NIELS VAN DER KAAP

INTERACTIVE NON-PHOTOREALISTIC MODELING

UNIVERSITY OF GRONINGEN

[November 11, 2009 at 15:01]

SUPERVISORS:
Tobias Isenberg

LOCATION:
Groningen

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Most games and simulators that are currently available try to create an environment that is as close to the real world as possible. The primary goal of modelers that are involved in this kind of projects is to create realistic 3D-models. Many tools exist that can perform this task, mainly because realistic modeling is more or less contrained to basic shapes (cubes, tetrahedra and spheres) and physical properties (reflectivity of light and gravity).

A branch of computer graphics that abandons the principle of obtaining a realistic rendering is NPR: Non Photorealistic Modeling. There are various reasons for using NPR techniques: it is used for emphasizing certain parts of a model (see figure 1) as well as for expressing certain ideas of the modeler. Examples of this last reason are the conversion of a real world model into a comic book rendering for inclusion in an animation movie, or the application of certain algorithms to create a pen stroke image (see figure 2).

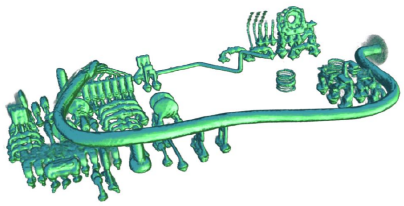


Figure 1: The engine is rendered in such a way that parts can be distinguished. Image taken from [6].

Figure 2: A NPR technique is used to render a 3D model into a situation that looks like a true drawing. Image taken from [8].

Due to NPR, the focus of 3D modeling shifts more and more to the artistic side of the job. Still, most 3D modeling software packages that are available today require extensively on the mesh based implementation in which models are stored. This is overcome in interactive NPM (Non Photorealistic Modeling). The basic idea is that designing models should be as simple as ordinary drawing. An example is TEDDY [3]: a sketching interface that can be used to draw simple 3D models. But

it might also be a tool that is capable of creating an animation from a static object [13]. The applications are thus very broad.

This thesis implements a method for editing NPM models in an interactive way that resembles oridinary drawing better than current 3D modeling packages provide. The application is capable of loading and editing 3D triangular mesh based models. This choice has been made because triangular based meshes are very common, which makes it easier to obtain sample models.

The thesis starts with an overview of related work. This includes a number of papers and methods that are currently used for editing (3D) models interactively.

Next, the algorithm is introduced that is used to apply local deformations to a 3D model. This discussion starts with the concept and the math of the method. After introducing the algorithm, the actual implementation is discussed in chapter 4. It does not only treat the implementation of the deformation algorithm, but also that of the framework that was written for it.

Finally, some examples are given for results that have been obtained by using the software that was developed. This includes multiple test cases in which different sequences of actions are executed on models.

2 RELATED WORK

Modeling is an iterative process. The first version of a model merely exists from a number of very basic shapes (cilinders, spheres and squares). The design is then refined and details are added. This procedure is not only common to NPM modeling using computers, but also to ordinary modeling on paper [11]. An important difference between the computer and paper methods is that the computer is capable to actually interpret a model as having three dimensions, while the paper version is just a projection. However, no direct method of converting projections into actual models is available. This causes computer modeling to be different (and harder) from modeling on paper.

Different methods have already been developed to calculate 3D models from 2D projections. Most of them rely on sketching interfaces in which freeform bodies are drawn using a mouse or a pen. An example of one of these implementations is TEDDY [3]. Another method of converting projections into 3D models was suggested by Steve Tsang et al in [14]. Instead of using a single projection to determine the model, a series of projections from different angles is recombined into the original model.

The two methods described above are capable of creating a 3D model from scratch. Another use of modeling sotware is, however, that is should be capable of editing existing model. An example of such a modeling operation was developed by Eyiyuekli et al [2]. It extends the functionality of TEDDY with the ability to add local deformations to an existing (photo realistic) model.

The goal of this thesis is to develop a local deformation algorithm that is capable of bending (certain parts of) a model using a sketch based interface. The remainder of this section will describe different techniques that can be used to implement this algorithm.

2.1 BENDING MODELS USING A SKELETON

A skeleton keeps the body together. In general, the position of the skin is determined by the orientation of the closest point of the skeleton. Besides that, a skeleton constrains the movements that a body can make: bending is only possible at the joints between bones. These two observations make the skeleton approach a good way of modeling creatures [7].

There are multiple ways for implementing the skeleton approach for a model. The easiest one is to define two spines: one old spine (see Figure 3) that resembles the current spine of the model and another one that descripes the new situation (see Figure 4). Each vertex of the model is mapped to some point on the old spine and its orientation with respect to the spine is stored. Next, the old spine is projected onto the new one, and all vertices are reallocated occording to their mapped point on the new spine [9].

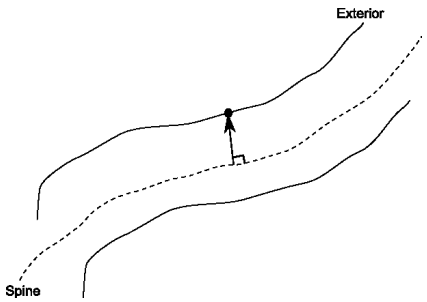


Figure 3: The old spine, the angle between the tangential vector of the spine and the vector that is shown is 90 degrees.

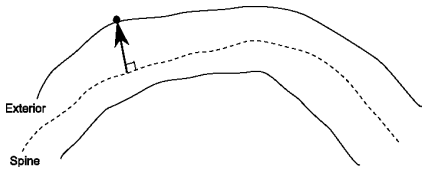


Figure 4: A newly defined spine. The angle between the tangential vector of the spine and the vector that is shown is kept at 90 degrees.

2.1.1 *Skeleton approach using vertices*

The approach of editing models using a spine comes forward in TEDDY, a program that was created by Igarashi et al. in 1999. [3] It's main purpose is to provide an interface to create NPR models in a straight forward way that is close to ordinary drawing on a paper. For example, a two-dimensional body is drawn first that will resemble the body of some creature. This body is then automatically extended into a three dimensional shape. It is then possible to rotate the shape and attach freeform limbs at different points of the body in the same way that the body was drawn (see Figure 5).

