore, the skeletal deformation needs to be rebuilt after a mesh, especially its resolution, is changed.

A new approach is found in the paper *Character Articulation through Profile Curves* published by Goes et al. in 2022. This approach benefits from the independence of the rig from the mesh, as it is profiled by Bezier splines traced along the mesh. This separates the articulation controls from the underlying surface. Therefore, the rig remains unaffected by changes to the model. Since the skeleton is only connected to the curve points and the deformation of the mesh can be re-evaluated at any time. It also provides the animator more precise control over the shape, as they can directly deform the object. Curvenets further reduce the number of control points that need to be added to the rig.

This work is a continuation of the project submitted in Animation Software Engineering and CGI Techniques in term one. The aim of the work is to provide a deeper and more detailed understanding of the presented paper and a solution for implementing its algorithm as a tool in Autodesk Maya, using exclusively the built-in tools and functions.

The work first provides a detailed understanding of the presented paper and other relevant works; it then explores technical algorithms necessary for discussing the implementation in Maya. It concludes with a user test of the tool and possible further improvements.

Previous Work

Different deformation techniques are used to skin a character to a rig. The chosen technique depends on the performance requirements and complexity of the animation. Common deformation techniques include skeleton-subspace deformation, pose space deformation, and free-form deformation. They can be used either individually or in combination. (McLaughlin et al. 2011)

2.1 Skeleton-Subspace Deformation

Skeleton-subspace deformation (SSD) is the standardised technique for industry software. Each vertex of a deformed surface lies in a subspace defined by the rigid transformation of some control mechanisms, mostly joints. Each vertex can be influenced by a series of joints Ω, and its new position is notated with:

$$\bar{p} = \sum w_k L_k^\delta L_k^{0^{-1}} L_P^0 p \tag{2.1}$$

Where L_p^0 is the transformation of point P from the local space of the joint to world space. L_k^0 defines the transformation of the stationary joint in world space, while L_k^δ defines the transformation of the moving joint. The deformation can be controlled by the weight factor w_k.(Lewis et al. 2000, Liu et al. 2003)

Besides its versatility, the deformation is limited to its defined subspace, leading to unpredictable results. Another limitation is the lack of direct shape manipulation.(Lewis et al. 2000)

2.2 Pose Space Deformation

Pose space deformation (PSD) can often be used as an addition to Skeleton-subspace deformation as it allows direct manipulation of the shape, thus providing a solution to the limitations of SSD (McLaughlin et al. 2011). PSD is a “purely kinematic approach to deformation” and therefore free “from underlying forces."(Lewis et al. 2000, p. 168). The new shape is defined by the pose space, which is defined by pose controls such as joints or abstract manipulators like UI controls. $\vec{\delta}$ for each vertex are stored in the pose space. The new position of the vertex point is determined by

$$p + \vec{\delta} \tag{2.2}$$

where $\vec{\delta}$ equals the radial basis function for scattered interpolation in 2.3. (Lewis et al. 2000)

$$\hat{\delta}(x) = \sum_{k}^{N} w_k \phi(\|x - x_k\|) \tag{2.3}$$

Rather than being dependent on the vertex point, the radial basis function only depends on the distance between the initial and deformed positions (Döring 2017). Each vertex can have more than one $\vec{\delta}$ (Lewis et al. 2000). Although this method allows direct manipulation of the shape, it is a time- and data-intensive method, as each vertex must be manipulated individually.

2.3 Free-Form Deformation

Free-form deformation (FFD) deforms an object directly using a coordinate grid and therefore eliminates the need for a control mechanism (Parent 2008). The edges of the grid are often Bezier curves, defining the faces as Bezier surface patches (Sederberg and Parry 1986). Free-form deformation creates a “local coordinate system that encases the area of the object to be distorted” (Parent 2008, p. 143). The local coordinate system is described by a set of three vectors (S, T and U). The user can then manipulate the grid directly by moving the control points of the Bezier curves, resulting in a relocation of the surface vertices (Parent 2008, Sederberg and Parry 1986). Each vertex has a world coordinate (x, y and z), but is also registered in the free-form deformation by having relative coordinates (s, t and u) to it. The local coordinates are defined as

$$x = x_0 + sS + tT + uU \tag{2.4}$$

With its vectors being

$$s = \frac{T \times U(X - X_0}{T \times US}, t = \frac{S \times U(X - X_0}{S \times UT}, u = \frac{S \times T(X - X_0}{S \times TU} \tag{2.5}$$

Where $0 < s < 1, 0 < t < 1, 0 < u < 1$, if the point lies within the FFD. The control points on the lattice in global space are defined by

$$P_{ijk} = X_0 + \frac{i}{l}S + \frac{j}{m}T + \frac{k}{m}U \tag{2.6}$$

Where (S, T, U) represents the unmodified local coordinate grid. Each direction of the local coordinate system can have an individual number of control points. This is taken into account with (l, m, n). “The deformation is specified by moving P_{ijk} from their undisplaced, lattical position” (Sederberg and Parry 1986, p. 153). The preferred interpolation is done “by a trivariate tensor product Bernstein polynomial" (Sederberg and Parry 1986, p. 153). The deformed position of any point within the lattice can be calculated using

$$x_{ffd} = \sum_{i=0}^{l} \binom{l}{i} (l - s)^{l-i} s^i \left(\sum_{j=0}^{m} \binom{m}{j} \left((1 - t)^{m-jtj} \left(\sum_{k=0}^{n} \binom{n}{k} (l - u)^{n-k} u^k P_{ijk} \right) \right) \right) \tag{2.7}$$

To compute this equation, the results of (2.5) and (2.6) are needed(Parent 2008, Sederberg and Parry 1986).

Calculating a free-form deformation is rather simple and allows easy and smooth control of a character's movement based only on the positions of the control points. However, it is not suitable for detailed articulation as its rigid nature contrasts with the predominantly organic nature of a character. Even combining multiple grids to better replicate the character is still not enough as they may not correspond to the meaningful parts of the shapes the user wants to change. Free-form deformation can also easily destroy the character if not carefully designed. (Joshi et al. 2007, Gal et al. 2009)

2.4 Wires

Singh and Fiume (1998) and Gal et al. (2009) introduce the idea that any object can be characterised and deformed by a small set of curves or wires. To preserve the essence of the shape, wires should preferably be constructed along "sharp mesh edges" or "intersections between smooth surfaces" (Gal et al. 2009, p. 33:4). This leads to a finer deformation of the surface while still being independent of its complexity. A wire is defined as a "curve whose manipulation deforms the surface of an associated object near the curve" (Singh and Fiume 1998, p. 1) and is defined with $\langle W, R, s, r, f \rangle$.

Where W is the wire curve and R is the reference curve. The scalar s defines the radial scale around the curve, while r defines the radial influence around the curve. $f : R^+ \to [0, 1]$ is the density function, with it normally being C^1 and is implemented with $f(x) = (x^2 - 1)^2, x \in [0, 1]$.

Wire deformation is calculated for each point P_0 of an object. Its deformation depends on the difference between W and R and is deformed only when it is within the volume of radius r with respect to the reference curve and 2.8 > 0.

$$F(P, R) = f\left(\frac{\|P - C(p_c)\|}{r}\right) \tag{2.8}$$

P_c defines the parameter value that minimizes the Euclidean distance between a point on the object and a curve point on the reference curve. So if $\|P - C(p_c)\| = 0|$, the point lies exactly on the reference curve and is completely influenced by the deformation. (Singh and Fiume 1998) Gal et al. (2009) improves on Singh and Fiume (1998) implementation by adding some intelligence to the curve, so that they are aware of other wires affecting the same shape. The relationships between them are established and maintained during deformation.

