Skeleton-free Example-based Animation

Kristian Evers Hansen

Kongens Lyngby 2006
Master Thesis IMM-Thesis-2006-82

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This thesis proposes a new animation framework simplifying the process of creating realistic deformable models, and using these in interactive posing and animation.

Problems and shortcomings with current animation methods includes rigging and skinning a model, and also creating adaptive realistic animations. Recent proposed methods to improve on this are discussed and evaluated, and finally a new animation framework is presented.

The presented method is based upon Laplacian Editing as deformation method, and is using an example based system for animation. The user paints a handle structure on the model, which can be manipulated to create new poses. These handles are also used to define which parts of the model that are rigid, and which is soft, this then determines how the surface deforms when posing. Using this method, the user can create a set of poses, and save them as examples which defines 'good' poses. These poses are to be used in the animation system.

The animation system is an example based system using constraints. The user can enable and manipulate constraints affecting the handle structure of the model. The system will then search for a blend of the examples to create a suitable pose for these constraints.

How this blend is composed, is determined by optimizing an energy function, which the user can alter and tweak to get a desired result. An energy function has been derived, consisting of several terms which can be used depending on which result is wanted. Several optimization methods have been implemented to optimize this energy function, and the user can also choose between these,

depending on which is a priority; speed or precision.

Finally the constraints can be made to follow user-defined paths, and thereby create a fully moving animation, for example a walk animation or a jumping motion.

Good results are achieved, both using Laplacian Editing to enable the user to create full-body deformations and also the example based animation system is creating very believable animations. The method is relatively fast (time per frame), but unfortunately the Laplacian Editing is still too slow to compete with traditional skeleton animation in terms of speed.

Keywords: 3D Animation, Computer graphics, Deformations, Laplacian Editing, Example based animation, Constraint based animation, Skeleton-free animation.

Resumé

Denne afhandling forelår et nyt framework til animation, der simplificerer processen til at lave realistisk deformerbare modeler og bruge disse i interaktiv posering og animation.

Problemer og mangler indenfor nuværende animations metoder inkludere 'rigging' og 'skinning' af en model, og også at lave adaptive animationer. Metoder der er forelået for nyligt, som skulle forbedre på disse mangler, er diskuteret og evalueret, og så herefter er det nye animations framework præsenteret.

Den præsenterede metode er baseret på Laplacian Editing som deformations metode, og den bruger et eksempel baseret system til animation. Brugeren maler en 'handle'-struktur på selve modellen, som kan manipuleres for at konstruere nye positurer. Disse handles bliver også brugt til at definere hvilke dele af modellen der er rigide og hvilke der er 'bløde', og det er med til at bestemme hvordan modellens overflade deformere. Ved at bruge denne metode kan brugeren lave et sæt af positurer, og gemme dem som eksempler der definere 'gode' positurer. Disse positurer skal bruges i animations systemet.

Animations systemet er et eksempel baseret system der gør brug af constraints. Brugeren kan påføre og manipulere constraints og dermed påvirke handle strukturen på modellen. Systemet vil herefter søge efter en blanding af eksemplerne til at skabe en ny passende positur til disse constraints.

Hvordan denne blanding er sammensat, bliver afgjort af en optimerings-funktion, som brugeren kan ændre og indstille for at opnå det ønskede resultat. En energifunktion er blevet udviklet, bestående af flere termer som kan bruges afhængigt af det ønskede resultat. Flere optimerings-metoder er blevet implementeret til

at optimere denne energi, og brugeren kan også vælge mellem disse, afhængigt af om hastighed eller præcision bliver prioriteret højest.

Constraint'sene kan sættes til at følge en bruger-defineret sti og derved skabe en fuldt bevægelig animation, for eksempel en gå-sekvens eller et hop.

Gode resultater er opnået, både ved brug af Laplacian Editing til at lave deformationer og også det eksempel baserede animations system resulterer i meget troværdige animationer. Metoden er relativ hurtig (tid per frame), men uheldigvis er Laplacian Editing stadig for langsomt til at konkurrere med traditionel skeletanimation på dette punkt.

Keywords: 3D Animation, Computer grafik, Deformationer, Laplacian Editing, Eksempel baseret animation, Constraint baseret animation, Skeleton-fri animation.

Preface

This thesis has been prepared at the Section for Computer Graphics, Department of Mathematical Modelling, IMM, at The Technical University of Denmark, DTU, in partial fulfillment of the requirements for the degree Master of Science in Engineering, M.Sc.Eng. The extent of the thesis is equivalent to thirty ETCS points.

The thesis proposes a new method for skeleton-free deformation of models, for use in an example based animation system.

It is assumed that the reader has a basic knowledge in the areas of Computer Graphics, computer animation, linear algebra and vector calculations.

The poster shown at the end in the appendix won the Visionday 2006 poster prize, sponsored by NVidia. Also a price was won at a competition arranged by Dansk Virtual Reality Selskab, based on an abstract written in the beginning of the project[1].

Lyngby, August 31, 2006

Kristian Evers Hansen

[1] This abstract can be found on the accompanying CD

Acknowledgements

I would like to thank my supervisors, Associate Professor Niels Jørgen Christensen and Assistant Professor Jacob Andreas Bærentzen, for their help throughout the project.

Further I would like to thank the following list of people:

- Assistant Professor Kenny Erleben, DIKU, for given his view of the method and having a good discussion about the possibilities.
- Technical Assistant Thomas Nissen for helping with the scanning of the Boba Fett model.
- Associate Professor Thomas Stidsen for a very nice discussion about search algorithms.
- CTO Steffen Toksvig, Lead Animator Karsten Lund and Lead Artist Tom Isaksen at IO Interactive for taking their time to give their views on the methods, and their applications.
- Last but not least, Eigil Jensen for his help, motivation and company throughout the project and other people at IMM who found time to discuss and help with my project.

Contents

Part I

Introduction

Chapter 1

Introduction

1.1 Brief introduction to computer animation

The history of animation begins in the early 20th century, as hand drawn cartoons are made as animations by showing individual frames fast after each other. This new way of bringing images to life, was made possible due to new medias; the cinema and later the television.

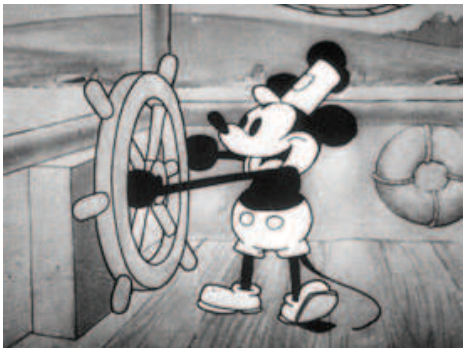


Figure 1.1: Steamboat Willie by Disney, 1928.

Around 1980 computers revolutionized animation, as they gave animators com-

pletely new possibilities. First of all the traditional 2D animation was aided greatly by this new tool, but also the development within 3D computer graphics created a hole new branch of animation: 3D animation. It was now possible to create complete figures and scenes in 3D, not just as seen from a specific point of view in 2D. These figures could be viewed from any direction and be used in any situation, enabling the construction of complex scenes.

Figure 1.2: Luxo Jr. by Pixar, 1986.

Computer animation also began to affect movies. First a few special effects were added, but recently full characters and scenes have been created with a very lifelike appearance using 3D computer animation, replacing sets and real models.

The advances within computer technology has also lead to the introduction of computer games, where animation plays a central role. But here the animations becomes interactive, responding to a users input. This means that the animation must be constructed and rendered instantly.

Figure 1.3: Hitman: Blood Money, by IO Interactive, 2005.

But unfortunately it is still a very difficult task to create animations, requiring specialized programs and skills. This thesis is presenting a 3D animation

method, which makes the process less cumbersome.

1.2 Character animation techniques

In computer games and movies, the most important and also most challenging animations, are those of characters, especially human beings.

When constructing a character animation, for instance a walking person, every frame shown could essentially be created by hand by an animator moving every vertex of the model. This would be a very time consuming process though, especially if it is an animation more than a few frames long. To avoid this, an animation is usually created using interpolation, so the animator only needs to create a subset of the poses, called keyframes, and then the system interpolates between these, creating a fluid motion.

These keyframes and the interpolation can be created using several different techniques:

The simplest interpolation is to interpolate vertex positions of a model in different poses, called *morphing*. This is a simple and highly efficient method, usually done directly on the graphics cards. But as this interpolation is usually linear, the animation is not good, distorting the overall shape between the keyframes.

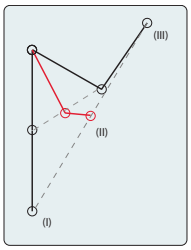


Figure 1.4: (I) and (III) are poses which are interpolated, the resulting (II) is noticeable distorted.

To create better animations, skeleton based systems are commonly used. Skeleton based animation imitates a real body, by having the polygon mesh act as the skin, and a simple inner structure function as a skeleton.

The animator first manipulates the skeleton to define the keyframes. The system can then interpolate the joint parameters in the keyframes, to create a nice motion. This animates the skeleton alone, and for every frame, the system calculates the appearance of the skin, which is the visual product seen by the viewer.

Unfortunately creating animations using this method has some drawbacks, for

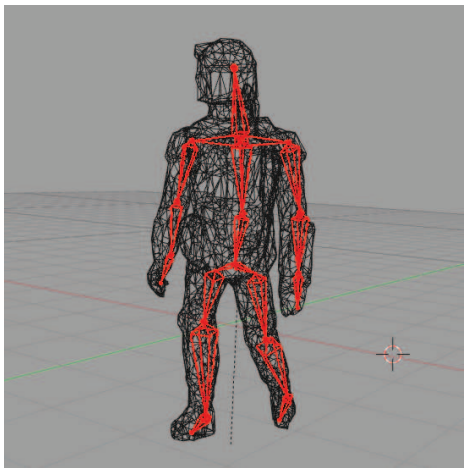


Figure 1.5: A usual bone structure for a character.

example it is a long and difficult process to rig and skin a model, and the animation from keyframe interpolation is not very adaptable. These shortcomings will be discussed more in section 2.1. The method presented in this thesis is meant to improve on these two topics and others.

1.3 Useful Terms and words

Adaptive animation An animation which is constructed on the fly, taking all changes of the environment into considerations. The animation system proposed in this thesis is adaptive.

Affine invariant Affine transformation invariant; when something looks the same independently of any affine transformations applied to it.

Affine transformation Group of transformations containing rotation, scaling, translation.

Animator The person who creates an animation.

Boundary The band of fixed vertices between the region of interest and the rest of the mesh.

Constraint A constraint is attached to a handle. It is acting like a target for this handle, so the system will try to keep the handle at this position.

Example A pose which the animator has created and therefore marked as a good pose.

Forward kinematics Using this technique, the animator alters each joint one bye one to get to a wanted pose.

Free vertices All vertices in the ROI not included in a handle (or boundary) is free vertices.

GPU Graphics Processing Unit, a processor on the graphics card.

Handle A collection of vertices which can be manipulated as one. Usually all vertices in a handle will be treated as rigid.

Inverse kinematics This is the opposite of forward kinematics. The animator defines a target for a given point on the model, and the inverse kinematics system finds the best parameters for all joints for this target to be reached. For example the animator sets a target for the hand of a model, and then the system finds the rotation of the shoulder, elbow and wrist to fit this target.

Keyframe Keyframes are poses created by the animator, which are used by the animation system to create a full animation by interpolating between these.

Laplacian Editing Using the Laplacian coordinates of a mesh to edit it.

Laplacian Reconstruction Term used in this thesis to indicate the reconstruction of the soft surface, given the handle-vertices' positions and the Laplacian coordinates.

Main handle If several handles are selected, one is the main handle. It is the main handle's rotation center which is used for all handles in the particular transformation.

Morphing Making a linear per vertex interpolation between two poses.

Motion Capture Obtaining motion data from motion capturing devices. Markers are placed on an actor, who performs a task, like walking. The position of the markers are then recorded and mapped to a skeleton of a 3D model. This allows for far more complex and realistic animations than an animator could create by hand.

Pose A specific combination of joint parameters, defining a models stance.

RHS Right Hand Side, the part of an equation which is to the right of the equality sign. The $\mathbf{b}$ in $\mathbf{Ax} = \mathbf{b}$.

Rigging Setting up a skeleton on a model.

ROI Region of interest, part of the mesh which the user selects. Only this part is subject to the deformation.

Rotation Center A point in space where a handle should be rotated around.

Secondary handle If several handles are selected, the handles which are not the main handle, are referred to as secondary handles.

Skinning The process of connecting surface vertices to an underlying bone structure.

Soft surface Non-rigid surface that bends under deformation.

Sparse matrix A matrix where the number of zeroes per row greatly outnumbers the number of non-zeroes.

1.3.1 Mathematical notation

Scalar x, lower-case italic letters.

Vector $\mathbf{v} = [x, y, z]$, lower-case bold non-italic letters.

Matrix $\mathbf{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, Upper-case bold non-italic letters.

Set $E = \{a, b, c, \ldots\}$, Upper-case italic letters.

1.4 Models used in the thesis

Figure 1.6: Scanned Boba Fett, 94938 faces (47477 vertices).

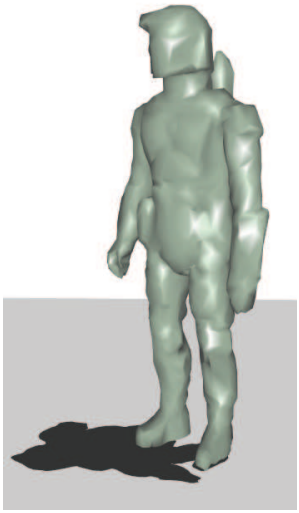


Figure 1.7: Scanned Boba Fett, reduced to 4250 faces (2138 vertices).

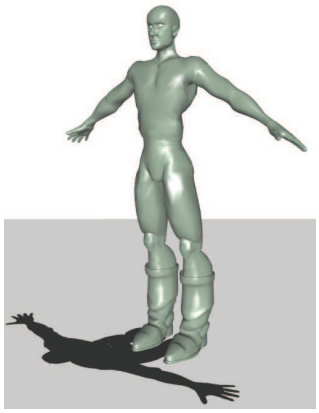


Figure 1.8: Man, 101856 faces (51061 vertices). Curtesy CGSociety.

Figure 1.9: Scanned Armadillo, reduced to 35286 faces (17645 vertices). Curtesy Stanford Scan Repository.

Figure 1.10: Scanned Armadillo, arms lowered, reduced to 17296 faces (8650 vertices). Curtesy Stanford Scan Repository.

Figure 1.11: Orc, 11234 faces (5631 vertices). Curtesy CGSociety.

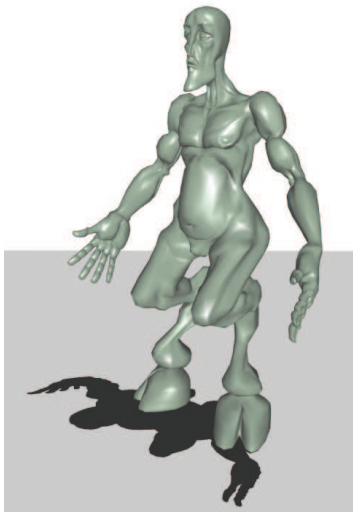


Figure 1.12: Satyr, 25472 faces (12770 vertices). Curtesy CGSociety.

1.5 Thesis overview

The thesis is split up into five parts:

Part I - Introduction Introduction to animation, and to this thesis, with motivation and goals.

Part II - Previous work Presenting all the previous work, which this thesis is based upon, split into two topics: Animation methods and deformation methods.

Part III - Proposed method Presenting the proposed method for deformation and animation.

Part IV - Results Summing up on the results achieved with this method.

Part V - Discussion Discussion of the current results and future possibilities of the methods, ending with a conclusion to the project.

1.6 Reader requirements

To read and understand this thesis, the reader must have a good understanding of computer graphics and computer animation.

A good knowledge of linear algebra and vector calculations is also an advantage.

Chapter 2

Motivation and goals

2.1 Motivation

Creating a good and realistic animation can unfortunately be both difficult and time consuming. Also it usually requires a skilled animator and/or expensive equipment like motion capturing devices, which few people have access to.

The most commonly used animation method, keyframe animation using skeletons, is discussed in the next section.

2.1.1 Keyframe animation using skeletons

The animation method most commonly used, is keyframe animation using a skeleton rigged model. The keyframes are created by an animator using for example inverse kinematics.

It is an intuitive setup and is very cost efficient, and you can usually get the results you want, but unfortunately the method has some drawbacks:

First of all it is a very complex and time consuming process to setup a skeleton

in a mesh; first the skeleton must be constructed using some primitives, and then relations between the vertices of the mesh and the bones of the skeleton must be created. It is the latter step, called skinning, which is the critical part, requiring many iterations of setup and testing to get acceptable results [MG03].

Also skeleton based systems are not useful when animating non rigid objects like a face. So in animations where you have both a character that moves, and changes his facial expression, a combination of skeleton and another type of animation must be used (usually morphing) [LCF00].

Finally animating using keyframes gives a very non-flexible animation. The animations covers some general cases, and does not adapt to minor changes in the environment. Larger changes, like walking onto a staircase, requires a switch to a new animation sequence.

A more thorough description of the workflow when making a keyframe animation with bones is found in appendix D.

2.1.2 Better solutions

There have been attempts to improve on these shortcomings, but no method has had a real breakthrough. New methods are either slow, requires to much preliminary work or simply produces bad results. The methods which works best are mostly extended skeleton systems. In chapter 7 many proposals are discussed.

There is the need for systems where a user, not necessarily skilled in the art of animation, can import a polygonal mesh, and without much preprocessing quickly and easy create some nice dynamic animations.

2.2 Project goals

The goal of this project is to create an example based animation system using a mesh-based skeleton.