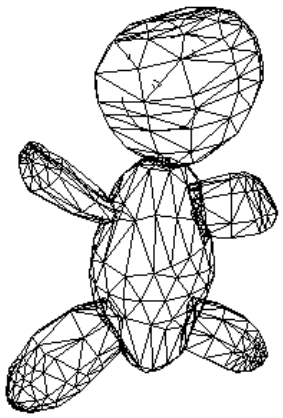
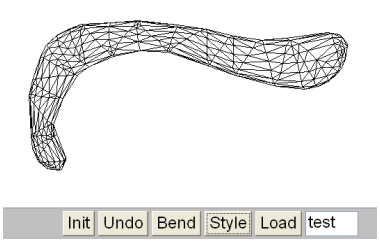
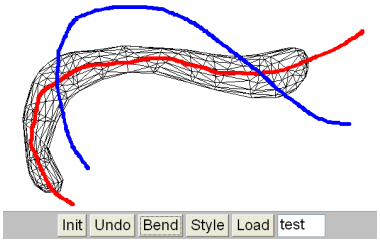


Figure 5: A creature that has been made using TEDDY. First, the body was drawn and then the limbs and head were added. The current representation uses meshes instead of a filled layout.

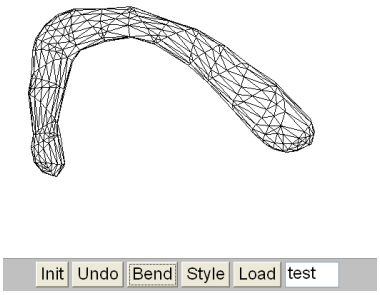
However, the feature that is most interesting for this thesis is that TEDDY contains a method for bending the entire model using a skeleton approach. The user first needs to press the bend button and then has to draw the existing spine. Finally, the user may draw the new spine, and the model is bent. An illustration of this method is shown in Figure 6.



(a) A simple body.



(b) The red line shows the old spine, the blue line shows the new one.



(c) The simple body after bending.

Figure 6: An impression of how models are bent using TEDDY. After the body has been created, the bend button is pressed and two spines have to be given. Finally, the body of the last body results.

A disadvantage of TEDDY is that it is not possible to select a small part of the model that should be bend: the operation affects the entire model. It is, for instance, not possible to just bend the arm of some creature. Two other minor drawbacks are that no custom meshes can be loaded into the environment and that a button needs to be pressed explicitally to enable bending. This last point makes the interface less intuitive as no buttons need to be pressed in ordinary drawing. This will be the starting point for the work in this thesis. But before proceeding with this, some other methods are explained first.

An extention to the skeleton method above would be to automatically extract a skeleton from the 3D model, instead of letting the user pick one [5]. This will save the user work and results in more realistic deformations.

2.1.2 *A skeleton using voxels*

When mapping models to a single spine, errors might occur for vertices that have a large distance to the spine. The reason for this is that is becomes harder to determine the point on the spine to which a certain vertex should be mapped. A method to overcome this problem is to use voxels [12].

Unlike the previous technique, this method requires the user not to input two spines. Instead, it creates an additional 3D grid that divides the region into a large number of small volume elements (voxels). Bending operations are then performed by reshaping the voxels.

The advantage with respect to the previous method is that no problem arise with vertices that have larger distances to the spine: all vertices remain in their original voxel. However, a disadvantage is that some bending operations are not possible. A problem arises for grids that are 'faulded' (see Figure 7): It is unclear what should happen with vertices that lie within two voxels.

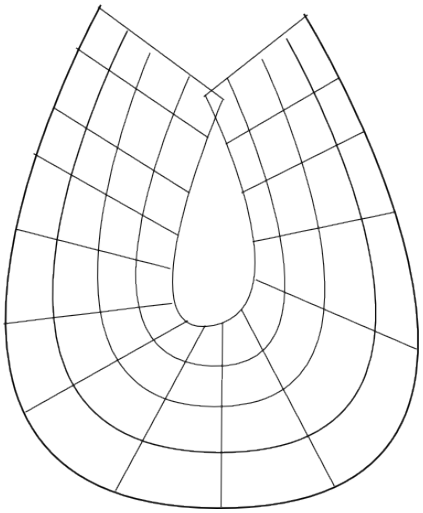


Figure 7: A faulded grid: parts of the grid overlap, which makes it hard for vertices to be assigned to a single voxel.

2.2 AS-RIGID-AS-POSSIBLE MANIPULATION

Another approach to perform bending is as-rigid-as-possible manipulation. It works by minimizing the distortion of each vertex in the triangular mesh [4], [10], [13]. The result is that the deformation is spread across the entire model. By adding a bend factor to each vertex, it is also possible to give parts of the model a different stiffness. This is actually the underlying idea of the skeleton method (though finding the right stiffnesses for each vertex is a problem). An advantage of the as-rigid-as-possible that no additional mesh is needed.

Like with the previous techniques, as rigid as possible manipulation tries to maintain the original shape of models. The main difference however is that movements are not constrained by a skeleton: any kind of vertex translation is allowed. If one would replace the wrist of a human body, the entire arm could end up in an arc. It is thus only possible to determine the endpoint of a single vertex. That makes this technique unsuitable for editing objects that only allow constrained movement.

2.3 SUMMARY

In this section, three techniques were reviewed that can be used for reshaping three dimensional models. The first was based on manipulation

using two user drawn spines: one before, and one after bending. This technique mainly relies on determining the shortest distance between model's vertex and the spine. The second technique is closely related to the first one. But instead of relying on the shortest distance to the spine, an additional grid is introduced to which each model vertex is mapped. The advantage of this approach is that determining the location of each vertex after bending becomes less of a problem, as the first algorithm fails for vertices that have a reletive large distance to the first spine. Finally, a physically more correct method was proposed that relied on as rigid as possible manipulation. This approach is mainly usefull for objects that do not have a definable spine, as motion is not constrained. The first method will be used later on for the actual implementation, as it is more suitable for modeling creatures that have a skeleton than the other method.

3 INVOLVED MATH

The models that are created and edited with the software that is beeing developed during this thesis will just be used for triangular meshes. All operations that are performed on models require paths for determining the type and extend of the action that needs to be taken. Paths start off as a sampled series of coordinates that are obtained by drawing a line with the mouse. Because this representation is not very suitable for analytical purposes, a method is first needed to obtain a more detailed view on paths. In this chapter, these operations are reviewed shortly and an overwiew is given of the most important mathematics that is used.

3.1 CONCEPTS

The operations that are reviewed in this section can be divided into two groups. The first group contains algorithms that are needed for creating and managing paths:

- Obtaining the continuous representation of a path from a series of coordinates.
- Determining the type of path[1]

The second group of algoritms are the actual operations that the user will perform on a model:

- Partial selection of the model: selecting all points that lie inside a closed path.
- Bending a model by specifying its spine before and after the operation.

3.1.1 *Obtaining the continuous representation*

The creation of a path starts by pressing the left mouse button: the screen coordinates that the mouse pointer traverses are then stored into a vector until the button is released again. In this way, a vector of 2D screen coordinates is obtained.