2.5 Curvenet and Cut-Mesh Algorithm

The drawbacks of the wire frameworks is the assumption that the mesh edges are connected to the control curves, limiting the setup of the wires to the specified mesh resolution (De Goes et al. 2022). De Goes et al. (2022) eliminates this problem by introducing a new mesh-cutting algorithm that detaches the curve networks from the mesh edges. The new method is based on the Cartesian cut-cell method, and the resulting mesh holds the information of the vertices of the input mesh and the curve samples while splitting the surface faces into smaller polygons. These polygons "can be non-planar, non-convex, or even include cracks" (De Goes et al. 2022, p. 139:6). Cracks are possible due to the permission of the cut-edges branching out of the faces. Before initialising the cut-mesh every profile curve is converted into a curvenet. A curvenet is a polyline with evenly spaced points. The cut-mesh is then initialised by a copy of the input mesh and “the neutral position $\breve{q}$ for each curvenet sample” is sampled onto “the closest point $\breve{p}$ on the surface mesh” (De Goes et al. 2022, p. 139:7). Each sample is categorised as either a vertex, edge or face sample. There are different combinations to connect the curvenet segments to the cut-mesh and cut the polygons accordingly. They can be seen in Fig. 2.1. To update the cut-mesh connectivity, the newly created faces must be identified. Therefore, a tangent space for every cut-vertex is identified. The algorithm benefits from the fact that the cut-mesh is a direct copy of the input mesh; therefore, the tangent space is transferred from the underlying point or polygon. Due to its discrete representation of the input mesh, it facilitates the deformation of the input mesh with the curvenet.

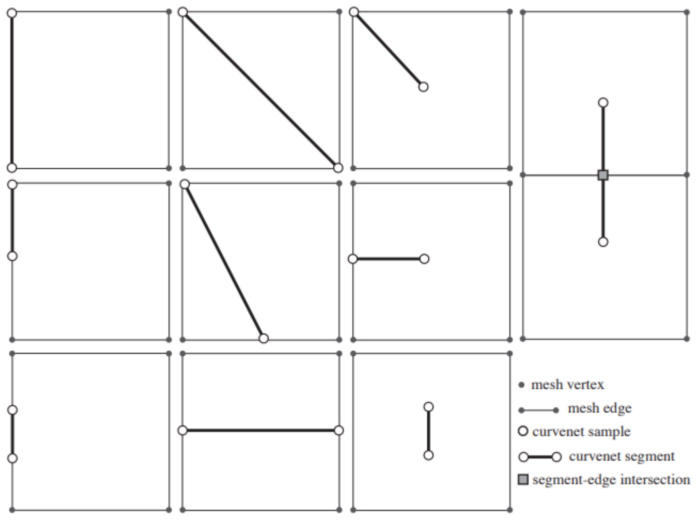

Figure 2.1: Group Interface (De Goes et al. 2022)

The interpolation is divided into two groups. The first is represented by any cut-vertex that coincides with a mesh vertex, while the second is represented by any cut-vertex that corresponds to a curvenet segment or an intersection between a curvenet segment and a mesh edge. Both groups define a solution space with

$$\phi_h = V\phi_v + C\phi_c \tag{2.9}$$

$V\phi_v$ represents points from the first group and $C\phi_c$ represents points from the second group, with V and C being matrices and ϕ_v and ϕ_c being vectors. After the discretization of the cut mesh has been defined, the final shape can be calculated. The deformation is defined using a two-step optimization. First, the "deformation gradients from the curvenet segments to the mesh vertices" (De Goes et al. 2022, p.139:8) are interpolated with positions $\breve{q}$ and q from the undeformed and deformed curvenets. The deformation gradient is described with the help of

$$min_{f_v} E_D(V\mathrm{f}_v + C\mathrm{f}_c) \tag{2.10}$$

and optimised with the Dirichlet energy

$$E_D(\phi_h] = \phi_h^t L_h \phi_h \tag{2.11}$$

for a more harmonious interpolation. The deformation gradient for each cut-face is stored in a matrix y. The final step is to compute the new "vertex positions that best match the interpolated deformation gradients while preserving surface details and reproducing the target curvenet samples"(De Goes et al. 2022, p. 139:8). Therefore, the new position approximates the values of y while still enforcing the location of the sample of the posed curvenet. Although De Goes et al. (2022) introduce a new method for free-form deformation, they are not eliminating the use of SSD, as the control points of the curvenet can be connected to an underlying control mechanism. Therefore, it can be said that curvenets are just a new method of skinning a character to a skeleton, which provides more flexibility for animators.

Technical Background

3.1 Bezier Splines

The paper defines the Profile Curves as cubic Bezier splines (De Goes et al., 2022). Bezier splines are often used to represent curves in computer graphics as “they are easy to control, have a number of useful properties and there are very efficient algorithms for working with them” (Gleicher 2016, p. 386). The curve is controlled by $d+1$ control points, where d is the degree of the curve. The control points form a convex hull to which the curve is bound. The curve interpolates through its start and end points while “the shape is directly influenced by the other points” (Gleicher 2016, p.385). This leads to the definition that the spline is *affine invariant* and transforming the control points “is the same as performing those operations on the curve itself.” (Gleicher 2016, p. 388), which makes it such a powerful curve for computer graphics.

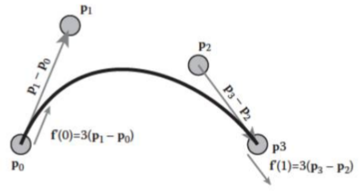

Figure 3.1: Bezier Spline (Gleicher 2017)

Therefore, a cubic Bezier spline has a degree of three and four control points. An example curve can is seen in Fig. (3.1) And can be written mathematically as :

$$\mathbf{f}(u) = \sum_{i=0}^{d} b_{i,3}\mathbf{p}_i \tag{3.1}$$

Where the blending functions of degree 3 for the Bezier are defined as

$$\begin{aligned} b_{0,3} &= (1-u)^3, \\ b_{1,3} &= 3u(1-u)^2, \\ b_{2,3} &= 3u^2(1-u), \\ b_{3,3} &= u^3, \end{aligned} \tag{3.2}$$

Which this leads to

$$\mathbf{f}(u) = (1 - 3u + 3u^2 - u^3)\mathbf{p}_0 + (3u - 6u^2 + 3u3)\mathbf{p}_1 + (3u^2 - 3u^3)\mathbf{p}_2 + (u^3)\mathbf{p}_3 \tag{3.3}$$

3.2 Calculate Control Points

As just seen, a Bezier curve is defined by four control points. Nevertheless, the tool only requires the user to define the start and end point of the curve. The inner control points are calculated by the tool. The paper of De Goes et al. (2022, p. 139:5) describes the interior control points as “points forming tangent handles” that are initialised “perpendicular to the surface normal”. A possible algorithm was already found during the project work in the course Animation Software Engineering and is described in more detail here. This approach is a combination of a plane in Hessian normal form and a line perpendicular to it. Hessian normal form is another way of writing a plane equation and is often used to determine the distance between a point and a plane.