It should not be based on traditional skeletons, due to their long setup time and limitations, but should be using some sort of pseudo skeleton structure based on handles on the surface.

The main animation method should not be using keyframes, but rather using poses as examples in a constraint based system.

The system should be relatively easy to use. A person with little animation experience should be able to edit and animate after only minutes of introduction. Of course creating animations true to real life will always require a lot of experience.

The method should be able to handle different types of animation, both large scale character animation like a walk cycle, but also smaller features like facial expressions.

The final system should be able to handle relatively large meshes with interactive frame rates. A number like 20.000 vertices should be doable.

The goal of the project can be split into three steps:

1. Pose creator: Develop method to deform a model, without the use of a traditional skinned skeleton. These deformations, poses, can be used as keyframes or 'good' poses, examples in the animation system.
2. Interactive Pose Interpolator: Create a system, that is able to blend created poses, and use this to construct new poses, from a pose space, by setting one or more handle constraints. The system must find weights for pose-blending to create a new pose satisfying the constraints given. The pose space is defined by the examples of 'good' poses, which the user has created.
3. Animation system: For demonstration and testing purposes, create a small world for an animation to work within. The animation can be created with the interpolation system above, or a simpler keyframe system.

The method should be easy to expand and use with other methods, so for example it could be attached to and work together with a known method like motion capture. It should be noted though, that it is not a goal to implement expansions and integrations, but it should be taken into considerations when designing the method.

2.2.1 Limitations

The animation method proposed in this thesis are not meant to be anatomically correct, or to simulate correct physics. They are just meant to create nice looking

movement, which the viewer perceives as plausible.

The thought was that the proposed method would be ideal for laser scanned models. But these are normally in a very high resolution, and it is not the objective of this project, to be able to edit these huge meshes directly. As long as the systems runs in interactive frame rates with around 20k vertices, it is acceptable.

The model to be animated should be manifold. That is it should not contain holes, open edges, or consist of several meshes[1].

The final program should be user friendly, but the graphical look is not important. It should function as a demo and a test-bench, so it is not necessary for the program to create output which can actually be used in other programs.

2.3 Editing and animation metaphors

This section explains some important terms and metaphors which will be used throughout the thesis.

There are two main metaphors in this report. The first one is for shape editing and the second one is regarding skeleton based animation systems.

2.3.1 Editing metaphor

The editing systems considered in this report consists of three parts; free vertices, handles and boundaries. Together this is referred to as the Region Of Interest, ROI.

Editing consists of the user dragging a handle, the boundary is kept in place, and then the free vertices are deformed depending on the method used.

The user starts by selecting a ROI, which is the part of the mesh he wants to deform. Then he selects a handle (one or more vertices depending on method and the wished deformation) within this ROI, which he uses to define deformations with. Additionally a band of some width is automatically defined as the boundary, between the ROI and the rest of the mesh. This boundary is usually several vertices wide, to create a more smooth transition. On figure 2.1 the

[1] This is possible though in the final implementation, to some extend.

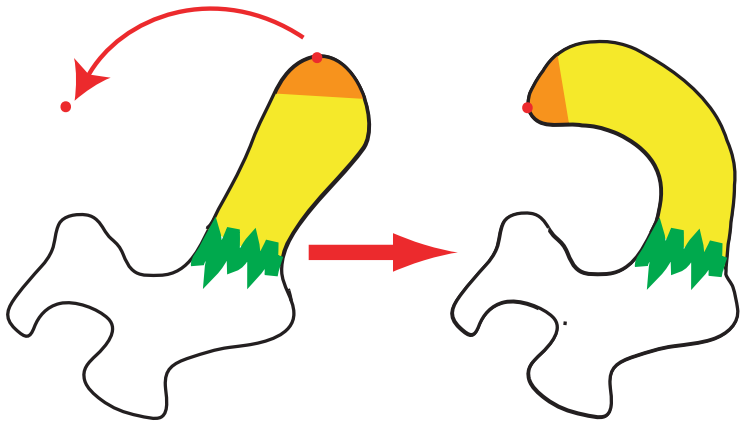


Figure 2.1: Editing metaphor. Orange area = handle. Green area = boundary. Yellow area = free vertices (support). The ROI is all three parts together.

different areas are illustrated. The boundary width, and handle size can vary, depending on the method used, and the wished deformation. For example in space deformation methods, the handle tends to be quite large, up to half the total ROI.

There are two reasons why a ROI is a good idea; first, the user has complete control over, which part of the model the editing will affect. Second, the computation needed to do the editing, can be limited to contain the data for the ROI. This means that the speed of the editing program, is not dependent on the size of the model, but on the size of the ROI. When the ROI and handle is selected, the user can move the handle, rotate it, scale it and so forth, these operations will then affect the whole ROI, and hopefully change the surface in a realistic way.

Unfortunately defining a ROI, which is a subset of the mesh (for example an arm), is not possible when the objective is full-body animation, as the whole model should be able to be deformed at once. So in this project the ROI will be extended to cover the complete model. Some parts can be left out though, namely the inner parts of the handles. This topic will be discussed in section 6.5.1.

Also in the method of this thesis, the ROI will consist of many handles and no boundary. You could say that some of the handles are acting as boundary, but they are still referred to as handles. A rotation center (a joint) is defined for each handle. Around these, the handles can be rotated, and thereby create new poses. These poses can be saved as examples, which is used to create animations.

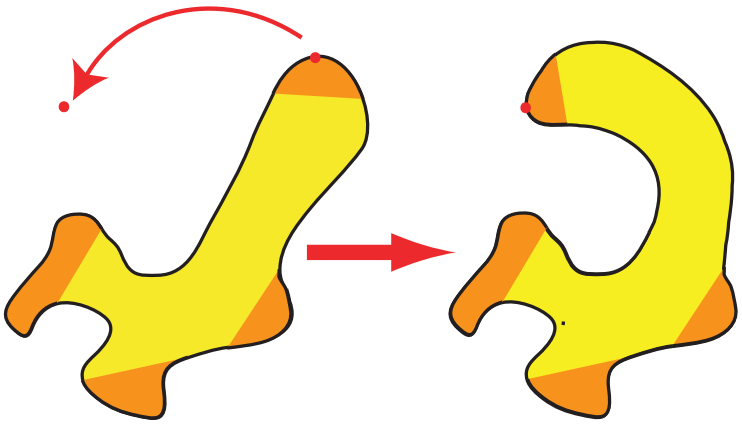


Figure 2.2: Extended editing metaphor for use in the proposed method. Orange area = handles. Yellow area = free vertices (support). The ROI is the complete model.

2.3.2 Animation metaphor

The reference system in this thesis is skeleton-based systems. The system consists of bones in a hierarchy forming a skeleton, and is influencing a mesh called the skin.

The method for creating the influence from bone to skin is usually referred to as skinning.

In these systems, the editing of a model usually consists of the user rotating the bones around a joint (forward kinematics) or the user specifying a wanted position for a bone, and then the system finds all the joint parameters needed (inverse kinematics).

Manipulating the bones creates new poses of the model. A set of parameters describing all the joints positions, defines a pose.

For some animation methods control vertices or handles must be defined, which are then given a target position, a constraint. The system must then find a pose or animation to reach this target position, using kinematics either applied directly to the bone structure or by using some predesigned example poses.

In the method proposed in this thesis, the animation metaphor is an extension to the editing metaphor described above, as it uses this to deform the model. The animation itself is a constraint based system using example poses.

A constraint is a positional target of a vertex or handle on the model, which the pose should satisfy, and an example pose is a user-defined pose, which is marked as a 'good' pose. The animation system then 'learns' from these good poses, when creating completely new poses.

2.4 How to test

Testing the method is very important. In this section, it is described how the resulting method should be tested, to see if the goals of this project has been reached.

The system should include a test bench, where an animation can be tested for a number of parameters and methods. So it is important to have consistency in the following areas:

- Model
- Handle structure
- Pose space (examples)
- Constraints

This will help evaluate different methods and parameters to see which is good and which is not, as the results can only be evaluated using visual inspection.

The performance of the method should also be tested, by timing different parts of the program, and also the scalability, how well it handles larger models.

A comparison with a traditional skeleton based animation should be made, both in terms of setup time, skinning quality and animation output.

Part II

Previous Work

Chapter 3

Animation Methods

In this chapter different ways of creating animations are described. Traditional methods are briefly described, but focus is kept on newer methods trying to solve some of the difficulties from the traditional methods.

If an animation is done by morphing a mesh in different poses (keyframes), these poses can be made with any system capable of deforming a mesh; both traditional editors like Maya, Max and so on and also editing methods described in chapter 4. Then the animation itself is usually a very simple operation carried out on the graphics hardware.

When creating more advanced animation, for example with bones, programs which supports these methods must be used. Programs like Maya, Max and Blender all have this functionality. This will briefly be discussed in the next section.

3.1 Skeleton based animation

3Ds max, Maya and more. In traditional commercial tools like 3Ds max and Maya animation is done using skeletons. The mesh is first constructed, and then the bones are created inside the mesh. The mesh vertices are set up to be

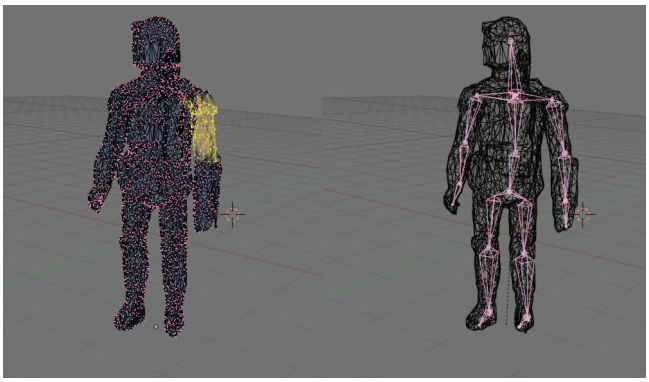


Figure 3.1: (a) Vertices attached to the upper arm is highlighted. (b) The skeleton structure is shown.

under the influence of one or more bones, and each joint between two bones is given some parameters defining its possible movements.

When the whole bone structure is finished, the user can grab a bone and move it. These systems usually uses inverse kinematics for determining other bones movement, due to the movement of a specific bone. For example how the knee and hip is rotated if the foot is moved.

The user then defines keyframes, using this method, and an animation can then be created by interpolating the angles in these keyframes. In practice this is done on the graphics hardware, and is very fast.

For a more detailed walk through of setting up a bone structure see appendix D.

A New Automated Workflow For 3D Character Creation Based On 3D Scanned Data [SJN03] This is a auto skinning algorithm, which uses scans of a model in several poses, mapping a generic model to these poses, and then to find the appropriate weights for each vertex using the scans.

This is a good idea to avoid the cumbersome skinning process, but it still requires having several scans in different poses available.

Building Efficient Accurate Character Skins from Examples [MG03] The basic skeleton method has been extended several times, a resent attempt is Mohr and Gleichers method [MG03]. It uses some mesh based examples, to automatically create some additional joints in the skeleton. These joints are

scaling joints that allows for richer transformation of the mesh than normal bones. This is especially useful for muscle bulges and the like.

This method can create some impressive results, and is very efficient as it uses existing bone functionality. But it still suffers from the setup of the skeleton and skinning which is very time consuming. Additionally it also requires a modeler to be able to create some good mesh poses, using an editing tool.

3.2 Example based systems

Mesh-Based Inverse Kinematics [SZGP05] Mesh-Based Inverse Kinematics, MeshIK, is a new method to create mesh animations, without the use of a skeleton. It is based on having several good poses, examples, of a mesh. Then new good poses can be constructed by a nonlinear blend of these examples. The transformation of each triangle is found, and these are used for a feature vector. A linear system created from the feature vectors and some constraints. When this is solved the blend weight for each example is found.

This method's major strength is to allow the user to create realistic animations without the use of a skeleton. It requires, though, a skillful modeler/animator to create the needed poses in the first place, using some other tool.

The downside of this method, is that you have no way of defining the transformation or path of a part of the mesh. That is the look of the mesh in between different example poses, so the animation can easily be distorted (see figure 3.2).

Figure 3.2: (left) Leg straight. (right) Leg bend. (middle) distorted leg between (left) and (right). Images captured from MeshIK demonstration video.

Style-Based Inverse Kinematics [GMHP04] This method uses example poses as training data to create an inverse kinematics system. The examples are created using some input data, for example motion capture, and not in the system itself.

Given these examples, a pose space is calculated, where every pose exist, but poses close to the examples are preferred.

To function optimal a rich example space is needed, so data from a motion capture session is vital.

The system does not handle the issue about skinning, and so is only working on a skeleton.

3.3 Other

Boneless Motion Reconstruction [KS05] This is a method which directly uses a form of Laplacian Editing (will be described in chapter 4), to transform a mesh accordingly to motion captured data.

Several control vertices scattered over the model are linked with markers in a motion capturing system. When these markers are moved, the control vertices are moved as well, and by reconstructing the surface using Laplacian Editing, the model is deformed to follow the motion capture data.

The major downside with this method is that you will need a complete motion capturing system. These are very expensive, and is not available for everyone. But the idea of using Laplacian Editing to deform the model seems very promising.

Pose Space Deformation [LCF00] This is a method dealing with the issue of having separate methods for animating large scale parts like limbs and small scale features like facial expressions. The method is claimed to combine the advantages of shape interpolation and skeleton driven animation, into one unform method.

It is based upon a traditional skeleton system, but the animator has the possibility to sculpt individual poses, making it possible to create far more realistic animations.

It looks like a great method, but it is still based on the skeleton system, with the difficult skinning.

Animatable Human Body model Reconstruction from 3D Scan Data using Templates [MFT04] This paper describes a complete system to animate a scanned human model. It covers the complete workflow, from filling

holes in the scanned data, to animation.

The system fits a scanned model to a template skeleton, by having the user pinpoint several landmarks on the model.

How the skinning at the joints is handled is not mentioned in the paper. Otherwise the idea seems good, but it requires to have a suitable template skeleton. After having constructed one for a human, it can probably be used for a large variety of human characters, but not for other objects like animals, where a new template must first be constructed.

Skinning Mesh Animations [**JT05**] Another new system, by James and Twigg [JT05], is based on having an existing mesh animation. Each triangle's transformations are then analyzed throughout the animation sequence, this enables the program to group triangles with equal transformation into a bone like rigid structure.

These 'mesh bones', can now be used to transform the mesh animation into a bone animation, which is much more memory efficient, and can easily be modified.

This system is great if you already have an animation, but can not be used to create one from scratch.

3.4 Conclusion

The methods which are expansions to a traditional skeleton system, are not very interesting in this thesis, as an alternative to this method is sought. This is mainly due to the skinning problems, but also the difficulty in controlling smaller and secondary deformations.

Methods to autoskin models using several different poses is not good either, as they requires having access to the model in many poses, or even an existing animation of the model. The system should not be dependant on other factors.

Example based animation ([GMHP04] and [SZGP05]) seems like a very interesting way of controlling the style of the animation. The animator can fairly quickly define good poses by modeling or obtain them from motion capturing, and then the system controls the animation, keeping it faithful to the examples, but still allows a certain flexibility. Because of this, it is what the proposed method in this thesis will use.

Also using Laplacian Editing to deform the surface as in [KS05] is a very good idea, as it is a way of accomplishing deformations without the use of a traditional skeleton structure. This is exactly what is wanted in this project, so this is also used in this proposed method.

In the next chapter Laplacian Editing and similar methods are described in detail.

Chapter 4

Shape Deformation Methods

The primary shape editing method used in this project is Laplacian Editing [Ale03], [LSCO04] and [SLCO04].

Laplacian Editing is based on the Laplacian operator, which is an operator representing a vertex from its neighbors. If for example the laplacian of a vertex is $\mathbf{0}$, it means that the vertex lies in the plane defined by the neighbors. If the laplacian grows, the vertex is then elevated above this plane. This property can be used to represent details on a surface, which is exactly what it is used for in Laplacian Editing. In section 4.1 a thorough description of Laplacian Editing is given.

In section 4.2 other methods which essentially could do the same job as Laplacian Editing is mentioned.

4.1 Laplacian Editing

The Laplacian operator is commonly known as a smoothing operator, used in various smoothing schemes for meshes. In this chapter it is explained how this operator can instead be used for mesh deformations. First a brief introduction to the Laplacian operator is given.

4.1.1 Definitions

Let the mesh $M = (V, E, F)$ be a given triangular mesh with n vertices. V is the set of vertices, E is the set of edges and F is the set of faces. Each vertex $\mathbf{v}_i$ in V is represented in absolute Cartesian coordinates $[x, y, z]$.

4.1.2 Laplacian operator

The Laplacian operator is a second order differential operator in the n-dimensional Euclidean space, defined as the divergence of the gradient. It can be expressed as the sum of the second partial derivatives:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

When working on grids or meshes the above can not be directly used, as a parametrization of the surface is assumed in the above, and on a mesh it is lacking. Instead the Discrete Laplacian, also known as the umbrella operator [DMS99], is used. A full derivation of this operator is found in appendix A.

It is a linear approximation of the Laplacian operator, to be used on meshes and grids. On a 3D mesh it is defined at vertex $\mathbf{v}_i$ as:

$$\boldsymbol{\delta}_i = \mathbf{v}_i - \frac{1}{d_i} \sum_{j \in N(i)} \mathbf{v}_j \tag{4.1}$$

where $N(i)$ is the indices of the vertices that shares an edge with $\mathbf{v}_i$ so $N(i) = \{j | (i, j) \in E\}$. $d_i = |N(i)|$ is the number of these neighbors (the valency).

This means that on a mesh the Laplacian is actually a vector from a vertex to the the center of its neighbors, and can thereby be seen as a detail vector, representing a vertex from its neighborhood.