A problem with this approach is, however, that the position is obtained as a function of time. While only the traversed path is required for the analysis. This leads to problems when the mousepointer stays as the same location for some time. Besides that, the time-density of the vector is too high, as a coordinate is taken every few milliseconds. The solution is therefore to resample the vector in such a way that it becomes less dense and less time- dependant. This is done by first taking the derivative of the vector, so a new vector is obtained with elements that each describe the relative change with respect to the previous element. Then, a sampling point is choisen from the original vector everytime that a certain distance has been traversed in the new vector.

1 See Figure 10 for the type of paths.

A series is now obtained that contains (approximately) equally distant 2D screen coordinates. These are then converted into 3D model coordinates.[2] This requires an additional dimension to be introduced: the screen coordinates will be placed in the view plane, while the additional dimension is defined by the view vector (see Figure 8).

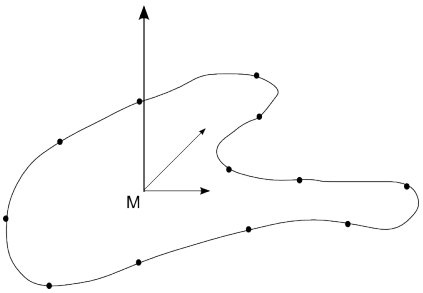


Figure 8: The basic layout of a path shown from a perspective view. The view vector is shown as the vector which is oriented perpendicular to the path's plane: this vector defines the additional dimension.

The discrete representation of a path can be used for drawing purposes, but is unsuitable for analytical applications like selection or bending. Some method is, therefore, needed to obtain a more precise (continuous) desciption of the path. One possibility would be to obtain a polynomial description (or actually three: one for each dimension) using a fitting algorithm (for example secant's method). However, large paths require high order polynomials, and these tend to oscillate rapidly [1]. A single polynomial is therefore not suited for obtaining continuous representations of paths.

Another approach is to use piecewise continuous polynomials (again one for each dimension). This is done by fitting a third order polynomial $S_j(t)$ between each two neighbouring points of the coordinate series (see Figure 9). Any of these polynomials then describes the path for $t|t >= 0 \wedge t <= 1$ between two coordinates. In this way, a parametric function is obtained that descripes the entire path continuesly[3] that can be used for further analysis.

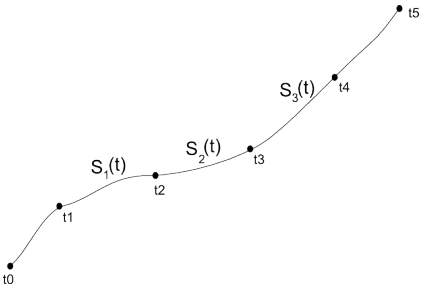


Figure 9: A piecewise continuous function: it exists of a number of polynomials that have the same value and derivative at the boundaries.

2 More on this in Section 4.2.
3 For a more detailed analysis, see Section 3.2 on natural splines.

3.1.2 *Determining the type of path*

The operation that needs to be performed on a model depends on the shape of the path that has been drawn. In this way, no button is needed in the program. There are three main types of path with a different meaning that can be distinguished: circular closed paths, circular open paths and regular paths. Each of these is detectable in a distinct way:

- **Circular closed path** This type of path is detected by a relative small distance between the start- and endpoint.
- **Circular open path** This type of path is the same as a circular closed path, only the signs of the derivatives of the begin- and endpoint are unequal.
- **Ordinary path** This type of path is selected when non of the other types is recognized.

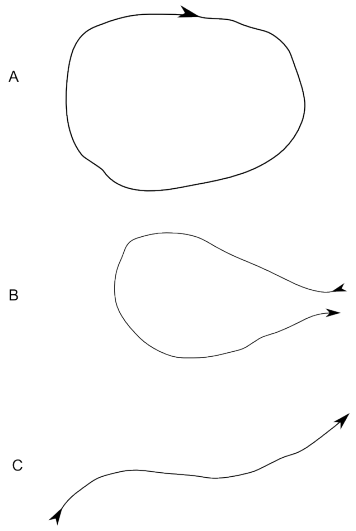


Figure 10: The three types of paths: a) shows a circular path, b) shows a circular open path and c) shows an oridinary path.

The second type of path requires the derivative of the begin and end point to be known. This is not very hard, as the parametric description of the path can easily be derived.

3.1.3 *Partial selection of the model*

In order to perform bending on a small part of a model, it needs to be possible to select only a part of the model. This is done by drawing a circular path around the part of the model that needs to be selected. After the path has actually been detected as circular, the selection algorithm kicks in.

The selection algorithm iterates through all vertices to determine whether or not they lie inside the circle. First, the point on the path is determined that is closest to the vertex, this is done using the algorithm

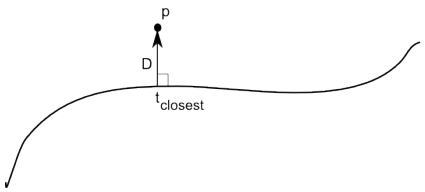


Figure 11: The closest distance between the path and point p is represented by the difference vector D.

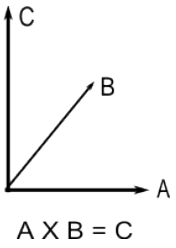


Figure 12: The visual representation of the cross product between vectors A and C.

that is explained in Section 3.3. Then, the difference vector between the closest point on the path and the vertex is taken (see Figure 11).

A clear distinction can be seen between points that lie above or below the line: they have either a 'left' or 'right' orientation with respect to the tangential vector of the path at their closest point. This approach can be extended to paths as points may either lie within or outside it. Some method is still required to test this mathematically though. This is done by taking the cross product (see Figure 12) between the tangential vector at the closest point on the path and the difference vector from the closest point to the vertex (see Figure 13). For the situation at hand, vertices that have resulting vectors that come towards the reader will lie outside the path and vertices that have resulting vectors that face away from the reader will lie within the path.

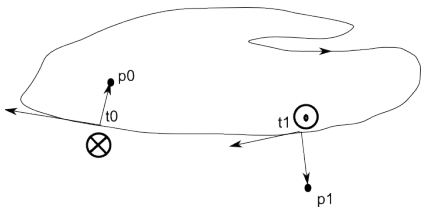


Figure 13: The resulting vector from the cross product point either toward or away from the reader, depending on whether the vertex lies inside or outside the path.