$$E : \vec{n_0}(\vec{x} - \vec{p_0}) = 0 \tag{3.4}$$

Each point on the plane satisfies the requirement of being an inner control point, as it is created with the outer control point and its normal vector. The normal vector is the normal vector of the face closest to the control point. However, to create the convex hull in the correct direction, the intersection point between the plane and a line passing through the other outer control point must be found. This line is also perpendicular to the plane and is defined by using the same normal vector and the other outer control point

$$L : \vec{x} = \vec{n} + t\vec{p_1} \tag{3.5}$$

The intersection point is calculated by setting up the plane equation in its coordinate representation.

$$E : n_1x_1 + n_2x_2 + n_3x_3 = n_1p_1 + n_2p_2 + n_3p_3 \tag{3.6}$$

The point variables of the plane equation are substituted with those of the line and the new equation is then transposed to parameter t. With the resulted parameter, the final intersection point can be determined by solving the equation of the line.

$$\begin{aligned}
&L : x_1 = n_1 + tp_{11} \\
&L : x_2 = n_2 + tp_{12} \\
&L : x_3 = n_3 + tp_{13} \\
&E : n_1(n_1 + tp_{11}) + n_2(n_2 + p_{12}) + n_3(n_3 + p_{13}) = n_1p_{01} + n_2p_{12} + n_3p_{13}
\end{aligned} \tag{3.7}$$

To ensure that the curve is drawn close to the surface, the distance vector between the intersection point and the control point is determined and normalised. After dividing the vector by a third of the length between the start and end point, the final inner control point is determined by adding it to the control point. A visualisation of the calculation can be seen in Fig. (3.2).

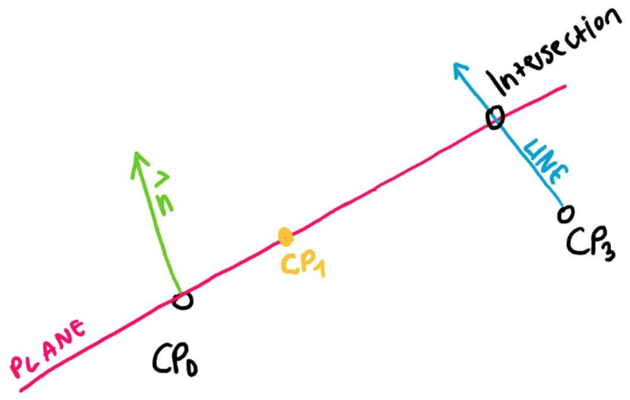

Figure 3.2: visualisation of calculating control point (personal collection 2024)

Implementation

The tool described in this work shows an approach to implementing the new algorithm introduced by Goes et al. (2022) in Maya. The goal of the tool is to provide a different method of skinning the character to the rig by replacing the skeletal subspace deformation with a wire deformation. Therefore, it should eliminate the need for corrective blend shapes and provide the animator more freedom to play with and improve the shape of the mesh. The tool offers a straightforward way to create a curvenet, provides multiple editing options to optimize the curves, and offers an easy way to add deformation to the mesh and rig.

The tool is written entirely in Python and uses only Maya's native functions to create objects within it. The goal is to create a tool that is easy to install and use. The whole source code is distributed across multiple functions. Important datatypes in the script are a list for all the different curvenets in the scene and a dictionary containing the individual information of a curvenet. The curvenets information are its name, the name of the mesh and a list of every profile curve with the name of the Bezier and its two tangents.

The tool is divided into different steps, which are reflected in the structure of the tool. This chapter provides further insight into the user interface and the approach, as well as explanations of various functions. It then concludes with a user test leading to possible further improvements.

4.1 User Interface

The tool's layout can be divided into a group and two tabs. The group (Fig. 4.1) provides information about the current selected curvenet and its mesh. In addition, it stores the functions for adding and deleting a curvenet and selecting a new mesh.

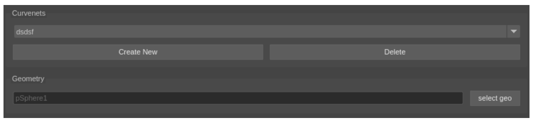

Figure 4.1: Group Interface (personal collection 2024)

The first tab (Fig. 4.2) contains all the elements required to create and edit the curvenet. It is divided into three groups. The first contains the functions for drawing the curves, the second contains the functions for editing the curves and the last allows editing the control points. Each of these groups must be activated by checking the checkbox. This adds a specific environment to the viewport and is simplifying the process. The user must actively uncheck the box before they can do anything else.

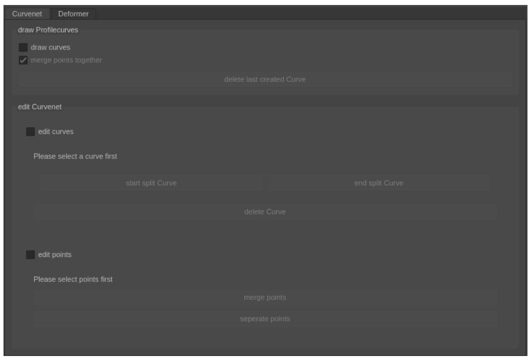

Figure 4.2: Tab 01 Interface (personal collection 2024)

The second tab (Fig. 4.3) is exclusively for connecting the curvenet to the mesh and the rig. The upper part contains the function of adding and deleting selected points to and from the skeleton, where the lower part adds or deletes the curvenet as a deformer.

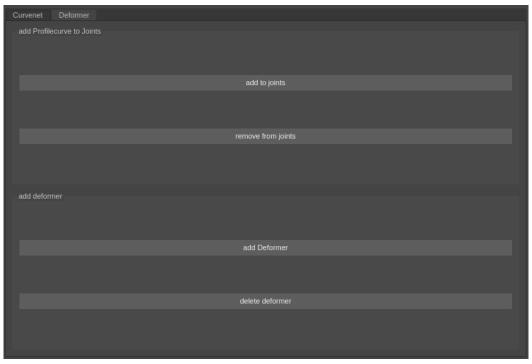

Figure 4.3: Tab 01 Interface (personal collection 2024)

4.2 Workflow

4.2.1 Creating Curvenets

Multiple curvenets can exist in one file, allowing different meshes of a character to be edited separately. All existing curvenets are displayed in the user interface. Each time a new curvenet is created, the curvenet data is reset and the user-selected name is added. The input name acts as a prefix for all objects and other associated data, such as a series of empty groups that are created after the new curvenet is initialised. This helps sort all the data of the curvenet.

If a user wishes to delete a curvenet, the associated groups and internal data are also deleted. When changing the curvenet, the data of this curvenet is loaded from the scene file. A more detailed description of how the data is stored and read can be found in chapter 4.10 .

4.2.2 Connecting Geometry

To draw the curves and later add a deformer, the tool relies on a mesh. All the users must do is select the mesh and confirm its selection. The name of the mesh is then stored in the curvenet dictionary and can be retrieved at any time. For a better overview, the name is also displayed in the user interface.