4.1.3 Variants of the Laplacian

If one changes the length of the laplacian vector, as is done in smoothing, using the above definition introduces tangential drifting of the vertex. This means

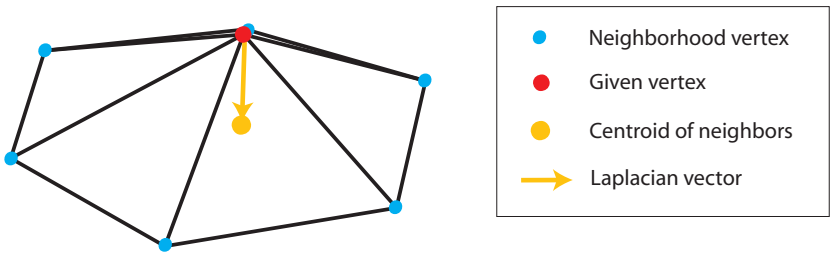


Figure 4.1: Visualization of the Laplacian vector

that the vertex is moving tangential to the plane defined by the neighbors, and this can result in changes in the surface's geometry which is not wanted.

In [Tau95], Taubin proposes a variant of the discrete Laplacian, which takes the edge lengths into consideration. This reduces tangential drifting of the Laplacian:

$$\boldsymbol{\delta}_i = \mathbf{v}_i - \frac{1}{\sum_{j \in N(i)} e_j} \sum_{j \in N(i)} \frac{\mathbf{v}_j}{e_j} \quad , where\ e_j = |\mathbf{v}_j - \mathbf{v}_i| \tag{4.2}$$

On irregular meshes a further advantage can be gained using cotangent weights, as proposed by Desbrun et al. in [DMS99]:

$$\boldsymbol{\delta}_i = \mathbf{v}_i - \frac{\sum_{j \in N(i)} w_j \cdot \mathbf{v}_j}{\sum_{j \in N(i)} w_j} \, , where\ w_j = cot(\alpha_j) + cot(\beta_j) \tag{4.3}$$

This is called the curvature operator, and is based on the gradient of the area of the 1-ring neighborhood. This operator also compensates for unequal face angles, and further reduces tangential drift (see figure 4.2).

Reducing tangential drift makes the Laplacian more stable, especially in cases where it is used for smoothing.

Another useful variant is the squared Laplacian, which is affected by a larger local region, and thereby usually gives better results than the Laplacian [DMS99].

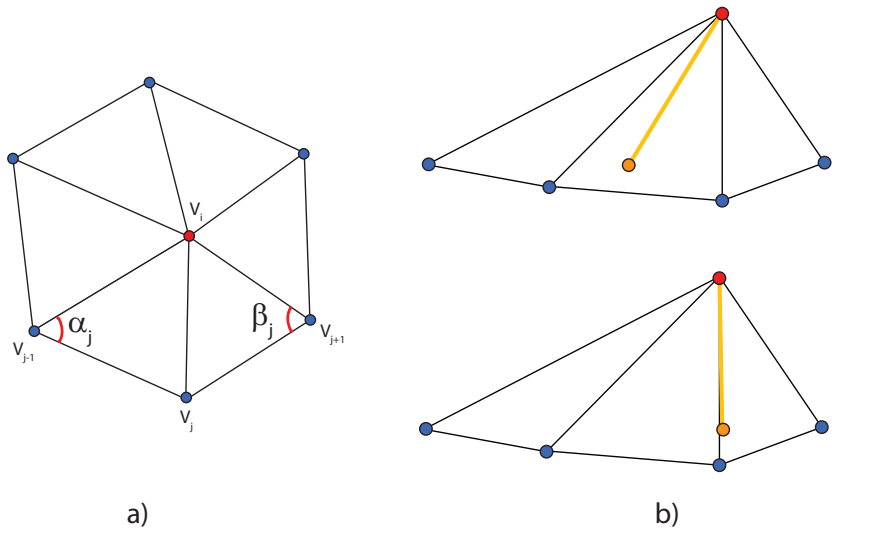


Figure 4.2: (a) The angles used in cotangent weights. (b) Top: Normal Laplacian. Bottom: Using cotangent weights.

But it is also more expensive to calculate.

$$\boldsymbol{\delta}_i^2 = \boldsymbol{\delta}_i - \frac{1}{d_i} \sum_{j \in N(i)} \boldsymbol{\delta}_j \tag{4.4}$$

These variants of the Laplacian is of most use when doing a thing like smoothing, where the Laplacians change. In this thesis only the normal Laplacian and the curvature operator has been used, giving the same result.

4.1.4 The Laplacian Matrix

The $\boldsymbol{\delta}$-coordinates of a mesh, can be expressed in matrix form, so for example, the Laplacian can be calculated for all vertices at once.

In the following, the standard Laplacian is used, but the matrix can be easily extended to support the other variants described in section 4.1.3.

First we need to construct the *topological Laplacian* of the mesh, $\mathbf{L}$ (see also figure 4.3), by using the following rules, which is derived directly from equation 4.1:

$$\mathbf{L}_{ij} = \begin{cases} 1, & i = j \\ -1/d_i, & j \in N(i)(i.e. share\ an\ edge) \\ 0, & otherwise \end{cases}$$

Then this matrix $\mathbf{L}$ can be used in this system:

$$\boldsymbol{\delta} = \mathbf{Lp}$$

where $\mathbf{L}$ is the matrix above, and $\mathbf{p}$ is a vector with the vertices global positions. The result of this multiplication gives a vector $\boldsymbol{\delta}$, which contains the laplacians of all vertices.

4.1.5 Reconstruction using δ-coordinates

The above Laplacian operator is commonly known in smoothing, where the $\boldsymbol{\delta}$ vectors are minimized, which means that you are minimizing the details, and thereby smoothing the mesh. Using the Laplacian Matrix the following system can be used for smoothing:

$$\mathbf{Lp} = \mathbf{0} \tag{4.5}$$

where $\mathbf{p}$ is the unknown vector of vertex positions.

Like this, the system can not be directly used, as all vertices will collapse into a single point. It must either be solved iteratively or some constraints must be added as will be shown later.

In reconstruction, the same technique is used, but instead of minimizing the details, a system is set up that tries to preserve them. Again the the $\mathbf{L}$ matrix is a great help.

To *reconstruct* it, basically the following system must be solved:

$$\mathbf{Lp} = \boldsymbol{\delta} \tag{4.6}$$

where $\mathbf{p}$ is the unknown vector of vertex positions.

The system will try to preserve the $\boldsymbol{\delta}$'s, and thereby the details of the model.

But $\mathbf{L}$ has the rank $n-1$ (n = number of rows/columns in $\mathbf{L}$), and can therefore not be solved directly. The position of at least one vertex must be known, to fix the model's position in global space. So to reconstruct the mesh from the Laplacians, some vertices need to be specified or constrained. These known positions are then added as constraints to the above system:

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{I}_{n\times m} \end{bmatrix} \mathbf{p} = \begin{bmatrix} \boldsymbol{\delta} \\ \mathbf{c}_{1:m} \end{bmatrix} \tag{4.7}$$

where m is the number of constrained vertices, $\mathbf{I}$ contains 1's at the positions of the constrained vertices, the $\boldsymbol{\delta}$ is the differential coordinates, and the $\mathbf{c}$ is the positions of the constrained vertices.

This is of course also true for smoothing a mesh, so equation 4.5 is extended to the following:

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{I}_{n\times m} \end{bmatrix} \mathbf{p} = \begin{bmatrix} \mathbf{0} \\ \mathbf{c}_{1:m} \end{bmatrix} \tag{4.8}$$

When adding the constraints to the system, it gets over-determined, and has to be solved in a least squares sense: $\mathbf{L}^T\mathbf{L}\mathbf{x} = \mathbf{L}^T\mathbf{b}$.

4.1.6 Deformations using δ-coordinates

Looking at figure 4.3, it is easy to see how this system can be used for deforming a model: If one of the red constrained vertices is moved, and it thereby changes its value on the right hand side of the system, the solution will try to interpolate this changed constraint, as well as the unchanged constraints. And when the system is still trying to preserve the $\boldsymbol{\delta}$ coordinates of the mesh, and thereby preserving the details, you achieve a deformation of the mesh, as close to the original as possible, but fulfilling the new constraint, and thereby deforming the mesh.

This leads to an extension of equation 4.8:

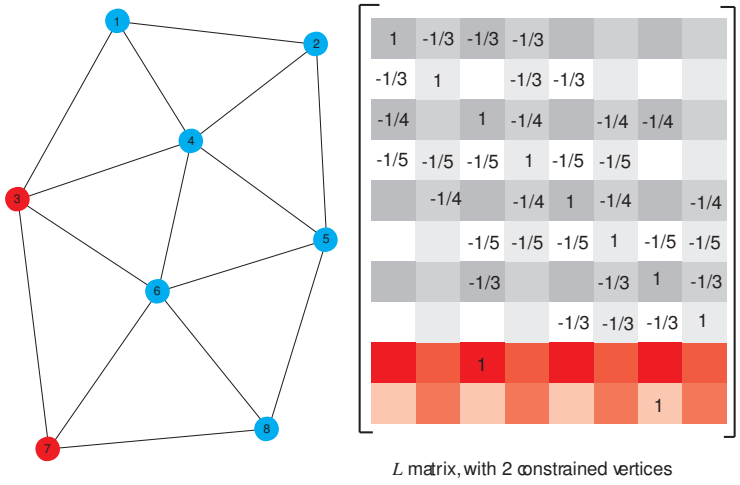


Figure 4.3: Simple Laplacian Matrix. Two red vertices are constrained.

$$\begin{bmatrix} \mathbf{L} \\ \mathbf{I}_{n\times m1} \\ \mathbf{I}_{n\times m2} \end{bmatrix} \mathbf{p} = \begin{bmatrix} \boldsymbol{\delta} \\ \mathbf{c}_{1:m1} \\ \mathbf{e}_{1:m2} \end{bmatrix} \tag{4.9}$$

where $m1$ is the number of constrained vertices, $m2$ number of edited vertices and the $\mathbf{e}$ is the positions of the edited vertices.

Moving vertices from their original positions, will of course induce errors into the system, but as a least squares solution is found, it help spread this error across the surface.

4.1.7 Detail correction methods

The one major problem with this method for deformations, is that the detail vectors, $\boldsymbol{\delta}$, are represented in global coordinates, and are therefore not by default, invariant to rotation and scaling as can be seen on figure 4.4. Some

translations even produce rotations of the mesh, which is not handled by the default Laplacian reconstruction (see figure 4.5).

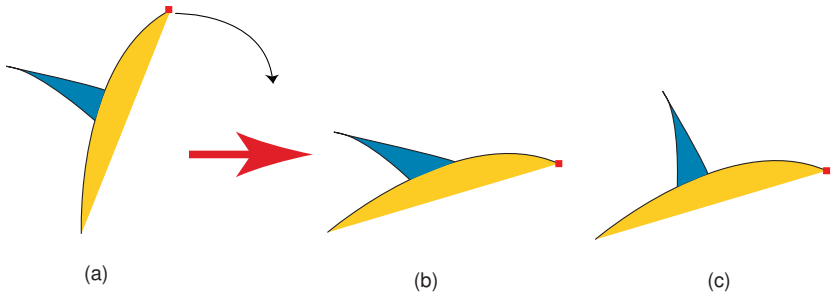


Figure 4.4: (a) A spike (blue detail). (b) The spike has the same global orientation as in (a), and thus is not transformed correctly, when the surface is rotated. The correct result is seen in (c).

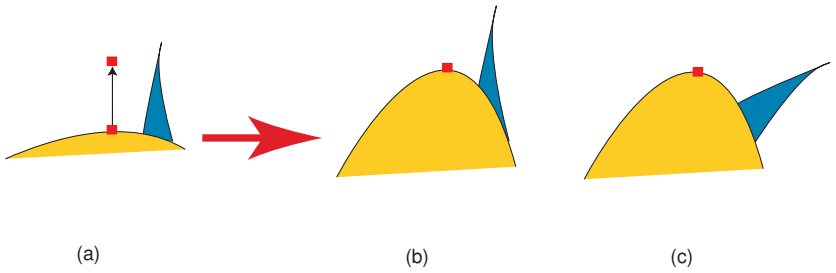


Figure 4.5: (a) A spike (blue detail). The red point is translated up. (b) The spike has the same global orientation as in (a), and thus is not transformed correctly, when the surface is rotated. The correct result is seen in (c).

The solution to this problem is to somehow rotate the laplacian vectors according to the base surface. How this should be done, is still an open research topic, and below is first described the method used in this project, and then several other proposals.

4.1.7.1 Lipman 04 [LSCO04]

This is the method used in this project. It is a good and simple approach, where the detail vectors are rotated explicitly. When edited, the surface is then reconstructed from these rotated differential coordinates.

The rotations of the differential coordinates, are done by estimating the rotations of local frames on a smooth version of the mesh.

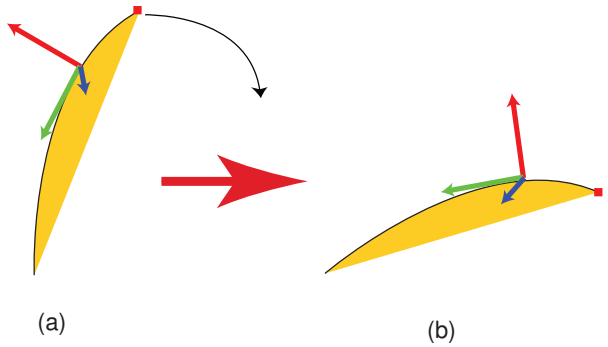


Figure 4.6: (a) Frame at a vertex. (b) Rotated surface, with rotated frame.

The local frame at vertex $\mathbf{v}_i$ is defined as $\{\mathbf{n}_i, \mathbf{u}_{ij}, \mathbf{n}_i \times \mathbf{u}_{ij}\}$ where $\mathbf{u}_{ij}$ is a unit vector obtained by projecting one of the edges from $\mathbf{v}_i$ onto the plane orthogonal to $\mathbf{n}_i$ (the vertex normal).

The simplest way of accomplishing this following method: Solve the Laplacian system with 0's instead of the $\boldsymbol{\delta}$ coordinates in the $\mathbf{b}$ vector. This creates a smooth surface between the handle and the boundary. The normals are now calculated, and the local frames are constructed from these.

The Laplacian of the given vertex, $\mathbf{v}_i$, is now projected onto the three axes of the frame, and these projections are stored. This way the Laplacian is represented in a local frame.

Now the position of the handle is changed, and by solving the system again with the right hand side set to 0's, the new orientation of the local frames can be found, and thereby the rotated laplacians, by using the lengths that were stored before.

1. Solve with RHS set to $\mathbf{0}$.
2. Calculate original laplacians in respect to the local frames of the smoothed mesh.
3. Move the handle.
4. Solve with RHS set to $\mathbf{0}$.
5. Calculate the local frames at each vertex.
6. From the local frames of the transformed smoothed mesh, calculate the rotated laplacians.

7. Solve using the rotated laplacians on the RHS.

The above method requires both an extra solve of the system, and then construction of the frames, which involves calculation of the smooth normals. So it is actually quite costly to do.

Another method is to estimate the smooth normals, from the original surface. This is done by averaging the normals of the original surface in some neighborhood of around vertex $\mathbf{v}_i$. Also the neighborhood normals are weighted by some weighting scheme, where Lipman et al propose the polynomial $p(t) = \frac{2}{r^3}t^3 - \frac{3}{r^2}t^2 + 1$, where r is the radius of the support of the averaging and t is the distance from $\mathbf{v}_i$ to a vertex $\mathbf{v}_j$. This distance should be the geodesic distance, but as this is costly to compute, Dijkstra's algorithm which finds the shortest path along the mesh edges can be used.

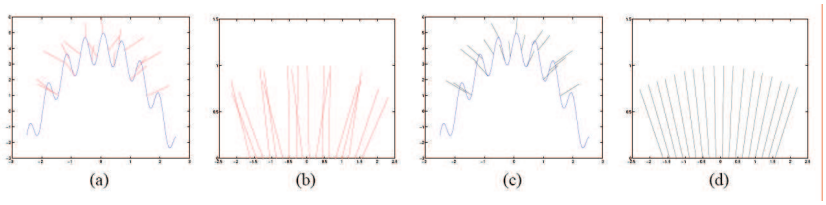


Figure 4.7: A 2D example of smooth surface normals estimation. (a) and (c) show the surface with details and the estimated normals of the underlying smooth surface. In (a) a naive averaging of the detailed surface normals was used. (b) shows the same normal vectors as in (a), but the y coordinate of the origin point of each normal is set to zero. This visualizes the problem of the naive estimation the resulting normals do not vary smoothly. In (c) we show the result of normals estimation using weighted average. As demonstrated in (d), such estimation leads to more smoothly varying normals which are closer to the real smooth surface normals. (figure borrowed from [LSCO04])

When the smooth normals are estimated, they can be used to find the rotations, and these can be applied to the laplacian coordinates[1].

4.1.7.2 Other methods

Several people has proposed other solutions to this problem, below is a description of the most prominent ones.

[1] This method has not been implemented.

Yu et al 04 [YZX04] It works by applying a transformation to the gradients of each triangle in the ROI, based on its geodesic distance from the handle. So close to the handle, the surface receives about the same transformation as the handle. The farther away, the smaller part of the transformation is applied to the surface, until it reaches the boundary, where no transformation is applied. This suffers from needing explicitly defined rotations of the handle, as it can not find rotations from a translation (see figure 4.5).

Sorkine 04 [SLCO04] Here the basic idea is to compute an appropriate transformation T_i for each vertex $\mathbf{v}_i$ based on the unknown deformed surface. These transformations are calculated together with the new vertex positions in a system of linear equations. This method is reportedly very good, but is a bit slower than [LSCO04] and from the paper it is difficult to understand all details, and thereby difficult to implement it.