In general, however, this depends on the clockwise or counter clockwise orientation of the path: this is shown in Figure 14. Two vertices are shown that both lie within the path. The upper path has clockwise orientation and the lower path has counter clockwise orientation. It can clearly be seen that the direction of the vectors that result from the cross product are opposite. The last step is, therefore, to find a way to cope with the orientation of the path.

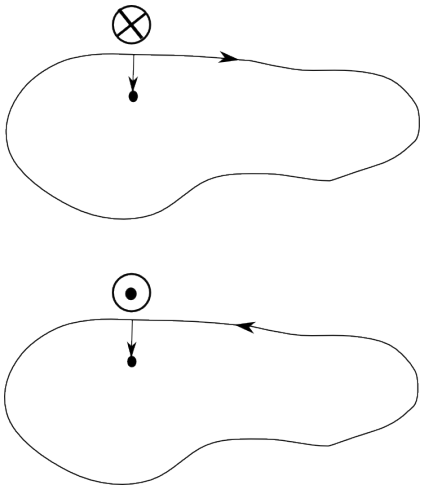


Figure 14: The direction of the resulting vector from the cross product depends on the clockwise or counter clockwise orientation of the path.

This is done by comparing the direction of the vector with that of the vector that results from performing the cross product between the middle point of the path and its closest point on the curve. If the directions are the same, the vertex lies inside the circle. Otherwise, the vertex lies outside the circle. This comparison might go wrong when the path is not convex, but it will succeed in most cases.

3.1.4 *Bending*

Like the selection algorithm, the bending algorithm strongly depends on the closest distance problem that is treated in Section 3.3. The bending algorithm requires two paths to be drawn. The first path is the initial spine, while the second path will be the new spine. Once the paths are known and the operation has been identified as a bending one, the algoritm first iterates through the selected vertices and determine the closest point on the first spine for each of them. The distance and orientation to the closest point is stored as well (see Figure 15).

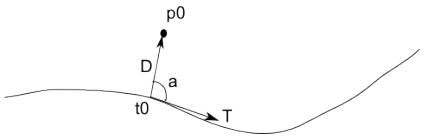


Figure 15: The situation before bending: both the closest point and the orientation with respect to the closest point are stored. The perpendicular part should be zero, but is stored for safety.

The second step is to map the selected vertices to the new spine. This is done by mapping the old spine to the new one, and adding the distance and orientation between the old spine and the vertex (see Figure 16). Other examples are shown in chapter 5

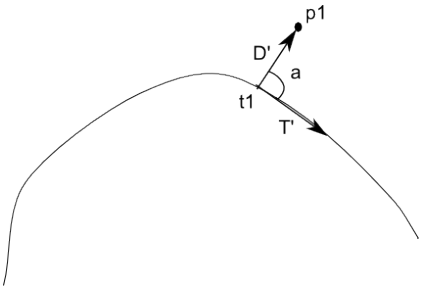


Figure 16: The position of the vertex is altered by applying the position on the new spine.

3.2 CUBIC SPLINE INTERPOLATION

Paths are initially obtained as a discrete series of coordinates in time: for instancef a path that exists of N points. Using these series for displaying purposes works just fine, but it lacks on detail for analytic features. The reason for this is that the description of the path is discontinuous: the path itself is known anywhere, but its derivative is nowhere known.

One way to overcome this problem is to use cubic spline interpolation: instead of a single continuous function for each dimension, a piecewise continuous polynomial is returned for each dimension that exists of $N-1$ third order poynomials between each two consecutive points from the original series. The third order polynomials are denoted $S_j(t)$ on the interval $[t_j, t_{j+1}]$ for $j = 0, ..., N-1$ (see Figure 9).

This means that, in total, $N-1$ polynomials are created per dimension $(S_0(t), ..., S_{N-2}(t))$ for a discrete path that consists of N points. Other properties of a cubic spline interpolations, that are important for further analysis are:

- $S_{j+1}(t_{j+1}) = S_j(t_{j+1})$. Two consecutive polynomials are equal at their boundaries.
- $S'_{j+1}(t_{j+1}) = S'_j(t_{j+1})$ with $j = 0, ..., N-3$. The derivative of two consecutive polynomials are equal at their boundaries.
- $S''_{j+1}(t_{j+1}) = S''_j(t_{j+1})$ with $j = 0, ..., N-3$. The second order derivative of two consecutive polynomials are equal at there boundaries.
- $S''_0(t_0) = S''_{N-2}(t_{N-1}) = 0$. The second order derivatives of the begin and end points are zero (So the derivative is constant).

A derivation [1] of the cubic spline interpolation algorithm is now given. Each polynomial has the following format:

$$S_j = a_j + b_j(t - t_j) + c_j(t - t_j)^2 + d_j(t - t_j)^3 \tag{3.1}$$

Using the first property from the above list and keeping in mind that a_j is equal to the point that is fitted the following equation can be derived:

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j j_j^3 \tag{3.2}$$

Where $h_j = t_{j+1} - t_j$. Taking the derivative of S_j at $t = t_j$, the following equation is obtained:

$$S'_j(x_j) = b_j \tag{3.3}$$

Using the second property from the above list, the next equation follows:

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j \tag{3.4}$$

In the same way as in Equation 3.3, it is now possible to define the following relationship:

$$c_{j+1} = c_j + 3d_j h_j \tag{3.5}$$

When Equation 3.5 is now rewritten for d_j and this value is substituted in Equations 3.2 and 3.4, the following relationships are obtained:

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \tag{3.6}$$

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}) \tag{3.7}$$

Rewriting Equation 3.6 for b_j and b_{j+1} and substituting these in Equation 3.7, the following tridiagonal linear system is obtained:

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{3.8}$$

This system is then solved numerically by Crout Factorization as discussed in [1]. In this way, a series of $N-1$ third order polynomials is obtained that is piecewise continuous and represents the path.

3.3 CLOSEST DISTANCE PROBLEM

An important issue throughout this thesis is how to determine the closest distance from a point p in space to a certain path (see Figure 11): it is both used for the bending and selection algorithms. The problem with this algoritm is that an analytic method does exist, but is quite complicated to implement.

The layout of the problem is to first take the description of the distance between the point in space and an arbitrary point on the path:

$$d(t) = \sqrt{(path.x(t) - p.x)^2 + (path.y(t) - p.y)^2 + (path.z(t) - p.z)^2} \tag{3.9}$$

The analytic approach would now require Equation 3.9 to be derived and then to set equal to zero to obtain the minimum distance. The problem with this approach, however, is that it would imply the solving of a sixth order polynomial equation. Which is possible, but takes a lot of effort.

An alternative is, therefore, to use the secant method to minimize Equation 3.9. It only requires the direct function to be known: the derivative is not needed. The secant method is similar to Newton's method, the only difference is that the derivative is estimated instead of calculated directly. The advantage of using Newton is that it will converge a bit faster than secant, but the derivative must be known.