4.2.3 Drawing Curves and calculating Profile Curves

Drawing curves is the core of this tool, as this step determines the shape of the profile curves that deform the mesh. Therefore, finding the best approach drawing the curves was a main focus . The first step was to figure out how to draw the curves onto the mesh. The challenge is to let the user draw only the start and end point of the curve while the other points are calculated by the tool. The first thought was to place locators at the positions of these points, but that seemed counter intuitive. A natural approach would be to simply draw a curve, which is then converted into the required curve shape. The EPCurveTool in Maya allows the user to draw a curve with a defined degree in the viewport. The length of the curve is set by the user when exiting the tool. Since the curve should only have two points and to prevent the user for having to manually enter and exit the curve tool, an environment needs to be created for drawing.

Drawing Environment

The environment allows the user to draw a linear line on the surface. To ensure that the points are on the surface, the mesh is made live inside the environment. The mesh is further set to the x-ray shading mode to make drawing easier. To ensure that a new curve is drawn after the previous one has two points, it is necessary to monitor the number of clicks or the index number of the control point. After drawing a curve point, Maya changes its selection to the point. So a possible condition is:

Algorithm 1 Drawing Condition 01

```
if selected control point == 1 then
    Quit the tool
    Calculate the new curve
    Re-enter the tool
end if
```

The missing part in this condition is a way to call the condition every time a point is drawn. The first approach was to use a filter event which tracks whether the left mouse button is clicked while the mouse is in the maya viewport. This filter is activated as soon the environment is entered. This changes the algorithm to:

Algorithm 2 Drawing Condition 02

```
if leftMouseButton == pressed then
    if selected control point == 1 then
        Quit the tool
        Calculate the new curve
        Re-enter the tool
    end if
end if
```

However, this resulted in an incorrect creation of the curves, as curves were not created after the second point was drawn, but rather after the third point. Therefore the first point of the next curve was drawn, which was not an intuitive solution. The problem is that the selection does not change until the mouse button is released. So the condition always received the previous information. A natural solution would have been to change the first if-condition in Algorithm 2 to “left mouse button == released”. Somehow, this caused unpredictable behaviour in Maya and did not produce the expected results. The final solution was found using a scriptJob. A scriptJob is an option to run a function every time an event happens in Maya. One of the valid events is SelectionChanged. Therefore, the job is created every time the environment is entered and deleted as soon it is exited. The algorithm used is Algorithm 1.

Calculating the Profile Curve

After a curve has been drawn, the new curve must be calculated. To do this, the curve points of the drawn curve must be extracted, and the missing ones calculated, as well as the control handles for the curve points. Before calculating the inner control points using the algorithm described in Chapter 3.2, the existing points are checked to see if a point is within the threshold, but only if the Merge Points option was enabled while drawing. The threshold is the radius of the sphere to be created as a control handle. The radius is determined using Algorithm 5 and checking for a closest point is done using Algorithm 9 (Appendix). If a point is found, within this radius, its position is taken.

After all points are calculated, a Bezier curve and two linear curves are created. The linear curves connect an outer control point to its nearest inner control point. This is for visual purposes only. The Bezier curve and tangent control points are controlled by a cluster, which is a child of the control handle.

To add the cluster and sphere handle, the script runs over each control point of the curve and adds it and the tangent point with the same position to a list. If the point is an outer control point and the merge points option is enabled, the script first checks whether a cluster already exists. This is done using Algorithm 11 (Appendix).

If the merge point option is disabled or the point is an inner control point, a new cluster and its control handle are created. The cluster is created using Algorithm 3 while the sphere handle is created with Algorithm 4 and the size is determined using Algorithm 5. The size of the control handles should not be too large if the object is small but still large enough for large objects. Therefore, the size is determined based on a ratio between the area of the object bounding box and the area of the underlying face. To prevent the sphere from becoming too small, the minimum is set to 0.1. An example of the calculated curve can be seen in Fig. (4.4).

Algorithm 3 Create New Cluster

```
select points to add to the cluster
create a new cluster
create a control handle
parentConstraint the cluster to the control handle
hide the cluster
```

Algorithm 4 Create Control Handle

```
get the radius of the sphere
create the sphere and assign a shader to it
lock and hide rotation and scale and visbility
```

Algorithm 5 Calculate Sphere Size

```
area = area of the bounding box of the mesh
areaFace = area of the closest face mesh
areaRatio = areaFace/area
radius = max(abs(math.sqrt(abs(areaRatio)/4 * math.pi))*0.5, 0.1)
```

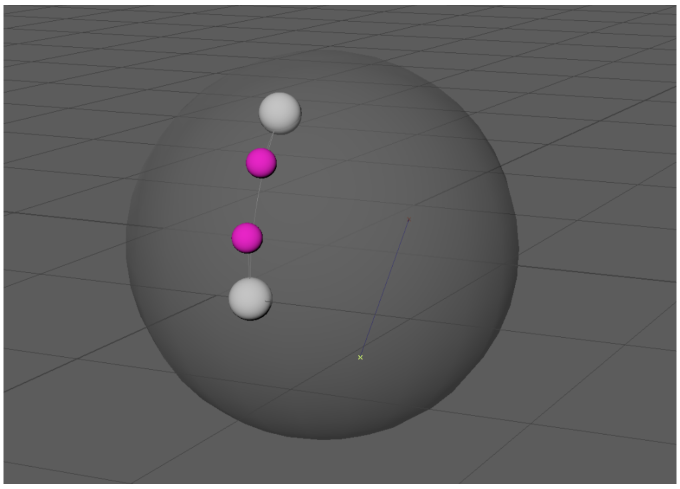

Figure 4.4: Drawn Curve in draw environment (personal collection 2024)

4.2.4 Editing Profile Curves

There are two ways of editing the curves after creating them. The first method focuses more on the actual curve while the other one focuses on the control points.

Editing Curves

In this environment the focus is solely on the Bezier curves. Therefore, the handles are hidden and the mesh is set to x ray shading (Fig. 4.5). The curves can either be deleted or split into two new curves. For both options a curve must first be selected. Since its name is needed to perform the corresponding functions.

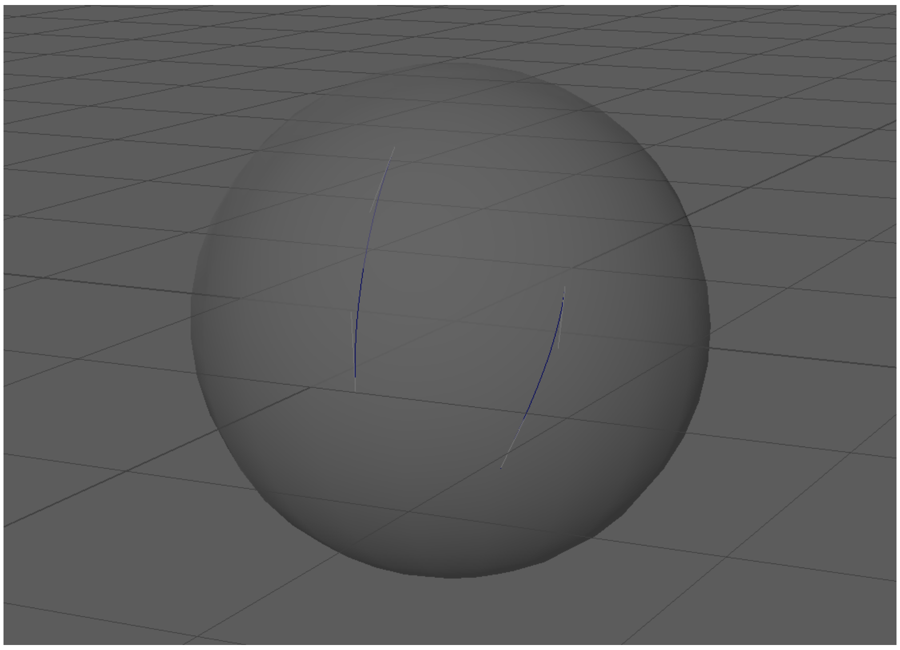

Figure 4.5: edit curve environment (personal collection 2024)