Lipman 05 [LSLCO05] is based on a new differential coordinate representation, and should give totally affine invariant coordinates. This should give the best possible preservation of details, even under large transformations. It consists of two discrete forms; one that is represented by the edge lengths and the angles between the edges of the one-ring, projected onto the tangent plane, the second is the signed distance from the one-ring vertices to the tangent plane. This is a bit complicated, but basically it is a better way of representing a vertex from it s neighbors, without the use of global coordinates at all. When reconstructing the mesh after a editing action, two system of linear equations is solved: One that reconstructs the local frames of each vertex, and then one to reconstruct the actual vertex position from this local frame. This method has one big flaw; It requires the handle to be rotated explicitly, it can not find rotations from a translation (see figure 4.5)

Which method is best is debatable, and it is highly dependent on the manner the system should be used in. The simplest method is undoubtedly the method from Lipman 04 [LSCO04]. It is easy to implement, reasonably fast and delivers good results as long as the deformations are kept relatively simple. For these reasons it is this method which has been chosen for this project, but there is no problem in replacing the method in future works.

4.1.8 Solving the Laplacian System

The reconstruction of the mesh surface using Laplacian Editing by solving the Laplacian system, relies heavily on linear algebra.

When working with the Laplacian system described in the previous chapter, it is the system $\mathbf{Lp} = \mathbf{b}$ that has to be solved. But when the constraints are added, the system becomes over determined, and thus has no exact solution, and a least squares solution must be found. So the least squares system is $\mathbf{L}^T\mathbf{Lp} = \mathbf{L}^T\mathbf{b}$.

4.1.8.1 Solving by factorizing and back substitution

Systems based on solving the Laplacian system, are usually using factorization and back substitution as solving method, as it is by far the fastest method for this particular problem [BBK].

Factorization and back substitution is a direct solving method, where the $\mathbf{L}$ matrix of a $\mathbf{Lp} = \mathbf{b}$ system is first factorized into triangular matrices, $\mathbf{F}_1$ and $\mathbf{F}_2$, which can be used to solve the system very fast. (see equation 4.10 below for an example of a system using a lower-triangular matrix.)

In Laplacian Editing the handle and boundary is usually defined once, and then a lot of editing is done using these. This means that as long as the handle and boundary is not re-defined, the $\mathbf{L}$ matrix is the same. So while editing with a given handle and boundary, the factorization of $\mathbf{L}$ can be reused, and only the highly effective back substitution is performed at each editing step.

$$\begin{bmatrix} f_{11} & 0 & 0 & 0 \\ f_{21} & f_{22} & 0 & 0 \\ f_{31} & f_{32} & f_{33} & 0 \\ f_{41} & f_{42} & f_{43} & f_{44} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{4.10}$$

With LU factorization for example, the factors $\mathbf{F}_L$ (lower-triangular matrix) and $\mathbf{F}_U$ (upper-triangular matrix) is calculated so that $\mathbf{F}_L\mathbf{F}_U = \mathbf{L}$.

Then it is the system $\mathbf{F}_L\mathbf{F}_U\mathbf{x} = \mathbf{b}$ that has to solved. This can be done in two steps:

1. $\mathbf{F}_L\mathbf{y} = \mathbf{b}$
2. $\mathbf{F}_U\mathbf{x} = \mathbf{y}$

Each of these steps can be performed very fast, because in each matrix, $\mathbf{F}_L$ and $\mathbf{F}_U$, there is a row with only one non-zero element. This 'row-equation' can be solved as a simple equation with only one variable. This solution can then be

used to calculate a solution for a row with two elements in the factor, and so forth. This is called solving by *back substitution*.

There exists several of these factorization algorithms, but for these kinds of problems, LU ($\mathbf{L} = \mathbf{F}_L\mathbf{F}_U$), Cholesky ($\mathbf{L} = \mathbf{F}_L\mathbf{F}_L^T$) and QR ($\mathbf{L} = \mathbf{F}_Q\mathbf{F}_R$) are the most useful.

QR is slowest but most numerically stable, and Cholesky is the fastest, but least stable [BBK]. Due to its speed Cholesky is used in this project.

4.1.8.2 Sparsity

A sparse matrix is one where the number of non-zeroes is a lot less than the number of zeroes. When a matrix is sparse, it is possible to do some extremely efficient calculations on it.

In the Laplacian system, when the number of vertices in the mesh is large, the $\mathbf{L}$ matrix becomes very sparse. An element $\mathbf{L}_{ij}$ is only non-zero if vertex $\mathbf{v}_i$ is a neighbor of $\mathbf{v}_j$ or that $i = j$.

For example if the mesh has 10.000 vertices with an average valency of 6, $\mathbf{L}$ has 10.000 rows and columns, with only 7 non-zeroes in each row and in each column, so only 70.000 non-zeroes out of a total of 100 million positions. Notice that in figure 4.3, the matrix is not sparse, due to the small number of vertices.

Even when solving $\mathbf{L}^T\mathbf{L}\mathbf{x} = \mathbf{L}^T\mathbf{b}$, the matrix $\mathbf{L}$ is still sparse. An element $\mathbf{L}^T\mathbf{L}_{ij}$ is only non-zero if the two vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ *share* a neighbor. This is not as sparse as before, but still sparse with approximately 19 non-zeroes per row.

Also in this form of Laplacian Editing we have extended the $\mathbf{L}$ matrix to include information about the constraints. This is an additional row per constrained vertex, where the corresponding position of the vertex contains a weighted constant. But even with this extension, both $\mathbf{L}$ and $\mathbf{L}^T\mathbf{L}$ are sparse.

The sparsity of the system can be utilized. There are several software packages available, which are optimized for sparse systems, and they can perform factorizations and back substitutions very fast.

4.1.8.3 Other methods - multigrid algorithms

In [SYBF06] Shi et al. proposes a multigrid algorithm to solve large systems of linear systems, instead of a direct factorization.

The methods main forces, are that it does not require the long factorization time, as the direct solvers, and it is also a lot more memory conserving. It is still slower than a direct factorization if solving several times though, due to the fact that in these types of problems, you can reuse the factorization, and only do the back substitution at each step. Also it should be mentioned that, at this years Siggraph conference the applicability of this method was questioned by others, due to a problem with convergence.

The reason it is mentioned here, is that it is possible with a multigrid solver, to turn down precision and gain speed in return. This sounds as an interesting possibility, and should be examined in future works.

4.1.8.4 Conclusion

In this project the factorization and solving by back substitution is used, as it fits the problem really well; the system is factorized once, and then in each editing frame only back substitution is done. Cholesky is chosen as method as it is the fastest and it does not seem to have stability issues in this type of problem.

When solving by back substitution, each axis is solved separately, so the back substitution is performed 3 times with different vectors on the RHS. At each editing frame, the back substitution is actually performed 6 times; 3 times for the smoothing step, and 3 times for the reconstruction.

This is achieved by using a solver package, TAUCS [TAUCS], available on the internet. It is a sparse solver, and it is very fast.

4.2 Other editing methods

In this section a brief introduction to alternative editing methods is given.

Multiresolution editing

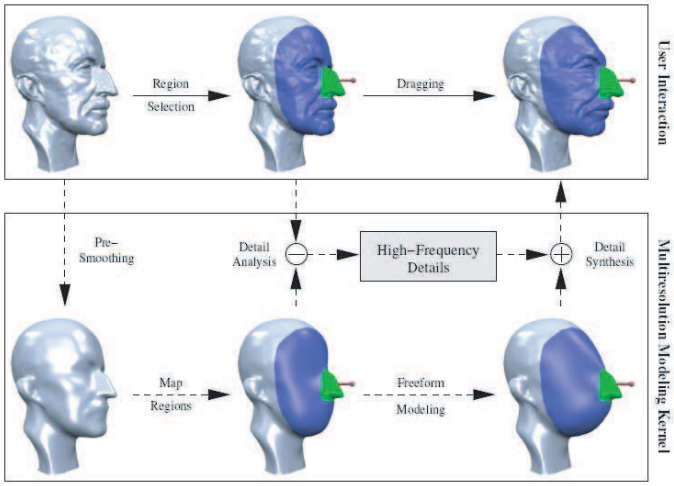


Figure 4.8: Multiresolution: Computes a low-frequency base (bottom left) for the input surface (top left). After selecting ROI and handle on the original surface (top center), these regions are mapped to the base surface and high-frequency details are encoded (bottom center). Moving the handle changes the base surface (bottom right), adding the detail information back, results in a correct deformation (top right). (figure borrowed from [BK04b])

The basic idea, is to construct a smooth base mesh from the original mesh, and then add the details as offsets vectors.

The base mesh is created by smoothing the original mesh, to a point where it does not contain any details. The lost details are then encoded in the corresponding positions on the base mesh.

Usually this encoding of details is done in frames, defined locally at each vertex. This means that the details are not represented in global coordinates, but solely in relation to the base mesh. This makes it straightforward in regards of preserving details, and it is invariant under rotation and translation by nature, because if the surface rotates, the local frames rotate, and then the detail-vectors follows, being encoded in these frames.

Editing is done on the smooth base mesh, where a handle is moved affecting the ROI. To deform the ROI accordingly to the transformation of the handle, an optimization problem is solved, satisfying the fixed vertices (handle and boundary) and keeping the surface of the base mesh smooth. Now, on the deformed mesh, the local frames are recomputed, and the details, represented in these, are added to the base mesh.

The disadvantage of this method is that the details have to be defined explicitly; the mesh has to be smoothed, and then the connection between the removed details and the base mesh has to be made. And sometimes many levels of detail may be required, to get a proper result in meshes with complex details. That is if the difference between the original and the base mesh is more than just offsets of the vertices, then several levels are required.

This type of editing is mainly developed by Leif Kobbelt, who in [Kob98] and [Kob99] describes a technique to perform multi-resolution editing, using smoothing to create the base mesh, and encode the detail levels locally in face-based frames.

In [BK04a] and [BK04b], Botsch and Kobbelt brings multi-resolution editing up to date. First by including a freeform modeling metaphor as described in section 2.3.1, and then by making the method even faster by first remeshing the base-mesh, and then by using some of the newest Linear Algebra packages (like TAUCS) to solve their systems.

Realtime editing using Radial basis functions Botsch and Kobbelt presents

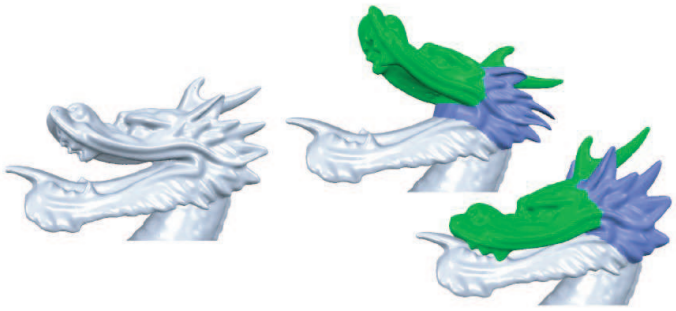


Figure 4.9: Space deformation: Opening and closing the mouth of the Dragon. This model contains holes and degenerate triangles. (figure borrowed from [BK05])

a method in [BK05], that uses radial basis functions to make space deformations.

The idea is to move a handle, constrain a boundary, and then let all vertices between these to areas, be transformed by a function that interpolates the transformations of the handle and the stationary boundary. This interpolation function is created using radial basis functions.

This has been done before, but one of the main improvements over earlier works,

is that Botsch & Kobbelt uses r^3 as basis function. Using this global function, gives a much nicer result, but it also slows down the calculations a lot compared to simpler functions. To compensate they present several optimizations, and with these they have been able to greatly improve the speed of their system:

They use an incremental QR (IQR) solver, to get an approximating solution, instead of the interpolating you would get by solving it directly. It speeds up the calculations hugely, with almost no visual evidence.

They also found a method to pre-compute the basis function, by utilizing that it is known, that only the handle is transformed, and the boundary is fixed.

With the IQR solver and the pre-computed basis function, the bottle neck ended up being updating the mesh; recalculating the normals or tangent axes (for point based models). So they implemented these functions for mesh updating, on the GPU.

When the above improvements is implemented, their results and timings, shows an extremely fast system, capable of deforming several hundred thousands vertices at interactive frame rates. For example it should be able to pre-compute a ROI of 880k vertices in 16s and each editing action in 0.030s ([BK05] table 1).

The speed of this method is making it very interesting, but it has big problems with the orientation of the details in some cases. Also how it will handle a more complex region of interest with many handles is an open question.

Dual Laplacian Editing for Meshes [**ATLF06**] This is yet another new proposal to solve the orientation problem for the detail vectors. Their contribution consists of two parts; The first is the realization that using a direct solver to reconstruct the surface in a single step is not going to produce the best results, as the orientation of the detail vectors are dependent on the resulting underlying surface and vice versa. So they propose an iterative method to approach the final solution in smaller steps. Their second contribution is to perform all this on the dual mesh ([Tau01]). This gives a mesh structure which is much more uniform, where all vertices have a valency of 3, and should therefore give much more stable results.

This of course sounds very good, especially the idea about using the dual mesh to correct bad meshes, but their problem lies in the iterative solver. It is probably good to create poses, but in interactive applications using large meshes, a single step of a direct solver is slow enough already. Iterating between 10 and 50 times will make the system just as many times slower, so unless working with smaller meshes it is not an option.

PriMo: Coupled Prisms for Intuitive Surface Modeling [BPGK06] A

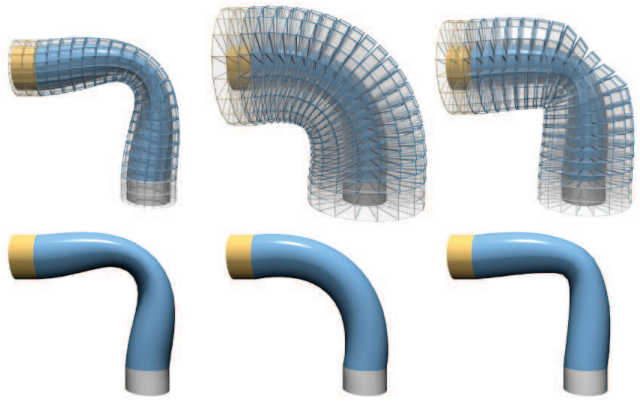


Figure 4.10: Height of the prisms controls surface stiffness and thereby the bending of the model. (figure borrowed from [BPGK06])

new and very interesting method for 3D shape modeling that claims to achieve intuitive and robust deformations.

It emulates physically plausible surface behavior inspired by thin shells and plates, by extending the surface to volumetric prisms, which are coupled through non-linear, elastic forces.

To deform the mesh, prisms are rigidly transformed to satisfy user constraints while minimizing the elastic energy. The rigidity of the prisms will prevent degenerations even under very large deformations.

The parameters of the elastic energy can be controlled by the user, who thereby can control how the surface should deform.

It is a much more computational demanding method than the other works, but it offers an improved robustness and the ability to handle very complex deformations.

Part III

Proposed Method

Chapter 5

The idea

In this thesis a new animation framework is proposed.

To deform meshes, a handle structure is used as a form of mesh-based skeleton and Laplacian Editing is used as a skinning method.

This setup can be used to obtain different poses of a model, which can be exported, used in keyframe animation or used in the animation system proposed here.

The animation is done by using an example based inverse kinematics method, which 'learns' good poses from predefined example poses created by the user.

5.1 Overview

The very first step in the pipeline is to create the model, which is to be animated. This could either be done by using a traditional modeling tool like 3D Max, but the main force of this method is dense plain models obtained from a 3D scanner.

When having scanned the model, holes must be filled and errors corrected. This is usually accomplished using software bundled with the scanner.

This thesis will not cover this process other than a description of a scan workflow in appendix C.

The main workflow an animator goes through after having obtained the model is:

1. Setup of the handle structure
2. Create the example poses
3. Animating / posing

5.2 Setup of handle structure

The handle structure is the backbone of the system. It is acting as a mesh-based skeleton, which the user can manipulate. The structure consists of handles painted onto the surface of the model, each with a user-defined rotation center. The handles are created individually, with no explicit connectivity between them.

After the model is loaded, the first step, is for the user to create this structure. Each handle should be created so it contains all vertices in a rigid area. Then the rotation center is positioned. It is around this that all rotations of this handle is centered, so it is working as a joint.

This will be explained in detail in section 6.2.

5.3 Create poses

The user can manipulate the handles using forward kinematics, either one at the time, or bundle them together, to create the desired pose.

Also the user has the option of using a developed feature called *Detail Deformation Layer*, which enables details like muscle bulges and wrinkles, to be sculpted for a given pose.

Several of these poses should be created as examples to be used in the animation step.

This editing process will be explained in section 6.3 and 6.4.

5.4 Animation

There are two different animation systems implemented in this method: A simple keyframe animation and the main method using examples and constraints to create a sort of inverse kinematics system.

The keyframe system is using a simple animation scheme to interpolate from one example pose to the next. This is useful to test a created pose sequence, or if the goal is simply a keyframe animation.

As discussed before, keyframe animation is not flexible. If for example you have a keyframe animation to enable a character to walk straight, you would need to create a totally different animation if it should walk up a stair, and then switch between these.

To create a more flexible system, another type of animation must be used. An example based dynamic animation system is a good alternative, as it avoids physically based constraints by letting the animator define a space of good poses.

In the example based system, the user adds some positional constraints to one or more of the handles of the model. The system then uses the examples created by the animator, to create a new pose which fits the constraints best possible.

The example system also contains functionality to add paths to the constraints. This way their positions becomes time dependent and thereby it is possible to create a moving real time animation.

How these systems are constructed is explained in chapter 7.

Chapter 6

Shape Deformation System

6.1 Introduction

First step towards animation, is to be able to deform a mesh. In this chapter, the shape editing system which is responsible for this, is explained.

Laplacian Editing as described in section 4.1, is a simple but powerful way of deforming arbitrary meshes, and therefore it is the method chosen to be one of the core elements in the proposed method.