The secant method asks for an initial guess of the point at which a function equals 0 and assumes that such a point exists. It then tries

to approach that point in a number of iterations. In each iteration, the guess is ajusted as follows:

$$p = p0 - \frac{S(p0)}{S'(p0)} \tag{3.10}$$

It thus tries to approach the intersection of the function with the y-axis. However, because Equation 3.9 is larger than zero everywhere (unless p is on the path), the method will only approach a minimum. It is not guaranteed that the minimum that is found in this way is the required global minimum. But it can be forced by giving a good initial guess in nearly all cases.

A good initial guess is important for obtaining the global minimum. The method that is used here uses the discrete representation of the path: it iterates through all points and takes the point that is closest to p.

3.4 SUMMARY

This chapter focussed on the mathematics that are important for the bending and selection algorithms that are explained in the next chapter. First, the conceptual algorithms were explained. This included methods for bending a set of vertices and selecting a subset from a certain set of vertices. From these conceptual descriptions, the important mathematics were discussed:

- **Cubic Spline Interpolation** This procedure is required for determining a continuous representation for a discrete set of points.
- **Shortest Distance Problem** This procedure is necessary for mapping arbitrary vertices to a point on a spine.

For Cubuc Spline Interpolation, Equation 3.8 was obtained. This linear equation is solved using Crout Factorisation to obtain a continuous representation. For the shortest distance problem, no direct solution was found. This was possible, but would take much time to derive. Instead, it was decided to use the numerical secant method to minimize the function. This approach turned out to work well in general.

4 IMPLEMENTATION

In this chapter, the implementation of the software is discussed. First, a small overview of the framework that is used for rendering models is stated. Then the actual implementations of the selection and bending method is given.

4.1 FRAMEWORK

The implementation of the functionality that we desire requires software that is capable of rendering and editing three-dimensional models. Because this is a part of computer graphics that has already been explored extensively, it would be reinventing the wheel to redo it here. Therefore, two software packages will be used to implement the editing and rendering of models:

- OpenGL is used for rendering models. Unlike OpenMesh, it does not contain methods for editing meshes: it is purely based on speed. Inside the program, models will first be manipulated using OpenMesh. After an operation has succesfully be completed, the OpenMesh representation is converted into an OpenGL representation and is rendered to screen. Along with OpenGL, GLUT will be used to handle tasks that have to do with user interface issues: determining the orientation of a model and gathering mouse pointer information that can later be converted into paths.
- OpenMesh is a package that is used for managing models. It contains methods for loading 3D meshes and iterating through all the points of these meshes. During this thesis it was used for managing and editing the actual model: All calculations on meshes were performed using this package.

OpenGL is written in C while OpenMesh has been developed using C++. A decision has therefore to be made on which programming language is used for writing the software for this thesis. Eventually C++ was chosen, because it seemed easier to write C++ wrappers for C packages instead of the other way around. A wrapper was therefore written for OpenGL. The class is called OpenGLWrapper and contains the following functionality:

- It contains code for setting up OpenGL.
- It contains all code that decides how models are drawn, and has methods that make the user able to rotate around the model.
- It contains methods for gathering discrete paths and code for managing and drawing them.
- All user interface handles are located here: operations that the user wants to perform are received here and then passed through to the actual algoritms.

Altought the class is responsible for determining how the model is drawn, the actual drawing occurs outside the class (in the owning Main

class). The reason for this is that it would pin down the way in which models are managed inside the editing part of the program: another possibility would have been to use an abstract OpenGLWrapper that contains a virtual class that is responsible for the actual drawing.

The OpenGLWrapper class is, thus, capable of obtaining paths of the Path class. So once a discrete path is drawn, it is converted into a 'Path' path. This new representation is then used to determine the type of operation that should be carried out on the model (bending or selection) and, finally, a callback function is called that handles the corresponding operation. The callbacks are handled inside the owner-class of the OpenGLWrapper class: the Main class.

Once a discrete path has been obtained using the OpenGLWrapper class, this representation is used to create an instance of a Path class. The path class contains methods for converting the discrete class into a continuous path and for determining the type of path that has been drawn (circular, regular or half-open). The Path class also contains the implementation for determining the closest distrance problem.

OpenMesh is written in such a way that custom properties and methods can be attached to vertex points. In order to use this functionality, a class is needed that overloads and extends the original mesh. This is done in the first part of the ModelLoader header file. It defines a new type of triangular mesh (called Model) that is based on the extended definitions that are put in a structure (see the OpenMESH website for more details)[1]. The rest of the ModelLoader class contains some basic functions for loading and returning the actual triangular mesh.

The Main class gathers the functionality of all classes:

- It loads a model using the ModelLoader class. The ModelLoader contains all OpenMesh related code and is responsible for loading models from and to files.
- It creates an instance of the ModelEditor class using the loaded model. This class is responsible for performing operations on the loaded models, this both includes preparing the OpenGL input data as handling the actual editing operations.
- It contains the callback function for drawing the actual model that is called from the OpenGLWrapper class. The model is obtained from the ModelEditor class.
- Finally, it ties together the selection and bending callback to their respective implementation inside the ModelEditor class.

An UML diagram of the framework is shown below in Figure 17. The purpose of the Notifiable class is to provide an interface for all the callback functions of the OpenGLWrapper class.

4.2 PATH CREATION

When drawing a path inside the GLUT screen, a series of 2D screen coordinates is obtained. These coordinates are not suitable for performing operations on a model, as they do not overlap with the models coordinate system. So the 2D screen coordinates need to be converted into 3D model coordinates. This is done using the gluUnProject() method that is contained in the GLU package of OpenGL. The function requires the

1 http://www.openmesh.org

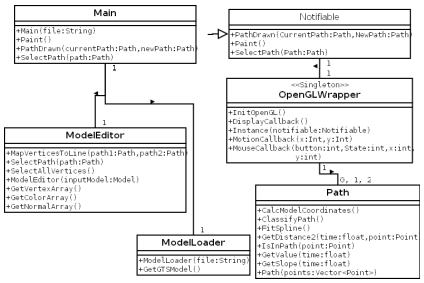


Figure 17: An UML overview of the framework: Only a single instance of OpenGLWrapper is allowed at a time.

screen coordinates as input argument, as well as a third coordinate that resembles the depth. Because this depth is not important for the path, it is set to zero.

As mentioned earlier in this report, a vector is still required that describes the orientation or plane of the path. Once a path has been drawn, this vector is equal to the current view vector: the vector that points from the (center of) the screen towards the view point. This vector is obtained by taking the difference between the vectors that are returned by the gluUnProject() method for the screencoordinates $(0, 0, 0)$ and $(0, 0, 1)$.