Due to the way the splitting algorithm works, the user must enter and exit the process. Entering the process works as follows:

Algorithm 6 Start Split Curve

```
create a locator
create a cube
create nearestPointOnCurveNode
and set the locator position to .inPosition
and connect .result.poistion to the cubes translation
select the locator
```

The user then moves the locator, while the cube indicates the location of the split (Fig. 4.6). When the user is satisfied with the position the process is ended by:

Algorithm 7 End Split Curve

```
newPoint = result position of nearestPointOfCurveNode
get the start and endpoint if the selected curve
ensure that newPoint is on the mesh, otherwise get the closestPoint on the mesh
delete the selected curve
create two new curves with
startPoint = startpoint and endpoint = newPoint
startPoint = newPoint and endpoint = endpoint
delete locator
delete cube
delete nearestPointOnCurveNode
```

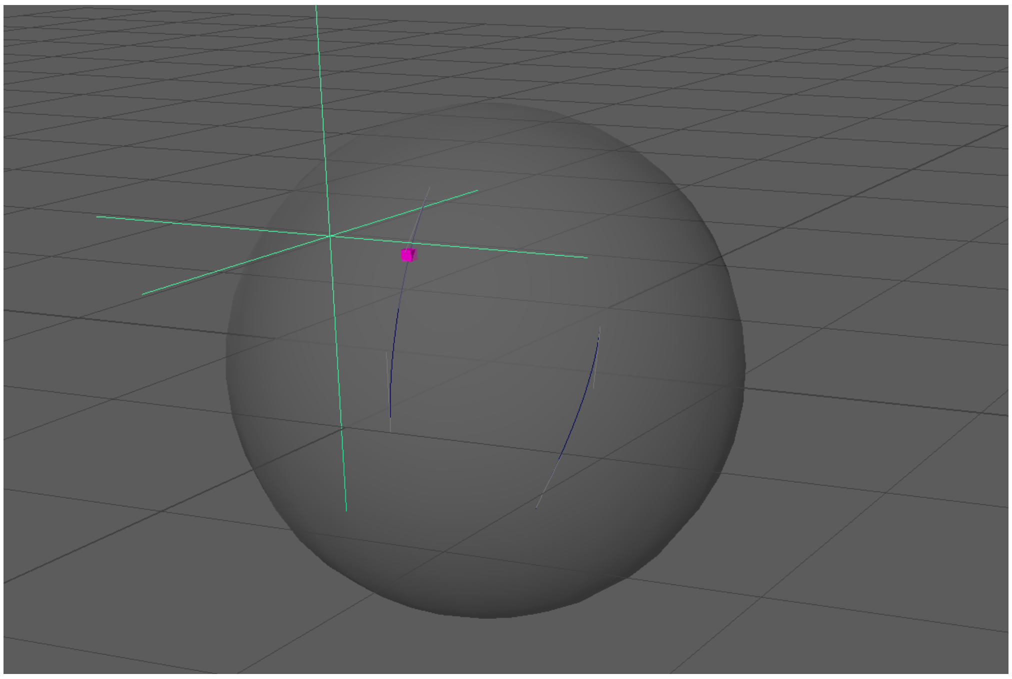
Figure 4.6: split curve with created locator and cube (personal collection 2024)

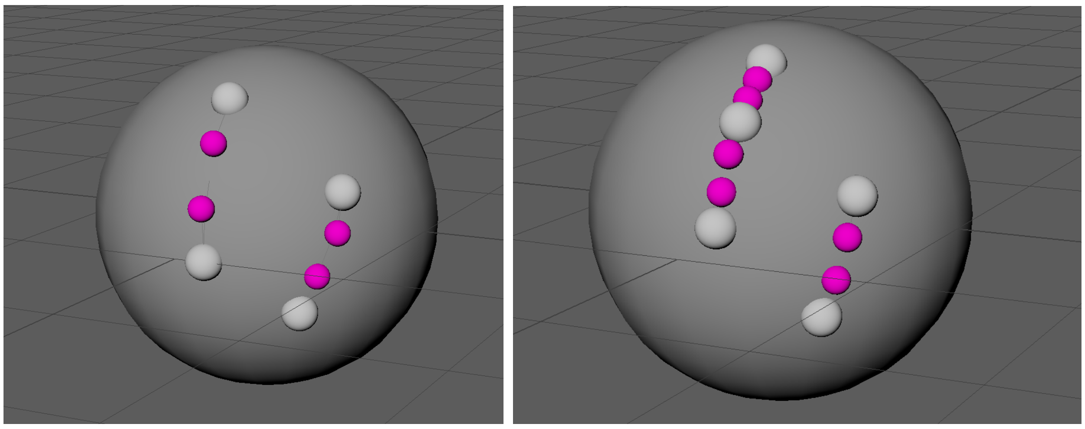

Figure 4.7: left: curve before splitting, right:curve after splitting (personal collection 2024)

To delete the curves the user simply presses the delete button after selecting the curve and Algorithm 10 (Appendix) is executed:

Editing Points

All curve handles can be edited and moved throughout the whole process. However, this environment provides an advanced option for editing the handles that move the outer control points and therefore those on the mesh. To allow better focus on these handles, the tangent handles are templated and cannot be edited in this environment. The advanced option is to make the mesh live, meaning the handles only move along the surface. This environment also offers the possibility of merging and separating handles. One or more handles must first be selected, to execute these processes . Control points of curves are usually controlled by shared handles, if they are within a certain threshold. If the user decides to connect curves together, the the process gets executed with the Merge Points Button, while Separate Points separates the curves from connected handles so they can be moved individually (Fig. 4.8). The process of merging points can be found in Algorithm 12 (Appendix), while Algorithm 13 (Appendix) shows the algorithm of separating points.

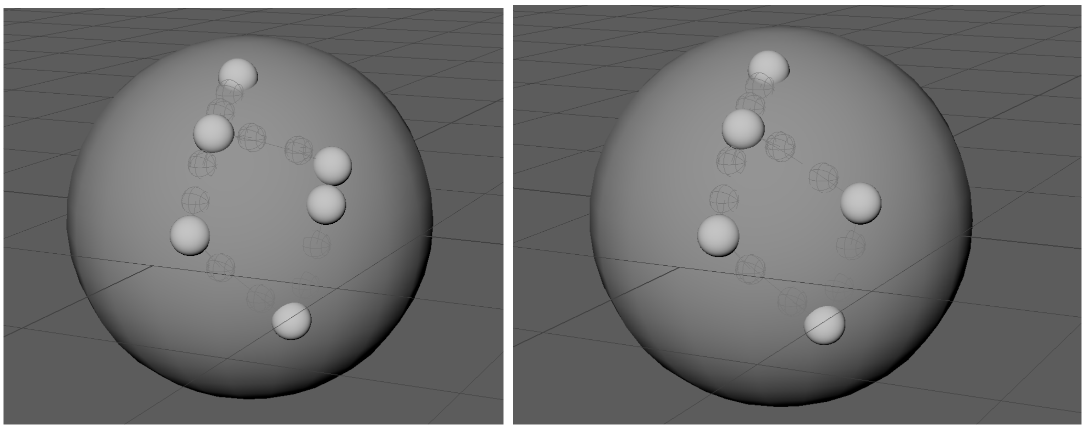

Figure 4.8: left: seperated points, right: merged points (personal collection 2024)

4.2.5 Connecting Curves and Mesh

In order to add deformation to the mesh, the curvenet must be added as a deformer. This is done using the Maya wire deformer. The user does this simply by clicking “Add Deformer”. Algorithm 14 (Appendix) runs under the hood. As can be seen in Chapter 2.4 a wire deformer requires wire curves and reference curves. Therefore, a copy of all curves is created. To provide greater control over the deformer, a custom attribute is added to the main curvenet group. This attribute controls the drop-off distance of the deformer, which is corresponds to the radius value of the wire deformer.