The way Laplacian Editing is used however, is slightly altered. In previous works, the handle and boundary is relatively far apart, and the larger inner area is then deformed. But when deforming for example a character, large parts of the model should remain rigid, and only parts around the joints are usually deformed.

So here the usual Laplacian Editing metaphor is extended, to include several handles, covering the rigid parts of the model as described in section 2.3.1. These can then be manipulated, either one at the time, or in groups, to achieve the desired pose.

The next section will cover the aspects of the handle structure, and then the

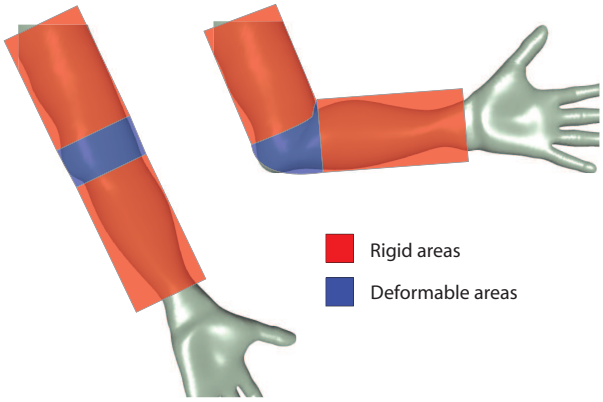


Figure 6.1: Only blue part around the joint needs to be deformed. The red is rigid.

deformations using Laplacian Editing is explained. After this, exactly how the deformations of the handles are done is explained, and finally a section containing performance considerations is presented.

6.2 Handle Structure

The idea behind the handles in this project, is that the user can manipulate them to control the deformation of the model. So a handle for each deformable part must be created. As described in the previous section, the handles should cover the large rigid areas of a model (as seen on figure 6.1). So the handles are actually used to define which parts of the mesh is rigid and which is not, and thereby determines how the mesh deforms.

There are several other important considerations regarding the handle structure and the handles themselves: They should be easy to create, easy to manipulate, but they should at the same time contain and store enough information to create good animations in the end.

6.2.1 A Mesh-based Skeleton

The handle structure used in this proposed method, as a replacement of a traditional skeleton, is represented directly on the surface. This means that there

is no need of an extra geometric structure to represent any bones.

The user simply selects the vertices to be part of a rigid handle by painting directly on the surface. This makes a very user friendly and fast setup process.

The only thing needed for each handle is a rotation center, a joint, which the user must place. When the handle is affected with a rotation, its rotation center is used to define a rotation axis for the rotation[1].

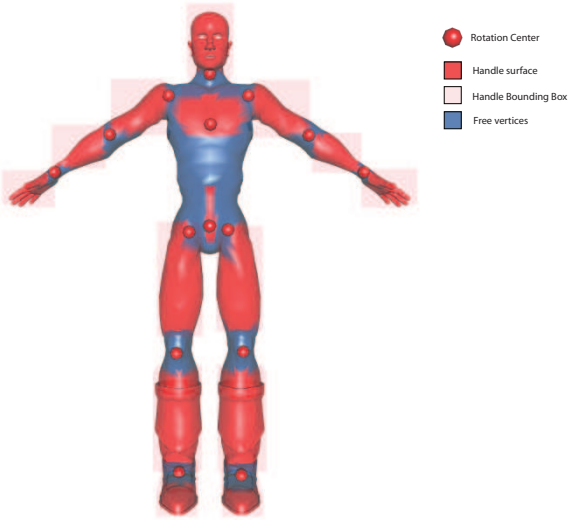


Figure 6.2: An example of a handle structure set up on a regular character model.

When the handles are defined for the rigid parts, and the rotation centers are functioning as joints, the structure is beginning to resemble a traditional skeleton structure. Although a traditional skeleton was not wanted, they still have some good features like being able to imitate a real body, and therefore is very intuitive to use.

But compared to skeleton based systems, this method has one huge advantage; there is no need for skinning. This 'mesh-based skeleton', the handle structure, is painted directly on the surface, and thereby there is no need of a connection between a skeleton and a mesh. When skinning in a traditional system, there lies a huge amount of work in connecting all the vertices with the correct bones, to make them deform in a natural fashion.

[1] Unless several handles are selected, will be explained in section 6.4

The handles are also more flexible than a skeleton, as they are not directly connected, they can be created wherever there is a need for one, and they can be affected one by one or in a group.

6.2.2 Flexible handle hierarchy

If a human model is rigged with a handle structure in the traditional bones fashion, then it should not be possible to edit the upper arm without the lower arm follows. This problem can be solved using a handle hierarchy, defining a set of secondary handles for each handle.

But such a fixed hierarchy is perhaps not always wanted. For example if the handles are setup to make animations like facial expressions, it should be possible to edit the handles one by one.

The chosen solution is to have the user define a hierarchy at each deformation. So the user selects a main handle, then he select the secondary handles which should be subjected to the same transformation. When doing a deformation, the transformation is added to each handle.

To gain the advantage of a fixed hierarchy (a faster selection process), the user is given the possibility to save a set of secondary handles for each handle. So when the user selects a handle, he can choose to select just that handle or to also select all stored secondary handles. This speeds up the process of selecting, especially if a deformation requiring many handles are wanted.

When editing multiple handles at once, it is important that things like rotation centers, and rotation axis, are transformed along with the handles. This topic of rotating parts will be discussed more in section 6.4.

6.2.3 Size of handles

The user is left with a choice on how large the handles should be. Larger handles is increasing the frame rate as the surface that needs to be reconstructed by Laplacian Reconstruction is reduced (this will be explained in section 6.5.1). Also with large handles like in figure 6.2, the user has better control of where the deformation should occur. Still there must be enough space between the handles to ensure a decent transition (see section 6.3).

If a smaller handle is chosen, only just marking the rigid areas, more of the

surface is defined by the Laplacian Editing, and is then not rigid. Also the speed of the system will be affected negatively as the Laplacian system grows.

Using small handles has the advantage of creating a more smooth transition between the handle and the free parts, as there is more surface to work with so to speak. But this also requires a lot of the model which is being edited; it must contain some natural bending places in the joints. If these areas are not well defined, the deformation will be spread across a larger area, resulting in a less realistic deformation. This is illustrated on figure 6.3.

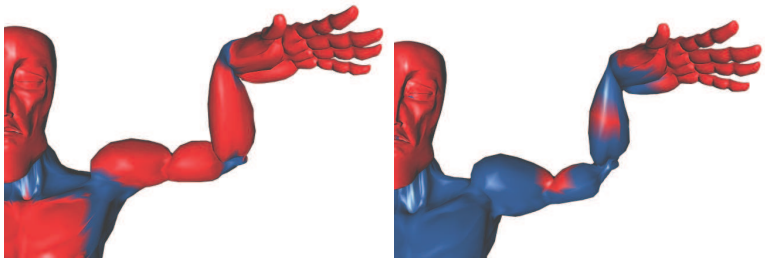


Figure 6.3: Left: Large handles. Right: Small handles. It can be seen that with well-defined large handles results are better than when using small, even though this model has well defined deformation zones (narrow at joints).

6.2.4 Weighted handle vertices

The method also supports assignment of weights to individual handle vertices. So that a weight of 0 makes the vertex act as if it was not part of a handle, when the weight is increased the vertex becomes more and more rigid until it reaches 1 which is the default; totally rigid. This gives the user possibility to fine-tune the deformation in a certain area.

It is not necessary for the user to use this feature, as the results will usually be fine without, but it gives the user more possibilities if needed to achieve a desired effect.

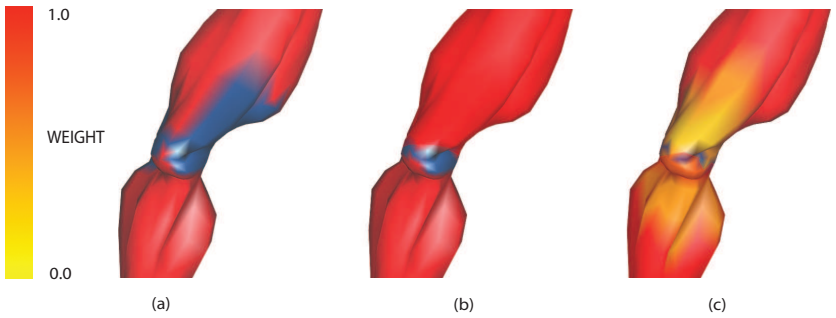


Figure 6.4: From left to right: **1.** After box selection **2.** Fine painting the handle **3.** Changing weights of handle vertices (yellow is less than 1.0).

6.3 Deforming soft surface

One of the main objectives in this project was to avoid the difficult skinning process present in traditional skeleton based animation. But still, the transition areas between the handles, the soft surface, have to deform in a realistic fashion.

The vertices, which are part of a handle, are deformed by any transformation made to the handle structure, so this part of the surface is defined directly by the transformation. But the handle structure does not offer a solution to the free vertices between the handles.

A simple solution would be to make all the vertices part of a rigid handle, but this would give some very bad stretching and self-intersections around the joints (figure 6.5a).

So a band of free vertices are needed between the handles, which is responsible for creating a smooth transition between the handles (figure 6.5b).

In a skeleton based system these vertices would be given weights by the user, to several bones to make a realistic smooth blend. But by using the Laplacian Editing framework this can be managed automatically.

As explained in section 4.1.2, reconstruction using the Laplacian coordinates, can deform the free vertices of a region, given the positions of some boundary vertices and the edited positions of handle vertices.

This conforms with the handle structure of this method, using the transformed

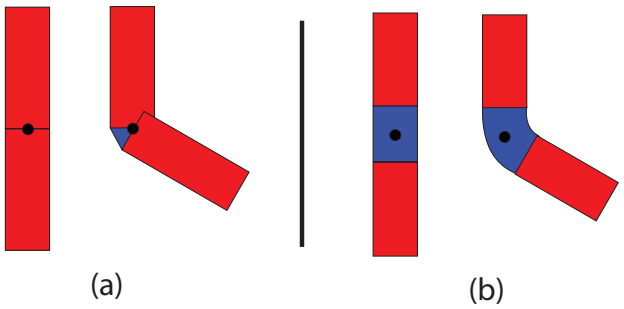


Figure 6.5: **(a)** Surface only consists of handles (red part). **(b)** A band of free vertices (blue part) are placed between the handles. **(b)** gives a much better deformation (right)

handles as handle, stationary handles as boundary [2], and all non-handle vertices as free vertices in the laplacian system.

The Laplacian system is created using this setup, and when solved the result gives the deformed positions of the free vertices. This method gives a realistic transition between the handles, where details on this surface is preserved best possible when deformed.

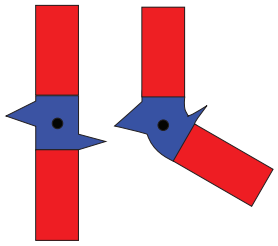


Figure 6.6: Details present in the free region, is preserved under deformation.

In this project Laplacian Editing as proposed in [LSCO04] is used (as concluded in section 4.1.7). Several other methods could have been used, but this particular method offers a simple solution, which performs well and is relatively fast to implement.

This method to reconstruct the free vertices from the position of the handles, will be referred to as Laplacian Reconstruction in the following chapters.

[2] In practice there is no difference between boundary and handle vertices, so the handles are treated the same, no matter if they are transformed or not. Their indices are used in the Laplacian matrix, and their positions are used in the right hand side vector.

6.4 Deformations

In the above section it was explained that by transforming the handles and thereby the vertices in the handles, and thereafter using Laplacian Reconstruction to deform the free vertices, it is possible to create a new pose for a model. In this section it is defined how the deformations are done.

The user must first activate the handles he wants to manipulate, by first selecting a main handle, and then some optional secondary handles[3]. For example if the whole arm should be raised, the main handle should be the upper arm, and the secondary handles would be the lower arm and the hand.

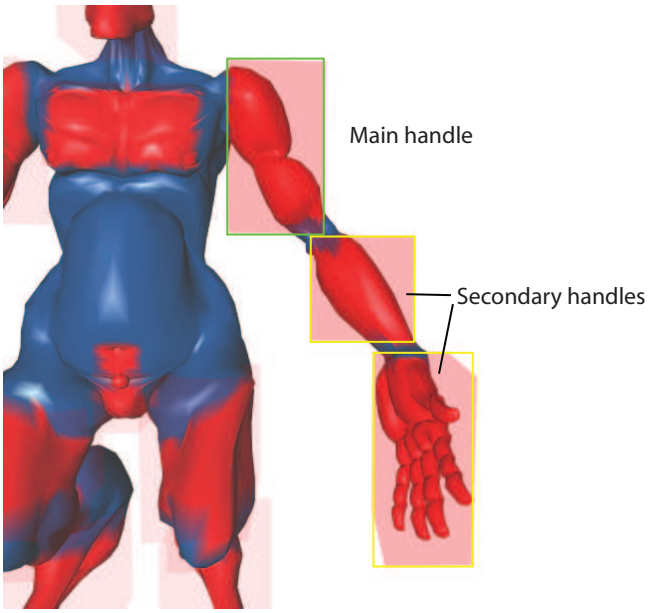


Figure 6.7: Example of main and secondary handles.

When the handles have been activated, they can now be manipulated by dragging them. All actions take place in a plane parallel to the screen. Possible actions are translation and rotation.

Rotations are done using an axis orthogonal to the screen, going through the rotational joint of the main handle. It is visualized in figure 6.8 and the method is described in detail in algorithm **1**.

[3] This is due to the lack of a fixed handle hierarchy, see section 6.2.2

Translation is simply moving the handle in the plane parallel to the screen in a direction corresponding to the input of the user.

The vertices in the handles are affected directly by the transformation created, and then the soft surface is reconstructed using Laplacian Reconstruction, ending up with a new pose with nice and smooth transitions between the handles.

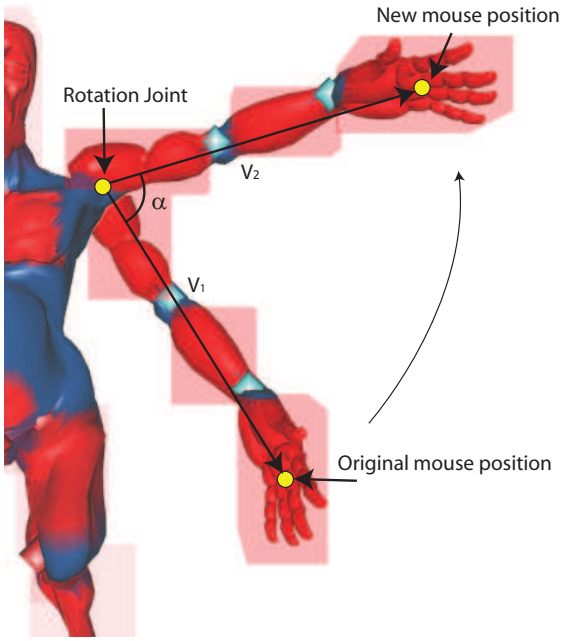


Figure 6.8: The vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ is found and the rotation is created from these.

Algorithm 1 calc_rotation

Require: $mousePos, mainHandleId$
$oldPos = getHandleCenter(mainHandleId)$
$rotationCenter = getHandleRotationcenter(mainHandleId)$
$converttoScreenCoords(oldPos)$
$oldPos.z = getScreenZ(rotationCenter)$
$converttoWorldCoords(oldPos)$
$newPos = mousePos$
$newPos.z = getScreenZ(rotationCenter)$
$converttoWorldCoords(newPos)$
$v1 = oldPos - rotationCenter$
$v2 = newPos - rotationCenter$
$Q = makeRotation(v1, v2)$
$rotateActiveHandles(Q, mainHandleId)$ // function rotating handles using a Quaternion.

6.4.1 Detail Deformation Layer

Inspired by *Pose Space Deformation* [LCF00], a functionality to sculpt the surface of the individual poses has been included in this method.

It has not been not fully implemented, as it was not the main objective of the thesis, but a proof a concept method has been made to test the possibilities.

The Detail Deformation Layer is an extra layer of information in each pose, giving the user an additional possibility for editing smaller details. It allows the user to add an offset in the normal direction for each vertex on the surface. This is useful for muscle bulges, wrinkles and other surface artifacts which can be encountered when animating.

These offsets acts as an independent layer, and is not affecting the calculations on the handle structure, or the Laplacian Reconstruction. It could probably be more efficient if incorporated better into these systems, but as it is a proof of concept method, it was a priority, that it did not alter the results of the original method.

Acting as a separate layer, it requires attention regarding the normals, as the layer will affect the way the model should be lighted. This means that the optimizations regarding lighting normals (coming in section 6.5.2) can no longer be applied, and the normals must calculated from scratch. Maybe a solution for

this can be found in future work.

This method requires though, that the surface is relatively detailed, as it is not possible to add additional vertices.

6.4.2 Saving the transformations

The poses the user creates must be saved as example poses for use in the animation or later retrieval. This section is about what should be saved.

First question is: Should the parameters of the transformation be saved, or is it enough to save the specific transformation matrix?

Just saving the transformation matrix would sure be nice, as it is compact and simple. But this will make it difficult to interpolate between poses later as it is not easy to linear combine transformation matrices, though it can be done. Usually it is done using spherical linear interpolation of quaternions (SLERP) or linear interpolation (LERP), but they are not perfect: SLERP is lacking commutative abilities, and LERP does not ensure constant angular velocity (meaning that 0.5*Rotation is not actually half of the rotation) [BBM]. In [Ale02] Alexa proposes a new method to combine transformations using an exponential map which looks promising, but is relatively slow.