Now that a vector of 3D coordinates is available, the actual cubic spline calculation should be carried out. This means that a linear system is solved and the coefficients of the polynomials are returned that together form the piecewise continues function that descripe the path. This is done using the Crout Factorization algorithm. See [1] for an overview of this method.

The final operation that is performed on the path before it can be used for selection and bending, is to determine the sign of the cross product between the avarage point of the path and the tangential vector of the closest point on the path to that avarage point (see Figure 13).

4.3 SELECTING VERTICES

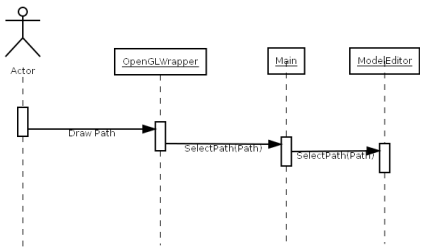


Figure 18: The selection of a path runs through the Main class.

The selection algorithm has already been explained previously on Section 3.1.3. Here, the implementation of it is reviewed in more detail. An activity diagram of algorithm is shown in Figure 18. The algorithm started by calling the selectPath() method in the ModelEditor class. It iterated through all vertexes of the model and calls the isInPath() method from the Path class for each vertex. This method then performs

the algorithm that was already explained. If it returns true, no action is taken. Otherwise, the vertex is flagged as unselected. Finally, the algorithm checks whether there are still vertexes that are selected. If no selected vertex is found, all vertexes of the model are reselected again. In this way, a selection operation on an empty part of space will set all vertices as selected again.

4.4 BENDING THE MODEL

The principle of the bending algorithm has already been reviewed in this thesis. Here, a few more details are given on the implementation on it. The algorithm is started by calling the mapVerticesToLine() method in the ModelEditor class. It requires two instances of Paths as input arguments. The first path descripes the initial spine and the second path is the new spine to which the old spine will be projected.

The first operation that is performed by the algorithm iterates through all selected vertexes and determine the location on the path that is closest to each vertex. Once this is done for all selected vertexes, the second loop is started. This loop iterates through all selected vertexes for the second time and first calculates the orientation of each vertex with respect to its closest point. This orientation is then diveded into a tangential and a perpendicular component. Ideally, the tangential component would be zero (due to the definition of the closest point), but it is recorded anyway. Finally, the new location of each vertex is calculated by taking the position of the closest point on the new spine and adding both the tangential and perperdicular components to it.

4.5 SUMMARY

This chapter discussed the implementation of the modeling application that contains the algorithms that were discussed in the previous chapter. First, a framework was written that is responsible for managing and rendering models. OpenMesh was used for managing, while OpenGL was used to perform the actual rendering. The reason for using OpenMesh is that writing a mesh framework consumes a lot of time.

Once the framework was completed, algorithms were written for selecting certain parts of the current loaded model and for bending that selected part. The main difficulty in this was to convert the data format extracted from OpenGL into model coordinates. Finally, a working application was obtained.

5 CASE STUDY

In this chapter, two examples will be given of results that have been obtained using the program that has been developed in this thesis. The first example just shows the bending capabilities, while the second example also uses the selecting functionality.

5.1 BENDING A MAN

This section shows how to bend a model of a man. The original model is shown in Figure 19. The man is entirely green, this means that all of its vertices are currently selected. The first step in bending is to draw the initial spine: it is shown as the blue line in Figure 20. For this model it just runs from the head to the toes, but other models might require a more complicated form.

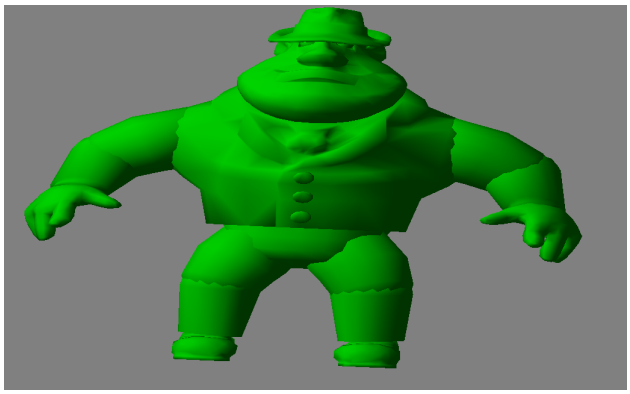


Figure 19: The original model of a man.

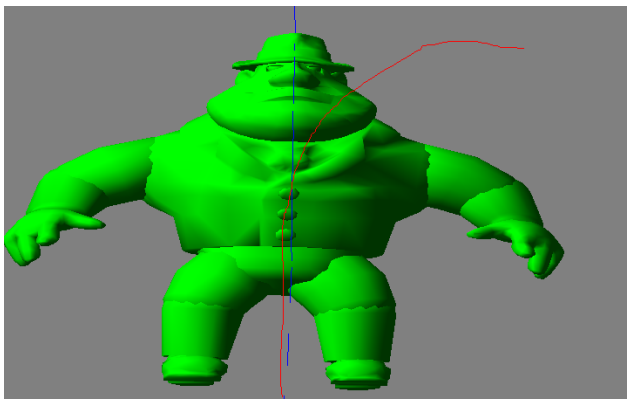


Figure 20: Two lines have been drawn that are blue and red, they represent the old and new spine respectively.

After the first spine has been drawn and the program has determined that it is indeed a spine (and no selection path), the user is required

to draw a new spine. This new spine is shown in Figure 20 as a red line. After this spine has been drawn as well, the transformations are calculated and the situation from Figure 21 is generated. The bending algorithm works as expected: the man's new shape is similar to the second input spine.

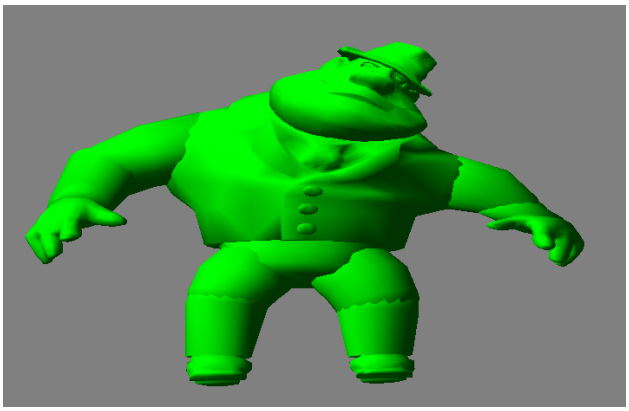


Figure 21: The man after applying the bending algorithm with the spines from Figure 20

5.2 DEFORMING AN EAGLE'S WING

In the second example, the selection algorithm is used to apply bending only to a small part of a model. In more detail, only the wing of an eagle is tried to be bent. The initial model of the eagle is shown in Figure 22. Again, it is all green because all its vertices are selected.