Before adding the deformer, the transformation of all control handles is frozen. It is important to have a zero transformation for each controller so that the animator can reset the sphere to its original position at any time, if necessary.

If the user chooses to delete the deformer again, each transformation of the sphere handles is reset to its zero position to ensure that the deformer and all copied curves and added attributes are deleted correctly. The following algortihm is run:

Algorithm 8 Delete Deformer

```
get the child groups of the mainCurvenetGroup
if baseCurvesGrp in child groups then
    reset the position of the sphere handles to zero
    get the input connections of the mesh
    and delete the wireDef
    delete the dropOffDistanceControl attribute
    and delete the baseCurvesGrp and baseClusterGroup
end if
```

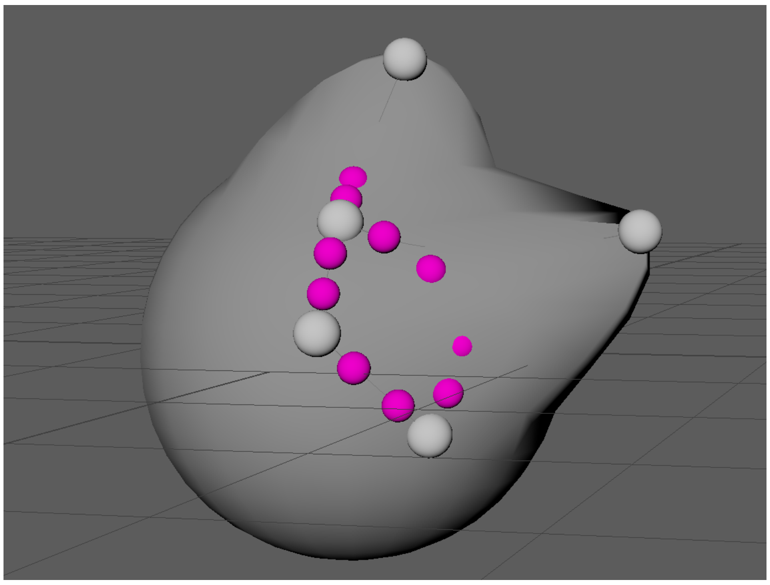

Figure 4.9: deformed object (personal collection 2024)

4.2.6 Connecting Curves and Rig

De Goes et al. (2022) describe that the handles of each profile curve can be added to the rig using well-known approaches such as skinning. The skinning method does not work, as the connection between the sphere and the handles relies on real transformation and otherwise would be lost. Therefore, the handles are added to the joints with a parent constraint. This led to the realisation that each sphere handle must be within an offset group, otherwise the animator would lose the ability to edit the handles directly. To connect the control points to the rig, each selected sphere handle is connected to each selected joint. Deleting the influence simply deletes the connection from the selected control points.

4.2.7 Saving Curvenet Data

It is important that users are able to manipulate the curvenet after reopening a scene. Therefore the curvenet data is stored in the Maya scene file. The stored data is the list of all existing curvenets and the dictionary of each curvenet. The data is updated after every change made in the scene. For example, adding a new curvenet to the list, selecting a mesh, or drawing and editing the curves. Only the curvenet list and the data of the currently selected curvenet are loaded in the script, as well as the names of the groups. This ensures that the code does not have to deal with a lot of data at a time.

There are various methods to save data in a Maya file. The methods are *optionVar*, *fileInfo* or within a *custom node*. It was first tried to save the data using *optionVar*, as this allows different data types such as dictionaries, lists, and strings. However, *optionVar* does not save the data within the current file, but as a local variable that can be accessed by any Maya file in the same environment. This would have lead to too much data and pose the risk of name collisions and therefore incorrect behaviour of the tool.

Therefore, the method of *fileInfo* was chosen to save data in Maya, as a custom node could be accidentally deleted by the user. File information is stored in the source code of the file, regardless of whether it is saved as a .ma or. mb file. The only downside is that *fileInfo* only accepts strings. Therefore, the data needs be converted to a string before being saved. This results in the data already being stored as strings in the curvenet dictionary. Since *pymel.core* works with its own datatypes, the strings must be converted back into the data types using *pm.PyNode()*.

4.3 User Test

The tool was constantly tested throughout the process, but only on simple objects. This test is intended to provide further insight into the functionality with more complex meshes. The objects tested are a human and a quadruped character.

4.3.1 Drawing the Curves

Drawing the curves showed no problems. The sizes of the control handles are reasonable, but with larger objects than the test object, it was sometimes difficult to connect the curves while in the drawing process. However, this can be easily fixed later by using the Merge Point option in the editing environment.

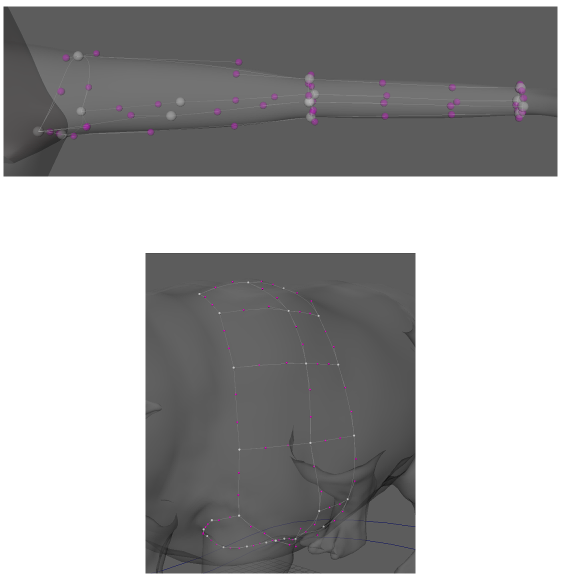

Figure 4.10: curves drawn on the test object (personal collection 2024)

4.3.2 Deformation

A quick deformation test without the Curvenet being connected to the rig showed that the created drop-off control attribute was much needed. It had to be changed for the test objects to get a nice deformation.