Fortunately the problem can be simplified, as we know exactly what the user did, transformation-wise, to create a given pose. This means that there is no need for spending time finding the rotation between two positions, as it is already given. So instead of interpolating matrices, the specific rotation angles and translation distances can be interpolated. This means that all the steps the user took should be saved as transformation parameters, to create a transformation history, which will help give a very efficient interpolation later (will be discussed in section 7.2), with the same qualities as SLERP; constant angular velocity and following the best path. Unfortunately, like SLERP, the method is not commutative, meaning the order of rotations can influence the outcome (more on this in section 7.2.2 and 7.5.6).

When a handle is rotated or translated, the parameters of this transformation are saved. For rotation the parameters stored are:

- Rotation axis
- Rotation angle

- Rotation center used
- List of handles affected

and for translation:

- Translation vector (direction and distance)
- List of handles affected

The specific coordinates of the rotation center for the given transformation can not be saved as it is dependent on any transformation applied before. Instead the id of the rotation center is saved, this makes it invariant to other transformations applied.

The rotation axis is also dependent on previous transformations, so it is defined relatively to the handle, by representing it in a local frame created at each handle.

It should be noted that these stored transformations are not specific for the given model, but can be applied to any model, as long as it is using an equal handle structure, and the models global orientation is the same. This is very useful for transferring animations from one model to another.

When storing the parameters themselves and not just the transformation matrices, it becomes very easy to blend examples. This is the topic of section 7.2.

6.5 Performance optimizations

Although performance has not been the main objective in this project, several solutions have been considered or implemented to improve on this area.

6.5.1 Laplacian Reconstruction

One of the main problems in using Laplacian Reconstruction as skinning, is the time it takes to solve the huge systems of linear equations. It is relatively fast, but it is still the bottleneck of the system by a large margin. Several solutions has been considered to optimize this part.

One consideration was that during editing, the quality of the mesh should not be of high priority. It could be acceptable to work on a decimated mesh, or only solve the system for a subset of vertices, and reconstruct the rest using a more efficient method.

But at the moment the system can handle more than 20k vertices in an acceptable frame rate, so it was not a priority to implement such a multi-resolution system, as it would complicate the system further.

Another optimization idea has been successfully implemented; the position of the vertices, which are part of a handle, is actually calculated twice: First time when the handle is transformed, and the second time when the surface is reconstructed using Laplacian Reconstruction. This is not optimal, but the handle vertices can not be taken completely out of the equations, as they help solve the Laplacian system (section 4.1.5).

But at least the number of handle vertices used in the system can be greatly reduced. The reduction should not be random, but should leave a band of vertices at the handle edge, connected to the free vertices. This will remove all the vertices in the inner handles, and leave the edges, which should be enough to solve the system. See figure 6.9 for illustration.

This reduction is done in the preprocessing step, all handle vertices are examined, and a narrow band of handle vertices are found at the edge of each handle.

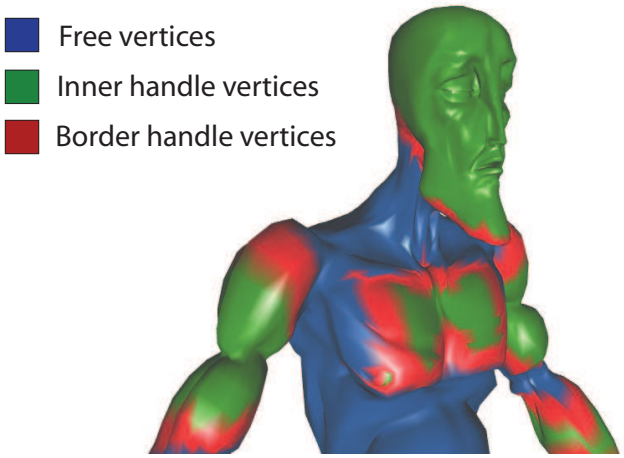


Figure 6.9: The green part of the surface can be left out of the calculations.

First a width of one vertex was tried, but it made the surface very unstable, and

resulted in a malformed object as shown in figure 6.10. When increasing the width to two vertices, the stability was immediately restored, and the result was as good as when using all vertices. In certain cases, a wider band of vertices are needed to avoid unwished deformations of the mesh, but when using a standard handle setup as in figure 6.10 2 is usually enough. Note that if weights smaller

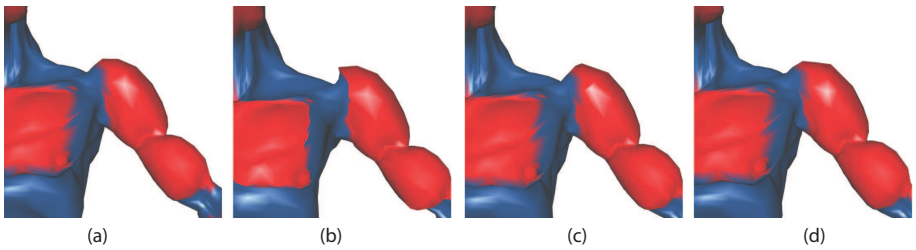


Figure 6.10: (a) Original method. All handle vertices used. (b) 1 vertex width band. Notice the shrinkage of the blue surface. (c) 2 vertex width, almost no difference. (d) 3 vertex width, no difference.

than 1 are used on handle vertices, these vertices must be included in the system as well. So it is an advantage having large totally rigid areas in the inner parts of the handles.

The reduction in number of handle vertices used, has two advantages; First the $\mathbf{Lp} = \mathbf{b}$ system is reduced accordingly, which can be as much as a factor 4, and second the time consuming process of calculating the smooth frames (section 4.1.7) of each vertex, need only to be run on the used vertices. For example, using a model with 12770 vertices, with the normal method we would get a frame time of approximately 0.25s and when using a 2-vertex boundary as handle, this was reduced to around 0.084s (table 6.1). A reduction of almost 70%.

NR	Description	Matrix size	Frame time
1	All	12770x22925	0.244s
2	Band 3 vertices	4803x6991	0.095s
3	Band 2 vertices	4197x5779	0.084s
4	Band 1 vertices	3465x4315	0.069s

Table 6.1: Frame times for different band-widths (12770 vertices in model)

6.5.2 Normals

When deforming a model, the normals of the surface changes. These must be updated to provide the user with an acceptable view of the deformation, as these

	Description	Frame time	Calculate normals
1	No normal updating	0.088s	-
2	Calculate all normals	0.193s	0.105s
3	Rotate ROI normals	0.157s	0.069s
4	Rotate all normals	0.095s	0.005s

Table 6.2: Timings on normal updating (12770 vertices in model, 4197 in ROI)

are used to calculate the lighting of the model.

A straightforward solution is of course to calculate the normals at each change, but this is unfortunately very slow, as it first must find all neighboring faces, calculate these face normals, and then average these into a vertex normal.

An idea for optimization was to use the newly rotated Laplacian vector, as it approximates the normal direction. A problem with this is that it is unknown wether the laplacian is pointing inwards or outwards, and also it is not always a good enough approximation.

A better solution, which has been implemented, is that the normals for the ROI vertices (free vertices and the handle vertices used in the calculations, see section 6.5.1), are not recalculated directly, but are rotated using the local smooth frames, together with the Laplacians as described in section 4.1.7.1. Notice that the smooth normals for the local frames must still be calculated, the optimization is only valid for the detailed surface normals used for lighting.

A further improvement is to improve the updating of the normals of the inner handle vertices, which is not included in the ROI vertices, as well. But as these are not included in the linear system, a smooth frame cannot be found easily. The best idea for these vertices, is to use the transformations applied to the handle, to transform[4] the normals as well. This has also been implemented; it is very fast, and works well.

In table 6.2 the progress in the cost of updating normals are presented. It shows a huge performance boost, by rotating the normals together with the local frames and the handles. The cost is reduced to under 5% of the cost to calculate them, and is only taking around 5% of the total frame time.

[4]Only rotation is applied to the normals, as translation does not affect them and scaling has not been implemented.

Chapter 7

Animation System

In the previous chapter, it was explained how the proposed method handles deformations of a model. Using this method, example poses should be made, and these can now be used to create animation. How the examples are used is the topic of this chapter. It consists of six sections:

Introduction This section contains a list of definitions, and explanation of important terms.

Blending A section on how blending of poses is achieved.

Example based animation A description of the example based animation system proposed here.

Objective function Discussion on what objective function describe a good pose.

Optimization method Discussion on how to create a function for optimization of the objective function.

Animation How it is put together to create animation.

7.1 Introduction

The overall idea is that the example poses created by the suer, acts as 'good' poses, which the system can learn from, to create a larger pose space. The poses in this new pose space is all possible blends of the example poses, where each pose can have a weight between 0 and 1. An objective function is then constructed to be able to evaluate a pose, obtaining a value for how good it is, compared to the initial examples.

When the pose space is created, the user can add positional constraints to the handles, and then the system will search the pose space for the best suitable pose, to satisfy these constraints.

The main inspiration for this method is Mesh-based Inverse Kinematics, *MeshIK* [SZGP05], where example poses can be interpolated and combined. *MeshIK* has two main disadvantages both originating from working on individual triangles: It is slow, and blended poses can appear very distorted as shown on figure 3.2.

But as explained in chapter 6, the proposed method in this project is based on handles, which each consist of a bundle of rigid vertices. The handle structure can then be seen as a graph, where each handle is a node, containing a part of the mesh. This can be used to improve on the shortcomings of *MeshIK*.

In *MeshIK* the poses are defined by the transformation of individual triangles, but if the poses instead was defined by the transformation of the nodes in a graph like the handle structure, a pose will have a much smaller dimension. This will lead to a simpler pose space, and a search in this pose space will be many times faster than presented in *MeshIK*.

Using the handle structure will also help create more realistic deformations, when blending poses (in contrary to *MeshIK*, see figure 3.2), as it defines the rigid and soft areas, and thereby the deformation zones.

So this idea has been utilized to create an example-based animation system without the downsides of *MeshIK*.

7.1.1 Definitions

Symbol	Description
p	Pose.
P	The pose space, poses defined using the examples $\mathbf{E}$.
e	Example pose. Consisting of a list of transformations.
E	Set of example poses. All examples the user has created.
h	Handle. A collection of vertices, with a rotation center.
H	Set of handles on the model.
c	Constraint. Target constraint for a handle. A position in space.
C	Set of active constraints.
w	Weight of an example.
W	Set of weights, usually one for each example.
t	Transformation defined by parameters and affected handles.
k	Weighting constant, used to apply weighting to an energy term.

Table 7.1: Definitions for use in this chapter

7.1.2 Set of examples

As the method is example based, an animation needs a collection of examples to work. These are created using the editing method described in chapter 6. Which examples and how many should be created, depends on what kind of animation is wanted.

If for example a walk animation is needed, it is best to create the examples to define the usual walk cycle on figure 7.1, [WALK1] and [WALK2]. Such a set can be extended though, for example with poses defining how to walk on stairs. It is best not to add examples which could conflict with the others, as this can lead to unwanted animation results.

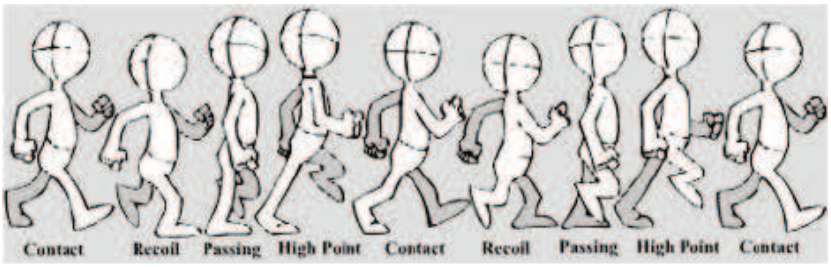


Figure 7.1: Walk cycle, figure borrowed from [WALK1].

If one wants a more general animation, with possibilities to create all kinds of animations, another possibility is to create examples of the extreme poses of the different limbs. For example; Leg all forward, leg all back, leg to the side, leg bent at knee. These 4 examples should be enough to define all positions a leg generally can have. So for all 4 limbs (arms and legs) this means 16 examples, plus eventual examples to control the head, torso, feet and hands.

This procedure is much more general than the walk cycle above, and as such requires more examples, which means longer setup time and longer search time. Also it can be much more difficult to achieve nice results, because the good poses are not clearly defined. This problem will also be mentioned in the next section *Constraints*.

Examples of the two types of example sets can be seen on figure 9.10 (set A, a sequence) and figure 9.11 (set B, extremes).

7.1.3 Constraints

The animation system is using constraints on some of the handles, acting as target positions. The user activates constraints on a handle and move the target. The system will then update the pose of the model, to fit this constraint as good as possible.

A constraint for each handle can be set, but the strength of this proposed method is that it is not needed for all handles. Using the examples and some objective function described in the section 7.4, one or two constrained handles can many times be enough to define the specific pose.

If the examples are created as extreme poses (see the previous section), it takes more constraints to reach a wanted pose, due to the examples defining extreme poses rather than good poses (see figure 7.2). When the number of constraints increases, the complexity and requirements for the user inputs grows as well. In this case, data like motion captured data, would be a great help.

In general it becomes more difficult, requiring more constraints, to create a realistic animation, the more general the examples are made.

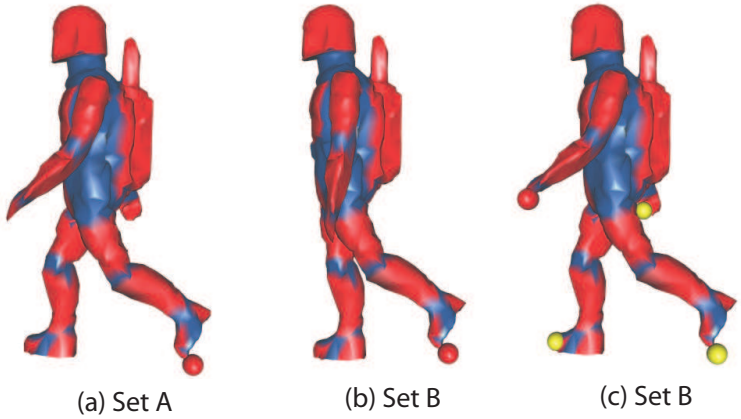


Figure 7.2: (a) With set A (examples of walk poses) it is possible to get a good walk pose using only 1 constraint. (b) This is not possible with set B (examples of limb extreme extends). (c) For set B it takes 4 constraints to get the same result. The example sets A and B can be found in section 9.2.1

7.2 Blending poses

The first requirement to the animation system, is to be able to blend two or more pose examples, to form a new pose.

A simple linear blend between vertex positions is possible, but unless the pose examples are closely related, the overall shape of the mesh can be heavily distorted during rotations, as shown on figure 7.3. As the method should be able to blend arbitrary poses, with no preconditions, a linear vertex blend is not adequate. It would have been sufficient if the only transformation was translation, but in animation rotations are most commonly used.

When interpolating rotations, interpolation of the *angles* of the rotations is wanted. So additional information besides positions of the vertices, are needed for each example. As the user defines a pose by transforming the handles, it is obvious to store these transformations as the examples as discussed in section 6.4.2.

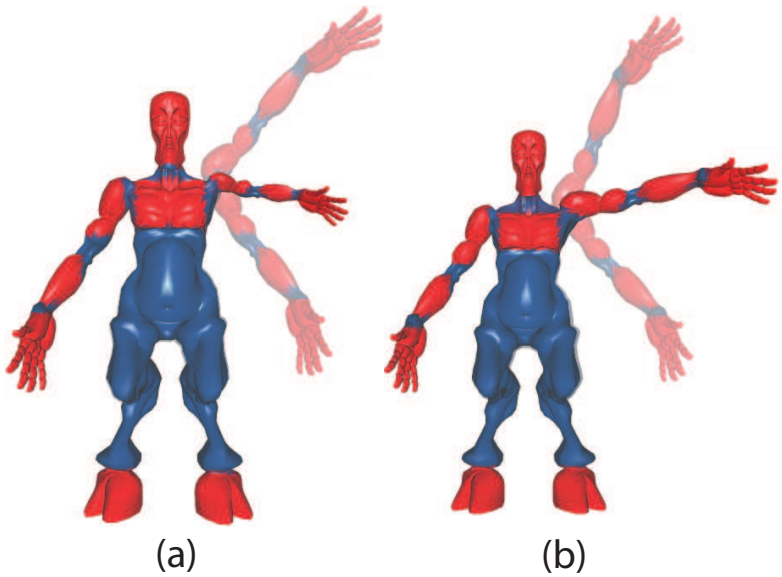


Figure 7.3: (a) Vertex morphing using linear interpolation. (b) Angle interpolation

7.2.1 Loading an example pose

When having all the transformation parameters, it is easy to load an example, and also for example, to load 'half of' of the example, by applying a weight of 0.5. 'Half of' an example is defined with a constant velocity in mind, so half of a translation, is simply half of the translation distance, and half of a rotation, is half of the angle around the same axis.

It is accomplished as shown in algorithm **2** *loadExample*[1].

Each handle has a transformation matrix, **M**, stored, which initially is an identity matrix, meaning no transformation. When an example is loaded, all its transformations are converted to transformation matrices, and multiplied with the matrices of the handles they affect.

The resulting **M** matrix in each handle, can be applied to all vertices in the handles. The rest of the surface can then be reconstructed using the Laplacian Reconstruction method as when constructing poses in the previous chapter.

[1] This code written here, is only for the case of rotation, but translation is pretty much the same.

Algorithm 2 loadExample - only for rotations

Require: *weight* w, *example* e, *handleset* H
  for each transformation t **in** e **do**
    //extract parameters from t:
    $angle = w * t.angle$
    $axis = toGlobalCoords(t.axis)$
    $rotation\_center = H(t.rotation\_id).rotation\_center$
    // create matrix $\mathbf{M}$:
    $\mathbf{R} = make\_rotation\_matrix(angle, axis)$
    $\mathbf{T_1} = translation\_matrix(rotation\_center)$
    $\mathbf{T_2} = translation\_matrix(-rotation\_center)$
    $\mathbf{M} = \mathbf{T_1} * \mathbf{R} * \mathbf{T_2}$
    for each handle h **in** t **do**
      $h.\mathbf{M} = \mathbf{M} * h.\mathbf{M}$
    end for
  end for

Translation is dealt with in the same fashion as rotation above, only difference is that $\mathbf{M}$ is a translation matrix created from the translation vector instead of a rotation matrix.