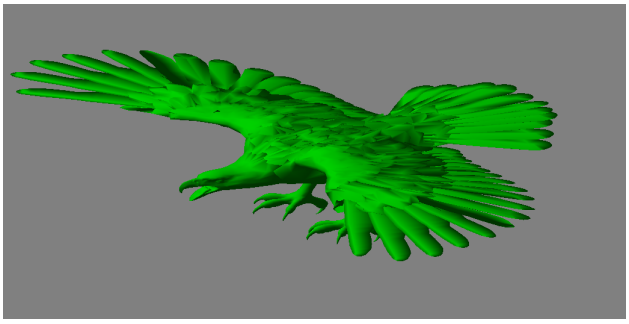


Figure 22: The model of an eagle before editing.

The first step is to select the right wing. This is done by pressing the left mouse button and dragging a circle arround the wing. This is shown in Figure 23. Once the button is released again, the program determines that it is indeed tried to select a part of teh model and the selection algorithm is started. The result is shown in Figure 24.

From the back, it seems that only the right wing is selected now. But when the bird is viewed from above, it can be seen that the right part of the tail is still selected. An additional selection operation is thus required to select the wing. Figure 25 shows the corresponding selection circle of this new selection operation. Finally, the selection from Figure 26 remains after this second operation.

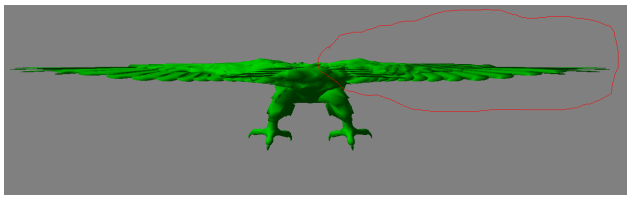


Figure 23: A circular path is drawn to select the vertices that lie inside the region.

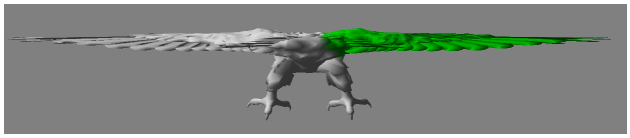


Figure 24: The result of the selection algorithm.

Now that only the right wing is selected, the bending operation should be performed. First the view is changed so that the bird is seen from the back, and then the two new spines are drawn (see Figure 27). Finally, the model from Figure 28 is obtained, in which the orientation of the bird's right wing has changed. The deformation was performed within a second on a laptop with an Intel Core 2 processor and 2 Gb of RAM.

5.3 SUMMARY

In this chapter, the algorithms that are developed in the previous chapters were used to perform selection and bending operations on two different models. For the first model, only bending was used: a general spine was defined for the entire model. For the second model, one part of the model (a wing) was selected before applying the bending operation.

The results that have been obtained are good in general. But sometimes problems may arise when the closest distance algorithm fails, and some vertices may end up at a wrong place after bending.

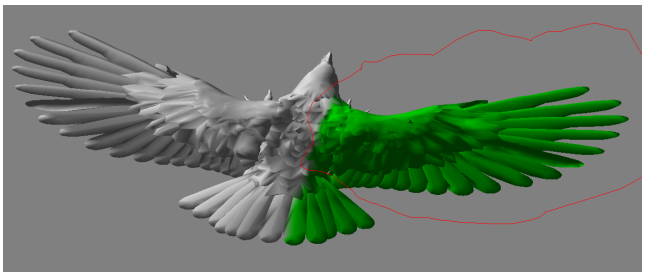


Figure 25: A small part of the bird's tail is still selected. The red circle will induce a selection operation to remove this part.

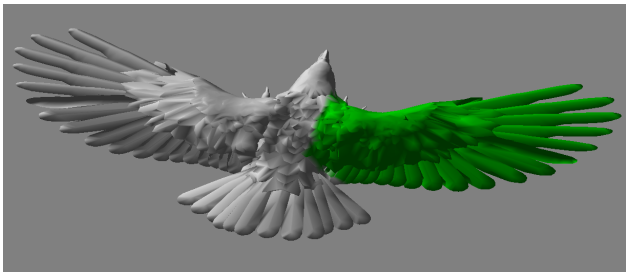


Figure 26: The selected part of the eagle after the second select operation.

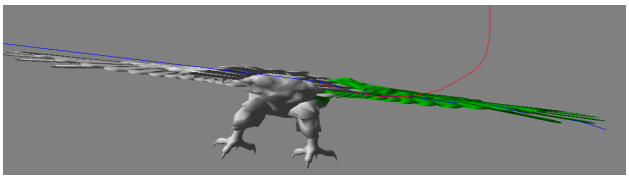


Figure 27: Performing the actual bending on the birds right wing.

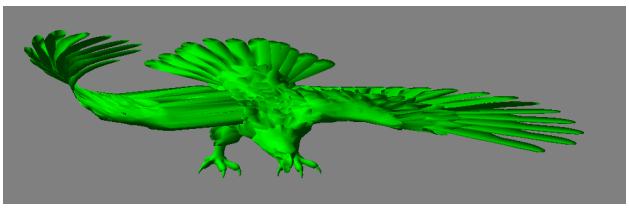


Figure 28: The right wing has been bent.

6 CONCLUSION

6.1 SUMMARY

The main goal of this project was to create a software program that is capable of performing Non Photorealistic Modeling operations on three-dimensional models in an interactive sense. The interactive part in this means that modeling should occur in a natural way that is more straight forward to use than the methods that are provided by most other tools. The reason for this is that most of today's tools are driver by the way in which models are rendered: they require the user to create models in the same representation as in which they are rendered. The creation of an interactive tool, therefore, requires an additional software layer that transforms natural user input to meaningfull three-dimensional operations.

It was decided in an early stage that paths would form this additional layer. Paths contain the sampled mouse data from lines that have been drawn by a mouse. These paths would not only determine how operation would be executed, they are also used for determining the type of operation that the user wants to perform. Algorithms were, therefore, needed to recognize the type of path and to convert these paths into three-dimensional transformations. Besides that, a framework was required to provide a clear interface to three-dimensional models for the algorithms to operate on.