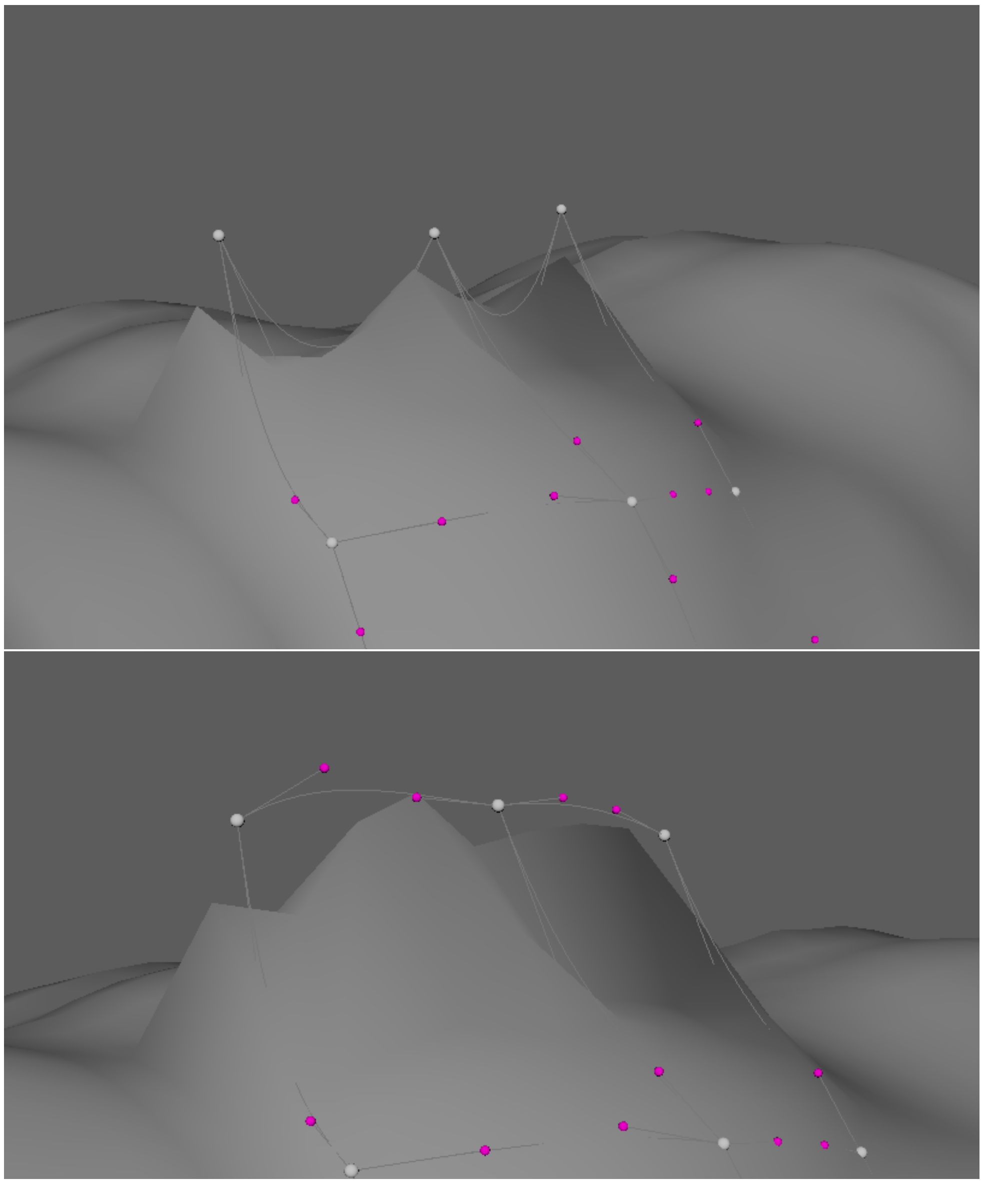

Figure 4.11: Top: default drop-off, Bottom: adjusted drop-off to match the characters size (personal collection 2024)

4.3.3 Animation Test

The created curvenet of the arm shown in Fig. (4.12) was added to the character's existing rig. However the wire deformer did not act as expected, as it was assumed that the wire deformer would completely replace the need for a skin deformer. But removing the skin deformer caused the wire deformer to stop working too. But after using the deformer in addition to the skin cluster, it worked fine and could be used to improve the character's shape.

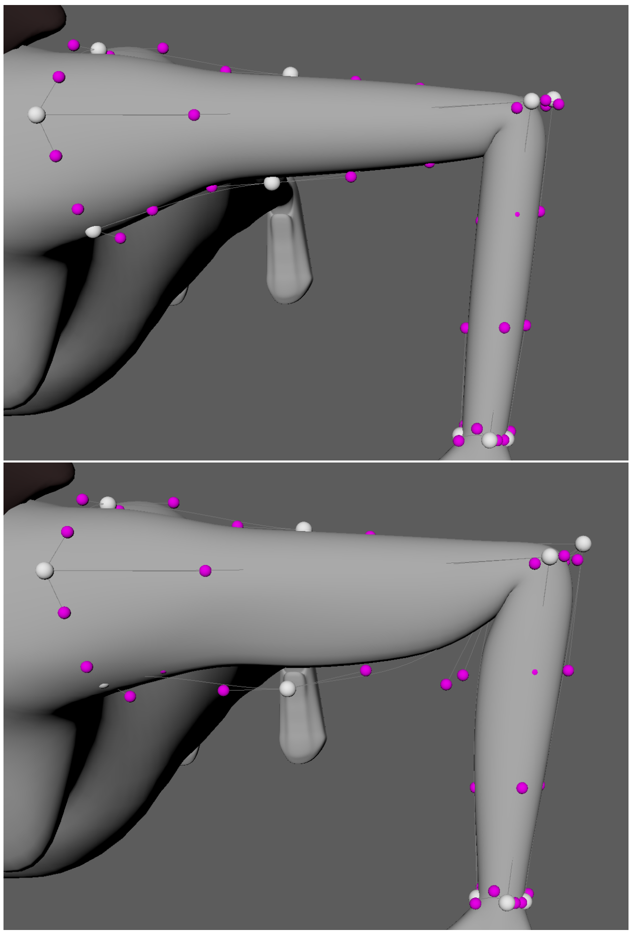

Figure 4.12: Top: deformation with the default skinCluster Bottom: shape enhanced with the curvenets (personal collection 2024)

Conclusion

Curvenets are a new approach to rigging, which uses the use of a free-form deformation with a possible combination to a skeleton-subspace deformation. Instead of influencing the mesh vertices with the skeleton, the control points of the curvenet are influenced by it. This leads to more freedom for the animators, as they can deform the shape of the input mesh with the curvenet while still animating in a traditional way. This eliminates the need for corrective blend shapes. The rigger hereby has fewer control points which need to be attached to the skeleton.

The goal of this work was to implement a tool in Maya for creating curvenets. Its requirements were ease of installation and use, as well as exclusive use of Maya's native functions. The user tests showed that the tool is working for complex characters and provided promising end results. However, there are a few things that could be improved.

Although the size of the control handles is based on the size of the object and its faces, it is not always created as excepted. The control handles tend to be too large for small objects, while also being too small for large objects. This complicates drawing the curves occasionally, as the same size is used as a threshold to connect the curves. Therefore, the existing approach needs to be improved, or a new one found.

To be able to edit the curvenet every time a scene is re-opened, the data is saved in the

Maya file with fileInfo. This leads to a large amount of data for large curvenets. Especially for saving each curvenet individually. A possible improvement could be to only save the list of existing curvenets in the scene and not each curvenet individually. The other information needed could be extracted based on the name and hierarchical structures. However, this could then lead to an incorrect execution of the tool, as names and hierarchies can be deleted or renamed by the user.