7.2.2 Loading multiple example poses

With the *loadExample* function above, blending is straightforward: By loading several examples, E, after each other, with some weights, W, the result is blending between these examples. This is done in algorithm *blendExamples*.

Algorithm 3 blendExamples

Require: *weights* W, *examples* E, *handleset* H
  for each e **in** E **do**
    $weight = W(e)$
    $loadExample(e, weight)$
  end for
  for each h **in** H **do**
    $h.apply(h.\mathbf{M})$
  end for

In *blendExamples*, the *loadExample* function is called for each example in the set E.

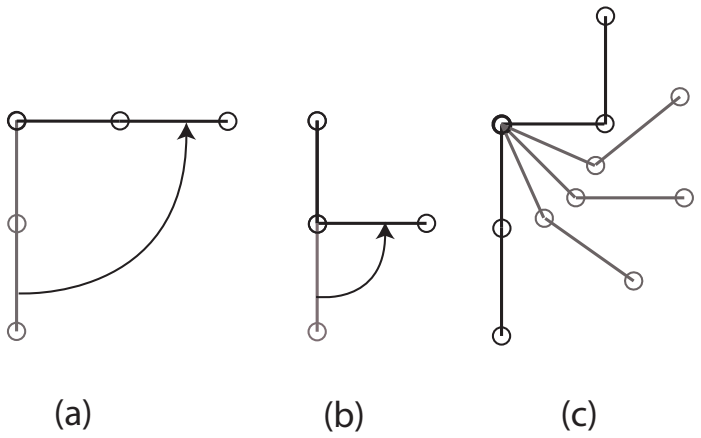


Figure 7.4: (a) Example 1. (b) Example 2. (c) Blending of example 1 and 2

When all examples have been loaded, each handle's transformation matrix is applied to all vertices in the handle, by simply multiplying the matrix with the vertex. Finally the free vertices are reconstructed using Laplacian Reconstruction.

One thing to be careful with, is that the order in which the examples are loaded can have a huge affect on the result, as the interpolation method is non-commutative. Loading the examples e1,e2 is not necessary the same as loading examples e2,e1. This will be discussed more in section 7.5.6.

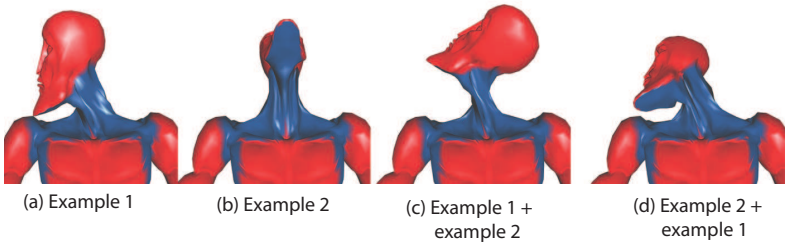


Figure 7.5: (a) Example 1. (b) Example 2. (c) Blending of example 1 and 2 (d) Blending of example 2 and 1, very different from (c).

7.2.3 Detail Deformation Layer

The Detail Deformation Layer as described in section 6.4.1, consist of an offset for each vertex. When loading a example e with a weight w, the offsets in the layer associated with this pose is also applied with the weight w.

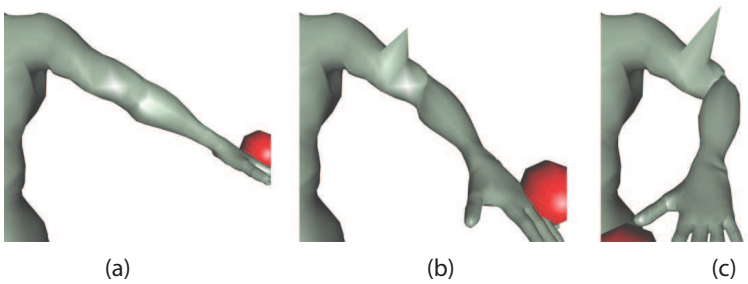


Figure 7.6: Exaggerated example of detail deformation layer: (a) Pose loaded with weight 0.0. (b) Pose loaded with weight 0.5. (c) Pose loaded with weight 1.0.

7.3 Example based animation

Instead of having a traditional keyframe system, which delivers nice but not adaptive animations, it was chosen to develop an example based system.

The animator creates poses, like he would do to a keyframe system, but these poses are not used as keyframes but examples. They define a pose space of good poses, which the system must 'learn' from. This pose space then consist of all possible blends of examples in E.

The proposed method is basically about the system finding a combination of the created example poses, which results in a new pose satisfying some constraints. These constraints can either be interactively set by the user, or by some path system.

The system has two types of parameters to optimize when searching for a combination of examples; weights of the individual examples, W, and the order the examples are loaded (The order is not optimized directly, the reason why is discussed in section 7.5.6).

An effective search algorithm must be derived to search through these parameters, to find the best possible pose, which minimizes an objective function.

But first it must be defined what a good pose is, this means that the objective function must be defined. This will be discussed next.

7.4 Objective function

How to define a good pose is very dependent on how the examples are created; they could be the extreme extends of the limbs, but they could also be more closely related, to create a more compact pose space, all to be used in a walk animation for example.

An objective function has been derived to express an energy as a value for how good a pose is. This function can consist of several terms, these are discussed below.

The main objective is how well the constraints are satisfied, but several other objective terms have been found useful. These can all be part of the energy function, all targeted at creating poses true to the examples created by the user. Seven terms have been found, which can be combined or used alone[2]:

- Minimize distance from target (constraints).
- Control the number of examples used.
- Minimize difference from last pose.
- Minimize the sum of the weights.
- Minimize 1 minus the sum of the weights.
- Minimize number of weights over a threshold weights.
- Minimize difference from examples.

7.4.1 Minimize distance from target (DtT)

The combined distance from the constrained handles to their constraints is the most straightforward term in the energy function. It is clear that the model

[2]The first 'Pose distance from target' should always be part of the function though.

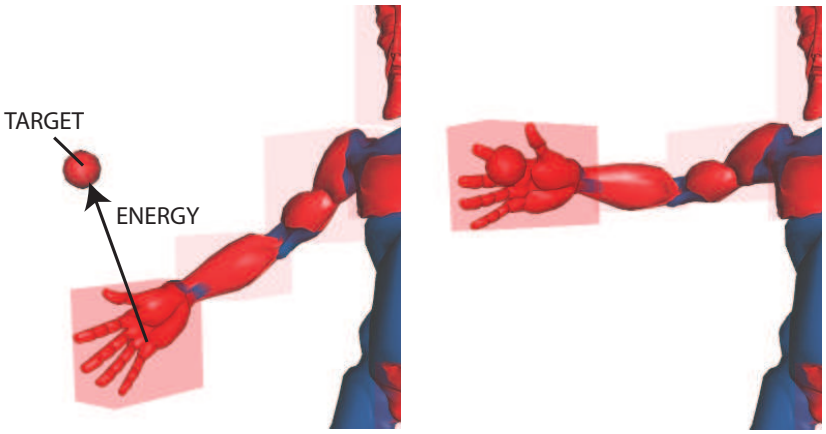


Figure 7.7: Energy: Distance from target

should interpolate, or try to interpolate, the constraints that are set up for one or more handles.

The constraints can easily be set up so that a solution which interpolates all of them is impossible. So instead of selecting solutions which interpolates, the solutions are penalized for moving away from the constraints. This is also known as soft constraints.

This energy term is the most important one in the energy function, and can not be left out. It is defined by:

$$\underset{p \in P}{\operatorname{argmin}} \; \|k * \sum_{h \in H_c} (\|c_h - h.position\|)\| \tag{7.1}$$

where p is a pose in the pose space P, k is a weighting constant, h is a handle in the set of constrained handles H_c and finally c_h is the position of the constraint of h.

7.4.2 Control the number of examples used (NEU)

If the examples are created to form a sequence, it can be desirable to limit the number of examples used for a pose. When doing a sequence, which should perform like a keyframe animation, actually only 2 examples should be in use at any time. This gives a very strict animation, with almost no variation possible.

Unfortunately this also often induce some jumpiness into the animation, and increase the possibility to get stuck in a bad energy minimum. So a higher value is usually better.

The function to control the number of examples used are finding the number of weights exceeding a certain threshold ε. This number is then compared to the desired number, and poses consisting of more than the desired number of examples are penalized.

This term is expressed as:

$$\underset{p \in P}{\operatorname{argmin}} \ \|k * (max(0, (USED(W) - nr_{desired}))^2\| \tag{7.2}$$

where $nr_{desired}$ is a user specified number of examples used, and the function $USED(W)$ is returning the number of weights in W which has a value larger than a threshold ϵ.

7.4.3 Minimize difference from last pose (DtL)

An animation which jumps and twitches is not wanted, and therefore it can be a good thing to limit the variation from one frame to the next.

A combined distance, handle-wise, from the last pose to the new is calculated, and the those poses which does not change much is favored.

An alternative to this energy term, is to incorporate this objective into the search function, and only search in a local area of the parameter space, instead of a global search. This will also limit the difference from one pose to the next. This will be discussed in section 7.5.3.

$$\underset{p \in P}{\operatorname{argmin}} \ \|(k * \sum_{h \in H_p} (\Delta h.position))^2\| \tag{7.3}$$

where H_p is the total set of handles in the handle structure, transformed to the pose p.

7.4.4 Minimize the sum of the weights (SoW)

Two different methods to control the sum of the weights have been examined.

The first try was simply to minimize the sum of all weights. This had the desired effect of eliminating the poses consisting of several large weighted examples, which actually just opposed each other. But it also had some unwanted effects, because it favored the zero weights, and therefor can get stuck at the initial pose of the model.

$$\underset{p \in P}{\operatorname{argmin}} \ \|k * \sum_{w \in W} w\| \tag{7.4}$$

where w is a weight of an example pose, W is the set of weights, one for each example.

7.4.5 Minimize 1 minus the sum of the weights (OMSoW)

A revision of the above method meant that instead of just minimizing the weights, a pose is penalized when the sum of weights moves away from 1. The idea behind this energy term, is to minimize the weights, but at the same time favor combinations which is close to the examples or in between.

$$\underset{p \in P}{\operatorname{argmin}} \ \|k * |1 - \sum_{w \in W} w|\| \tag{7.5}$$

7.4.6 Minimize number of weights (NoW)

One problem was found in the above method; a combination of poses summing to 1 was just as favored as a single pose with the weight 1. The goal should be, that if some constraints can be fulfilled by a single example, this should be the best solution. An extra term was derived to achieve this, namely minimizing the sum of the square roots of the weights. This will make the energy function prefer one weight of 1 over two weights of 0.5.

$$\underset{p \in P}{\operatorname{argmin}} \; \|k * \sum_{w \in W} \sqrt{w} * \frac{1}{\sum_{w \in W} w}\| \tag{7.6}$$

This term can be added to the term above (equation 7.5) to help keeping the poses good.

7.4.7 Minimize difference from examples (DfE)

Another approach to match the examples, is to take a more geometric approach, by looking at the handle's geometric positions, instead of the weights of the examples. So this energy term tries to minimize the sum of the squared distance from the tested pose to the example poses. This is done using a gaussian function to evaluate the distance from the handle positions in the tested solution to all examples (their handle positions as well):

$$G(x) = e^{\frac{-x^2}{\rho^2}} \tag{7.7}$$

Then summing over all examples in the set:

$$\underset{p \in P}{\operatorname{argmin}} \; \|k * \sum_{e \in E} |(1 - G(dist(p, e)))|\| \tag{7.8}$$

where e is an example in the example set E and $dist$ is a calculated per handle distance.

This method is effective when the examples is more of a sequence, and not just extreme cases, where you rarely want them to resemble the specific poses, but rather a blend with small weights.

7.4.8 Conclusion

All the above energy terms have their use, but usually only 2 or 3 terms should be used at one time. Which terms to use is dependent on which animation is

wanted, how the examples are set up and how the constraints are used. The fewer constraints are used, the more important the objective function becomes, as the resulting pose is defined less by the constraints.

In general the examined terms can be split into three groups:

A Mandatory term. This must be included.

B Important term. At least one term from this group should be included.

C Optional term. This group is not always useful.

Objective function	**Group**	**Uses**	k
Distance to target	A	Mandatory, can not be spared.	1.0
Sum of weights	B	Simplify weights. Improves quality by keeping the pose as close to the original as possible.	0.1
Difference from examples	B	Ensures resulting poses close to the examples, and thereby improves quality.	~ 0.2
1 - sum of weights	B	Pursues weights summing to 1. Improves quality as the pose is kept close to the examples.	~ 0.2
Number of weights	C	Ensures that 1 example is better than 2, if giving the same pose.	~ 0.1
Distance to previous pose	C	Ensures a smooth animation.	~ 0.1
Number of examples in use	C	Controls number of examples active.	~ 0.1

Table 7.2: Uses for different terms.

7.5 Searching the pose space

7.5.1 Introduction

In the previous section, the blending of poses and construction of an objective function was discussed. In this section it will be explained how to search through the pose space, finding a pose minimizing the objective function.

It involves six parts described in the next sections: How to test a pose, what minimum to search for, descriptions of different optimization methods, limits of weights, example load order and a conclusion to this topic.

7.5.2 Testing a pose

A pose is defined by a set of weights W, one for each example. These weights have a lower and upper boundary, where 0.0 and 1.0 is usually suitable. In certain cases weights which exceeds the examples, like -0.2 and 1.3, can be useful, but most of the time they result in unwanted deformations, like self intersections. More of this in section 7.5.5.

When testing a new pose with weights W, all examples are loaded with their respective weights as explained in the *blendExamples* algorithm **3**, except the **M** matrix is not applied to the vertices in the end.

When the examples have been loaded, all handles will have an corresponding transformation matrix, **M**, which does not need to be applied to all vertices in the handles, as we do not want to actually create the pose. As we just need the pose for testing, only the center point of each handle is multiplied with **M**. This way the center points of all handles of a pose is known, and can be used to evaluate the objective function.

7.5.3 Global or local minimum?

When the objective function has been specified a new question arises; should the algorithm search for a global or local minimum of this function?

In usual search and optimization problems, you wish to find the global minimum, and use a lot of effort not getting stuck in local minima.

But in this particular problem, of finding weights for the examples, a global minimum does not seem suitable, depending on the energy criteria.

One of the main goals of an animation system, is a smooth and continually animation, that is the current frame-pose must not be totally different from the previous pose, and so on. If the poses of two consecutive frames are too dissimilar, the animation will look jumpy and unrealistic.

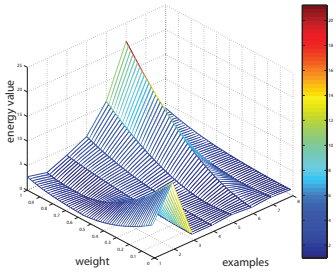
So what we actually want in our system, is a minimum, not far from the previous combination, to get a smooth transition. This will many times actually be the global minimum, but in some areas, it is not. This is where a global optimization scheme will fail, and create a jumpy animation.

When searching globally, the jumpiness can be avoided by using the energy term *Difference from last pose*, which punishes solutions which alters the pose too much. This should give some good results, but it is probably not the most efficient method. So a local search, without the *Difference from last pose* term, should be able to give approximately the same result, and be a lot more efficient.

7.5.3.1 Analysis using Matlab

It is impossible to visualize the pose space if there are more than two examples involved, as the dimension of the system becomes to large. Some plots can be made though, providing a general idea about the behavior of the energy function.

(a) All examples.



(b) Example 4 and 5

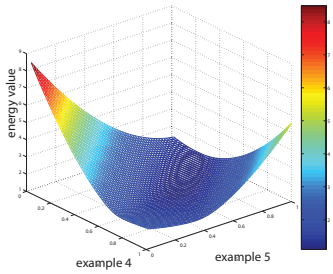


Figure 7.8: Using example set A. In **(a)** all examples are tested with different weights, and in **(b)** two examples are selected, and plotted together, to test their dependencies. A lot of these plots have been made, a selection of them can be seen in appendix B.

These plots gives the impression that the energy function is smooth curve with only one minimum when looking at a single example, and a smooth surface also with a single minimum when looking at two examples.

These plots reinforces the believe that a simple local search will be sufficient.

7.5.4 Optimization Methods

Two local optimization method, *Hill Climbing* [YM93] and *Hooke and Jeeves* [HJ61], and a global optimization scheme, Simulated Annealing [Trss01], have been looked at. Also a straight forward exhaustive search has been implemented for testing.

An important thing to note is that all functions here, assume that the derivatives of the energy function are not known, as it is a highly complex function. Numerical differentiation could have been used, but was regarded too costly and also outside the scope of this project.

7.5.4.1 Hill Climbing

Hill Climbing is an optimization function, where the latest optimum is used as a starting point, and then an improvement is sought by moving out from that, one step at the time.

This is a very simple optimization function, which comes in two variants; *Simple Hill Climbing* and *Steepest Ascent/Descent Hill Climbing*.

Simple Hill Climbing is selecting the first better solution it finds, and then iterates.

Steepest Descent Hill Climbing is comparing all possible directions, and chooses the best of these, and iterates. Usually the gradient is used as the value to be found, but here it is simplified to evaluate the energy function per step, and choose the one giving the best energy.

Both types of Hill Climbing has been implemented and tested.

Simple Hill Climbing: This is a description of the implemented method of the type Simple Hill climbing.

The algorithm goes through the examples one at the time, and tweak its weight. Starting at previous best, it tries to increase the weight, if no gain is found, it tries to lower the weight.