It was decided to write all software in C++, because many libraries are available that contain code for rendering and managing the meshes of models in this language. By using these libraries, a lot of work was avoided and more time could be spend on the actual NPM part. Eventually, it was chosen to use OpenGL for rendering and OpenMesh for managing meshes. Both of these packages contain fast and easy methods for rendering models and accessing meshes respectively.

The framework was setup first. It ties together the functionality of OpenMesh and OpenGL: once changes have been made to the mesh on OpenMesh, the model that is shown by OpenGL is updated as well. The framework also deals with all issues that are related to the user interface and the loading process of models, this includes gathering the path information that is used to perform NPM operations. Next, two algorithms were written that operate on the currently loaded mesh. The first of these methods is responsible for selecting a subset of the model's vertices, this is needed for applying local changes to a model. The second method allows the user to bend (parts of) the currently loaded model.

Because both methods rely on paths extensively, parts of the code that are responsible for managing these paths were extended by operations that allowed spline fits to be calculated and the closest distance between the path and arbitrary points to be determined. Once these functions had been written, work started on the actual implementation of the NPM functionality. Using the path functionality and vector calculus, this part of the software was easily implemented. So finally, a program

was created that allowed the user to load a three-dimensional model and to apply (local) bending to it.

6.2 RESULTS

The resulting program has been tested on multiple three-dimensional models (see Chapter 5). The selection algorithm works well, as all vertices that lie within a drawn path are selected. The only problem is that some vertices outside the path are selected as well. The reason for this is that the function that determines the closest distance between the corresponding vertex and the path is only an approximation. This might be improved by replacing the algorithm by a more analytical approach, which takes more time to implement. The problem might also be avoided by performing an additional selection operation, as changes are very small that the approximation function will fail twice.

The bending algorithm relies heavily on the paths that resemble its pre and post spine. These may not be to complicated, or vertices might be mapped to the wrong point on that path. However, the spine approximation function sometimes creates tight loops when sample points lie too close to each other. The solution for this problem would be to use a different approximation method that does not fail on these small distances. Another difficulty arises when spines are drawn for models that have vertices with relatively large distances to the spine: small fluctuations in the first spine might cause close distant vertices to be mapped relatively far away from each other on the spine. This can result in models that are screwed up entirely. This is a fundamental problem of spine algorithms, that might be avoided by using a voxel approach.

The overall results that have been obtained with the program are good: it is possible to apply bending to certain parts of a model. Applications for which the approach can be used include video games and computer animation tools. Although certain parts of the program can still be refined, a user friendly interface is provided that makes editing 3D models more straight forward. This also comes forward when looking at the case study 5. It can thus be said that the primary goal of this thesis, creating an interactive modeling tool for 3D models, is achieved.

6.3 FUTURE WORK

The results that have been obtained in this thesis can be improved by implementing the suggestions from the previous sections: creating better ways for determining spine approximations, using voxels instead of spines and improving the closest distance problem for selection purposes. Another improvement might focus on the ability to bend the main body of a model, which places uneffected parts on the right place after bending. Besides that, additional NPM operations might be implemented that focus, for example, on enlarging certain parts of a model.

BIBLIOGRAPHY

[1] Richard L. Burden and Douglas J. Faires. *Numerical Analysis*. Brooks Cole, December 2004. ISBN 0534392008. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0534392008. (Cited on pages 10, 14, 15, and 19.)

[2] M. Eyiyurekli, C. Grimm, and D. Breen. Editing level-set models with sketched curves. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 45–52, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-602-1. URL http://doi.acm.org/10.1145/1572741.1572750. (Cited on page 3.)

[3] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5. URL http://doi.acm.org/10.1145/311535.311602. (Cited on pages 1, 3, and 4.)

[4] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1134–1141, New York, NY, USA, 2005. ACM. URL http://doi.acm.org/10.1145/1186822.1073323. (Cited on page 6.)

[5] Florian Levet and Xavier Granier. Improved skeleton extraction and surface generation for sketch-based modeling. In *GI '07: Proceedings of Graphics Interface 2007*, pages 27–33, New York, NY, USA, 2007. ACM. ISBN 978-1-56881-337-0. URL http://doi.acm.org/10.1145/1268517.1268524. (Cited on page 5.)

[6] Eric B. Lum and Kwan-Liu Ma. Hardware-accelerated parallel non-photorealistic volume rendering. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 67–ff, New York, NY, USA, 2002. ACM. ISBN 1-58113-494-0. URL http://doi.acm.org/10.1145/508530.508542. (Cited on pages iv and 1.)

[7] Chen Mao, Sheng FengQin, and David Wright. A sketch-based approach to human body modelling. *Computers & Graphics*, 33(4):521–541, August 2009. ISSN 00978493. doi: 10.1016/j.cag.2009.03.028. URL http://dx.doi.org/10.1016/j.cag.2009.03.028. (Cited on page 3.)

[8] Jason L. Mitchell, Chris Brennan, and Drew Card. Real-time image-space outlining for non-photorealistic rendering. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*, pages 239–239, New York, NY, USA, 2002. ACM. ISBN 1-58113-525-4. URL http://doi.acm.org/10.1145/1242073.1242252. (Cited on pages iv and 1.)

[9] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. Sketch-based modeling:a survey. *Computer & Graphics*, 33(1):85–103, February 2009. doi: 10.1016/j.cag.2008.09.013. (Cited on page 3.)

[10] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006. ISSN 0730-0301. URL http://doi.acm.org/10.1145/1141911.1141920. (Cited on page 6.)

[11] Ryan Schmidt, Tobias Isenberg, Pauline Jepp, Karan Singh, and Brian Wyvill. Sketching, scaffolding, and inking: a visual history for interactive 3d modeling. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 23–32, New York, NY, USA, 2007. ACM. ISBN 9781595936240. doi: 10.1145/1274871.1274875. URL http://dx.doi.org/10.1145/1274871.1274875. (Cited on page 3.)

[12] Masamichi Sugihara, Erwin de Groot, and Brian Wyvill. A sketch-based method to control deformation in a skeletal implicit surface modeler. *Eurographics workshop on Sketch based interface modeling 2008 June 11-13*, 2008. (Cited on page 5.)

[13] Daniel Sykora, John Dingliana, and Steven Collins. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *NPAR '09: Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, pages 25–33, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-604-5. URL http://doi.acm.org/10.1145/1572614.1572619. (Cited on pages 2 and 6.)

[14] Steve Tsang, Ravin Balakrishnan, Karan Singh, and Abhishek Ranjan. A suggestive interface for image guided 3d sketching. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 591–598, New York, NY, USA, 2004. ACM. ISBN 1581137028. doi: 10.1145/985692.985767. URL http://dx.doi.org/10.1145/985692.985767. (Cited on page 3.)