Currently, the tool allows the creation of different curvenets in a scene, depending on the selected mesh. However, it is not possible to combine two different curvenets to the same mesh. Therefore, the introduction of sublayers for the curvenets can make sense. This would provide a better overview of the curvenets and allow the user to divide the curvenets into further sections, such as the distinction between arms, legs and body. This would also open new methods like mirroring profile curves.

An important feature of a good rig is its scalability. As part of this work, the scalability of the curvenet was not tested and therefore requires further testing and possible improvements.

For the purpose of this work, Maya's built-in wire deformer was sufficient and produced good results. However, as was seen in the user test, the wire deformer did not replace the skin deformer, as initially assumed. Rather, it is used as an additional layer. However, this eliminates the need for corrective blendshapes and still achieves a good deformation. In further iterations, a customised wire Deformer based on De Goes et al. (2022) newly introduced cut-cell algorithm can be developed and therefore replace the need of skeleton-subspace deformation completely.

References

De Goes, F., Sheffler, W. and Fleischer,K., 2022. Character articulation through profile curves. *ACM transactions on graphics* [online], 41 (4), 1–14. Available from: http://dx.doi.org/10.1145/3528223.3530060.

Döring, D., 2017. radiale Basisfunktion [online]. Spektrum.de. Available from: https://www.spektrum.de/lexikon/mathematik/radiale-basisfunktion/8900 [Accessed 11 Aug 2024].

Gal, R., Sorkine, O., Mitra, N. J. and Cohen-Or, D., 2009. iWIRES: An analyze-and-edit approach to shape manipulation. *In*: *ACM SIGGRAPH 2009 papers.* New York, NY, USA: ACM.

Gleicher, M., 2016. Curves. *In*: *Fundamentals of Computer Graphics.* Fourth edition. | Boca Raton: CRC Press, Taylor & Francis Group, [2016]: A K Peters/CRC Press, 359–404.

Joshi, P., Meyer, M., DeRose, T., Green, B. and Sanocki, T., 2007. Harmonic coordinates for character articulation. *ACM transactions on graphics* [online], 26 (3), 71. Available from: http://dx.doi.org/10.1145/1276377.1276466.

Lewis, J. P., Cordner, M. and Fong, N., 2000. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. *In*: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*. New York, New York, USA: ACM Press, 165–172.

Liu, F., Liang, R. and Ye, D., 2003. Skeleton Subspace Deformation with displacement map. [online]. Available from: http://dx.doi.org/10.2312/egs.20031045.

McLaughlin, T., Cutler, L. and Coleman, D., 2011. Character rigging, deformations, and simulations in film and game production. *In*: *ACM SIGGRAPH 2011 Courses*. New York, NY, USA: ACM.

Nguyen, D., Talbot, J., Sheffler, W., Hessler, M., Fleischer, K. and de Goes, F., 2023. Shaping the elements: Curvenet animation controls in pixar's elemental. *In*: *ACM SIGGRAPH 2023 Talks*. New York, NY, USA: ACM.

Parent, R., 2008. *Computer Animation: Algorithms and Techniques*. Burlington: Morgan Kaufmann.

Sederberg, T. W. and Parry, S. R., 1986. Free-form deformation of solid geometric models. *In*: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM.

Singh, K. and Fiume, E., 1998. Wires: A geometric deformation technique. *In*: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*. New York, New York, USA: ACM Press, 405–414.

Appendix

Algorithm 9 Find Closest Point

```
listOfPointsOnMesh = all the curve points that are on the mesh
if not point in listOfPointsOnMesh then
    newPoint = point
else
    if point in listOfPointsOnMesh then
        newPoint = point
    else
        get closest Point in List with threshold
        if distance of clostestPoint and point are <= threshold then
            newPoint = closestPoint
        else
            newPoint = point
        end if
    end if
    append Point to listOfPointsOnMesh
    newPoint
end if
```

Algorithm 10 Delete Curve

```
curveParentGroup = parent of curve
connectedAttributes = connections to the curve
clusters = []
for connection in connectedAttributes do
    if connection == clusterHandle then
        append connection to clusters
    end if
end for
delete curveParentGroup                    ▷ deletes bezier curve and the two tangents
remove curve from the profilecurve list of the curvenet
for cluster in clusters do
    children = children of the cluster
    if len(children) == 1 then
        clusterHandle has still a connection
        appears if the curve was not connected to any other curve
        if children == "parentConstraint" then
            get the target of the parentConstraint           ▷ == control handle
            delete control handle and cluster
        else if connection of the cluster == Cluster then
            cluster is not connected to anything anymore
            happens to curves that were connected to other curves
            for child in children do
                if children == "parentConstraint" then
                    get the target of the parentConstraint   ▷ == control handle
                    delete control handle and cluster
                end if
            end for
        end if
    end if
end for
```

Algorithm 11 Check For Existing Cluster

```
listToAdd                         ▷ List of all the points to be added to the cluster
clusterExists = False
existing Cluster = 0
clusterInScene = all existing clusters in the scene
for clusters in clusterInScene do
    pointsInCluster = all the points connected to the cluster
    pointPosition = positon of Cluster
    if distance between pointPosition and listToAdd Point position < 0.001 then
        clusterExists = True
        existing Cluster = cluster
    end if
end for
if clusterExists then
    add the listToAdd to the cluster
else
    create a new cluster
end if
```

Algorithm 12 Merge Points

```
connectionList = []
poinList = []
for for spheres in selected Spheres do
    get the connected cluster
    for connections in cluster do
        append connections to connectionsList
        if not "tangents" in points then
            only adds beziercurve points
            append points to pointList
        end if
    end for
end for
averagePoint = 0
for point in pointList do
    averagePoint += point
end for
averagePoint ¯len(pointList)
ensure that averagePoint is on the mesh, otherwise get the closestPoint on the mesh
for point in pointList do
    get start and endpoint
    delete Curve
    if ".cv[0]" in point then startPoint = averagePoint
    elseendPoint = averagePoint
    end if
    create new ProfileCurve
end for
```

Algorithm 13 Seperate Points

```
connectionList = []
poinList = []
for for spheres in selected Spheres do
    get the connected cluster
    for connections in cluster do
        append connections to connectionsList
        if not "tangents" in points then
            only adds beziercurve points
            append points to pointList
        end if
    end for
end for
for point in pointList do
    get start and endpoint
    delete Curve
    create new ProfileCurve
end for
```

Algorithm 14 Add Deformer

```
get the child groups of the mainCurvenetGroup
if not baseCurvesGrp in child groups then
    freeze the transformation of all control handles
    create baseCurveGroup and baseClusterGroup
    bezierlist = []
    baseCurveList = []
    for bezierCurve in curvenet do
        bezierList.append(bezierCurve)
        bezierCurveList.append(duplicate of bezierCurve)
        get the cluster connections of bezierCurve
        for cluster in cluster connections do
            get the group of the connected sphereHandle
            connect the group to a new cluster, which controls the duplicate curve point, so
the baseWire moves along with the rig
        end for
    end for
    add attribute "dropOffDistanceControl" to mainCurvenetGroup
    add wire deformer wireDef to the mesh
    for i in bezierList do
        connect bezierList.worldSpace[0] to wireDef.deformedWire
        connect baseCurveList.worldSpace[0] to wireDef.baseWire
        connect dropOffDistanceControl to wireDef.dropOffDistance
    end for
    set rotation of wireDef to 0
    hide baseCurveGroup and baseClusterGroup
end if
```