If a better weight is found, it keeps going in this direction until no gain or a boundary is found. The boundaries are narrowed considerable to force a local search. It is only allowed to move 10% of the available span in one step.

The Simple Hill Climbing implemented can be seen in algorithm **4**.

Algorithm 4 simpleHillClimb

Require: examples E, previous weight w_e
  $bestEnergy = findEnergy$
  for each e **in E do**
    $goLeft = true$ //decrease weight
    $goRight = true$ //increase weight
    while $goLeft\ AND\ w_e > limitLow$ **do**
      $w_e- = stepping$
      $energy = findEnergy$
      if $energy < bestEnergy$ **then**
        $bestEnergy = energy$
        $w_{best} = w_e$
        $goRight = false$
      else
        $goLeft = false$
      end if
    end while
    while $goRight\ AND\ w_e < limitHeigh$ **do**
      $w_e+ = stepping$
      $energy = findEnergy$
      if $energy < bestEnergy$ **then**
        $bestEnergy = energy$
        $w_{best} = w_e$
      else
        $goRight = false$
      end if
    end while
  end for

Steepest Descent Hill Climbing: This is a slower but more accurate version of the above. Before a weight of an example is tweaked, all weights are examined, and the one giving the best result is selected. When this weight has been tweaked, the method iterates.

Where the Simple Hill Climbing only optimizes each weight one time, this method iterates until no improvement can be found. This means we are sure to end in a minimum, possible a local. This also makes the cost of the optimization less predictable, which is not good. To avoid too big frame-times, the number of iterations can be limited. A limit equal to the number of examples was found suitable during testing. This also makes this method scalable, as this limit can be made time dependent, and thereby run for an equal amount of time on different problems and hardware. Fewer iterations is of course having an affect on the precision, but in practise this is not noticeable unless the number of iterations is very low.

For more details about this function, please look in the source code on the accompanying CD.

7.5.4.2 Hooke and Jeeves

Hooke and Jeeves Direct Search Method is an interesting method for this particular problem. It is a simple direct search method which given an objective function with n arguments, searches for a local minimum from a given initial guess. There is no requirements that the objective function must be differentiable or continues.

The method starts at the initial values, it tests all directions for an improvement in the objective function. If it finds an improvement, it moves its best guess to this position, and iterates. It constantly reduces the step size to increase the accuracy of the search of the minimum.

The method is very effective, but can be inaccurate because it gets stuck at local minima. This can normally be avoided by selecting an initial step size big enough to escape local minima. But as discussed in section 7.5.3, it not necessarily a bad feature.

These properties makes it very interesting for this particular search problem. As initial guess, the last best solution is used, and it will then adjust all weights to optimize the energy function best possible.

A problem with the Hooke and Jeeves algorithm is that it is not stable. Sometimes it runs loose, spending much more time than its average. This makes it jumpy and annoying for the user.

To correct this, a simplified version has also been implemented. The main difference is that it does not try all directions, but it selects the first better

solution found, like the Simple Hill Climbing method does.

The algorithm goes through the examples one at the time, and tweak its weight. Starting at previous best with a large stepping, it tries to increase the weight, if no gain is found, it reduces the stepping by a factor 2, and tries again.

If the high boundary for the weight is found without any gain, it then tries to lower the weight in the same manner.

If a better weight is found, it keeps going in this direction until no gain or a boundary is found.

7.5.4.3 Simulated Annealing

Simulated Annealing is an algorithm for the global optimization problem, where we want to find the global optimum of a function in a large search space.

Annealing is the method in metallurgy where one heats up the metal, and the let cool slowly down again, to make the metal stronger. Simulated Annealing works by the same principles and is therefor named after this old method.

It starts by choosing some random start solution and a temperature. A value for how good this solution is, is found. For each step of the algorithm we move to a new solution, which solution is dependent on how good the solution is and the temperature of the algorithm. In the beginning, with a high temperature, both better and worse solutions can be chosen, but when the temperature is lowered, as the algorithm progresses, it is less likely to choose a worse solution.

Being able to move to a worse solution, is what makes this algorithm special, it prevents it from getting stuck in local minima.

Unfortunately this also makes the method slow, as it needs to search a large part of the space, and does not use the fact that the previous best solution is known.

This method has not been implemented and tested due to its apparent high cost.

Algorithm 5 simple Hooke and Jeeves

Require: examples E, previous weight w_e
  $bestEnergy = findEnergy$
  for each e **in E do**
    $goLeft = true$ //decrease weight
    $goRight = true$ //increase weight
    while $goLeft\ AND\ w_e > limitLow$ **do**
      $w_e- = stepping$
      $energy = findEnergy$
      if $energy < bestEnergy$ **then**
        $bestEnergy = energy$
        $w_{best} = w_e$
      else
        if $stepping < \varepsilon$ **then**
          $w_e+ = stepping$ // GO BACK
          $stepping = stepping/2.0$ // REDUCE STEPPING
        else
          $goLeft = false$
        end if
      end if
    end while
    $resetStepping$
    while $goRight\ AND\ w_e < limitHeigh$ **do**
      $w_e+ = stepping$
      $energy = findEnergy$
      if $energy < bestEnergy$ **then**
        $bestEnergy = energy$
        $w_{best} = w_e$
      else
        if $stepping < \varepsilon$ **then**
          $w_e- = stepping$ // GO BACK
          $stepping = stepping/2.0$ // REDUCE STEPPING
        else
          $goRight = false$
        end if
      end if
    end while
  end for

7.5.4.4 Exhaustive search

All other implemented methods have the feature that they find a local minima. As discussed in section 7.5.3, it appeared that a local minima would actually suit the proposed method best. To test this a global exhaustive search method was implemented. This search method is not meant to be fast, but simply exists to compare the results from the local search methods to a global result.

Algorithm 6 Exhaustive search

Require: examples E
  $bestEnergy = INF$
  for each e **in E do**
    $w_e = w_{low}$
    while $w_e < w_{high}$ **do**
      $energy = findEnergy()$
      if $energy < bestEnergy$ **then**
        $bestEnergy = energy$
        $w_{best} = w_e$
      end if
      $w_e += stepping$
    end while
  end for

7.5.5 Limits for weights

The system contains some limits for weights of the examples. Several settings has been tested, but keeping the weights between 0.0 and 1.0 produces the best results.

Letting the weights escape these traditional values, extrapolating the examples, sounds very interesting, but in practice it leads to self intersections and limbs bent in wrong directions, which practically ruins the animation.

When using the simple Hill Climbing method, the weight limits are adjusted each step, to force a local search. These limits are set at +/- 0.15 from the previous best weights. This enables some very smooth animations, but sometimes requires several iterations to converge on a final solution.

7.5.6 Example order

In section 7.3 it is stated that there is two types of parameters to optimize; weights and the order of the examples.

So far the order has not been mentioned much, so in this section this parameter will be discussed.

As shown in section 7.2.2 the order in which the examples are loaded can have a big influence on the result. Because of this, it is obvious to think that the optimization function should try to reorder the examples to give the best pose. But in fact a reordering can change the result so much, from one frame to the next, such that the resulting animation becomes bad and jumpy.

Another issue with the load order, is apparent when using an optimization method which tweaks the examples one by one, in the initial order[3]; the weights assigned to the examples depends on the order of tweaking. A 'wrong' pose could be given higher weight than a 'better' pose, just because it was tweaked first.

Using Hooke and Jeeves or the Steepest descent Hill Climbing though, reduces this problem greatly, as they already test for different orders by the way the algorithms are build. But to test the effect on the simple schemes, two reordering functions have been implemented: One which reorders based on the examples distance from the constraints, and one which reorders by the weights

[3]Simple methods: Simple Hill Climbing and simple Hooke and Jeeves.

the examples were given in the last frame.

Reordering by example distance Each example's distance to the target is measured in the same manner as described in section 7.4.1. The order in which the examples are loaded are then changed, so that the example closest to the target is loaded first, and so on.

Reordering by weights The order in which the examples are loaded are changed by looking at which weight the example was given last frame. Examples with larger weights are loaded first, and so on.

The reordering by weights, is not a good approach, as it changes the order too much and to the worse, and this makes the animation look bad.

Reordering by distance on the other hand, can actually be quite helpful, especially regarding issue number 2 above, it really saves the simple optimization methods from following a wrong tweaking order. This is visible in the walk example, where the motion using one leg is the first 4 examples, and the other leg is the next 4 (see set A on figure 9.10). Using a simple method and no reordering, will lead to the character dragging the second leg a bit while the first looks better. Introducing reordering drastically reduces this effect.

7.5.7 Conclusion

When finding single poses using the Pose Interpolator, all implemented optimization algorithms produces approximately the same result. So it is possible to select the least costly algorithm, namely the Simple Hill Climbing.

If creating a smooth moving animation, the Simple Hill Climbing might be to likely to get stuck in local minima. S.D. Hill Climbing or the simple Hooke and Jeeves performs better, but at a little more cost per frame.

It should be mentioned that Genetic Algorithms were also considered and examined, but as Simulated Annealing they do a global search and they are not particular fast, so they were left out from this project, but in future works it might be interesting to see if ideas could be borrowed from these types of optimization schemes.

7.6 Animation

In this section it is explained how the previous parts; blending, objective functions and optimization functions, are used for actual animation.

7.6.1 Interactive Pose Interpolator

This is a system which allows the user to place constraints on some handles, and manipulate these. The system then uses the method described in this chapter to create the best suited pose: It uses the optimization function to optimize the objective function. The resulting pose is then loaded as described in section 7.2 and displayed.

The user can change the optimization function, and can select which terms the objective function should use, to tune the resulting pose.

The workflow can be described as:

1. The user adds constraints to some handles.
2. The user drags a constraint to a new position.
3. The system searches the pose space using the selected optimization method and objective function.
4. When the optimal weights are found for each example, these are loaded using the *blendExamples* function.
5. Finally the free vertices are deformed using Laplacian Reconstruction.

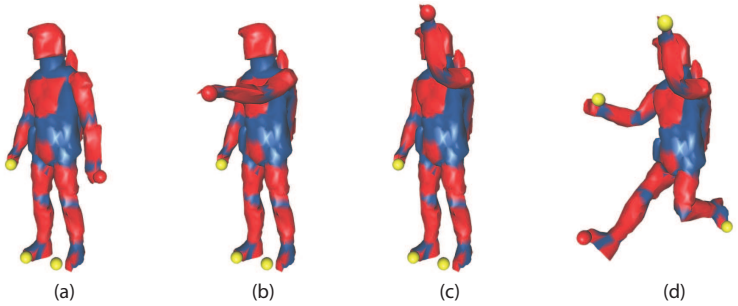


Figure 7.9: Interactive Pose Interpolator: Manipulating 4 constraints.

7.6.2 Path Animation System

This is a more complete animation system, enabling the model to actually perform a moving dynamic animation.

It is based on the Interactive Pose Interpolator from above, but in addition it contains a system for creating paths for the constraints.

The user must, for each constraint, define a path consisting of a number of points in space. When the animation is started the system will move the constraints along the paths, and after each move, find the best suited pose to fit this new constraint set.

7.6.2.1 Global positioning

When the model should follow a path, a problem not yet discussed becomes apparent: Global movement.

To create a fully functional global animation scheme is not in the scope of this project, but a simple global movement method has been developed, as a proposal to how it can be done when using this animation technique.

A technique where the global position was optimized as a part of the energy function was considered, but this resulted in too unstable animations. Instead it is up to the user to define a handle from which the global position should be defined. This works very well, but for a walk, the part of the model defining its global position, shifts between the two feet. To cope with this, it is possible

to select several handles to define the position. Which of these are used in a given frame is determined by its change in target; the stationary foot (contact with the ground) will have little or no change in its target, so this is used. This approach works out very well.

The workflow can be described as:

1. The user adds constraints to some handles.
2. The user selects the handle(s) which should define the global position.
3. The user creates a path for each of the constrained handles.
4. The user activates the animation.
5. The system finds the points on the paths corresponding to the given timestep.
6. The global movement is found using the technique described above.
7. The system searches the pose space using the selected optimization function and objective function, taking the global movement into account.
8. When the optimal weights are found for each example, these are loaded using the *blendExamples* function.
9. The free vertices are deformed using Laplacian Reconstruction.
10. Finally the model is translated according to the global movement found.

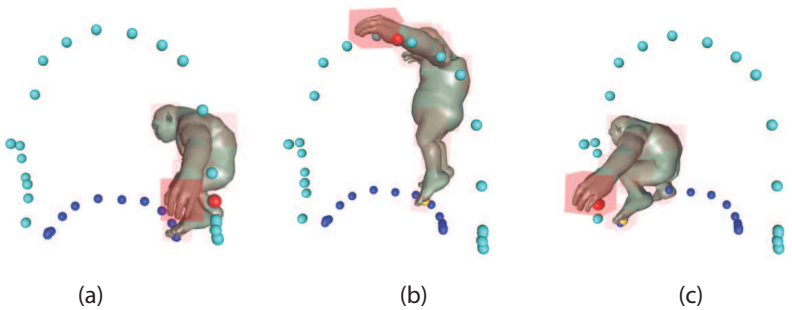


Figure 7.10: Using 2 constraints with paths (hand and foot) to create a jump. The foot is selected as global position reference.

7.6.3 Keyframe animation

The system also contains a standard keyframe animation system. It uses the examples created by the user, in an order specified by the user, as keyframes. It

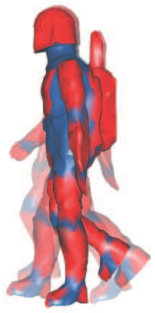


Figure 7.11: Walk cycle (example set A) as keyframe animation (Also see accompanying video).

does not use constraints, the optimization function or the objective function. It simply makes a linear interpolation of the weights of two examples at the time.

This can be used for testing examples for a specific animation, as it can be run while the examples are created.

	examples			
Step	1	2	3	4
1	1.0	0.0	0.0	0.0
2	0.5	0.5	0.0	0.0
3	0.0	1.0	0.0	0.0
4	0.0	0.5	0.5	0.0
5	0.0	0.0	1.0	0.0
6	0.0	0.0	0.5	0.5
7	0.0	0.0	0.0	1.0

Table 7.3: Example of keyframe weights. Stepping = 0.5

Part IV

Results

Chapter 8

Implementation

8.1 System overview

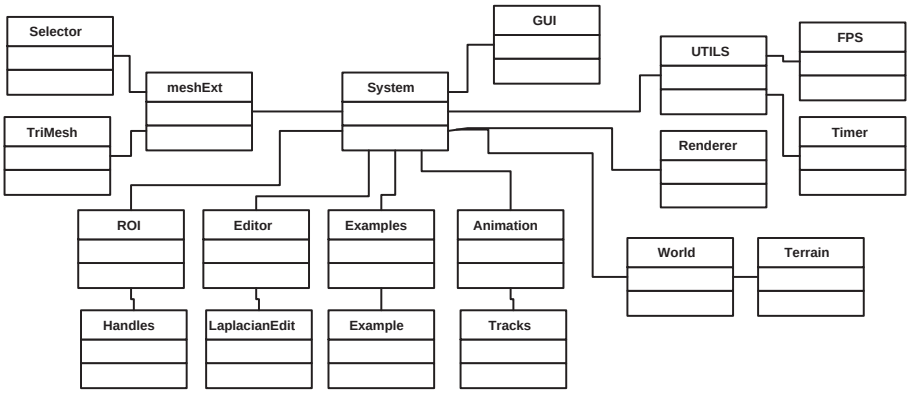


Figure 8.1: Class diagram.

System Main class, also contains user input functions.

GUI Graphical User Interface.

Renderer Render class, responsible for all openGL calls.

UTILS Container class for different utilities.

FPS For measuring frame times.

Timer A timer.

MeshExt Extended TriMesh class.

TriMesh TriMesh class from GEL [GEL].

Selector* Controls selection specific functions.

ROI* Region of Interest class. Keeps track on which vertices goes where.

Handle* A handle class, contains all necessary information and functions regarding a handle.

Editor Class to control the Laplacian Editing system.

LaplacianEdit* Laplacian Editing class.

Examples Class which contains any number of examples, and functions to manipulate these.

Example Example class.

Animation* All animation related functions.

Tracks The class to create and maintain the paths the constraints can follow.

World Class to control the world (terrain, skybox etc).

Terrain Terrain class.

* The source code of these classes can be found as printouts after this report. The rest can be found on the accompanying CD.

8.2 Packages used

GEL Is used for loading and saving model, and storing the mesh in an indexed face set.

TAUCS Used for Laplacian Editing, to solve linear systems.

OpenGL As 3D renderer.

GLUT As utility package for OpenGL.

GLUI As Graphical user interface.

8.3 Basic implementation details

The program has been developed in Visual Studio 2003, on a Pentium 4 3.4GHz with 512mb Ram and a GeForce 6600GT running Windows XP. All timings in this thesis is also from this setup. The program is approximately 9.000 lines of C++ code.

8.3.1 Loading the model

Loading the model into the system is done using GEL. But a auxiliary data structure is wrapped around GEL's indexed face set, containing useful data and functions as neighborhood connectivity, normal calculation and calculation of the Laplacian.

8.3.2 Geometry representation

As the program work with one model, which does not change in regards of connectivity, Vertex Buffer Objects (VBOs) has been chosen as store vertex data during visualization.

8.3.3 Selection

Selection was a huge concern in the design of the program. It should enable the user to easily choose his handles, and manipulate them, and at the same time it should be fast.

The main selection tool, was chosen to be a box selection. The user presses the mouse button, and drags it to create a box. When the button is released, all vertices within this box, will be part of a new handle. In practice all vertices are projected onto the screen, and tested against the box borders. This is extremely fast, and is many times more effective than OpenGL's render to selection. One concern though is that all vertices in this box is selected, and sometimes the user might just want to select the visible vertices. A solution is to do a normal check, to find out if the normal of a vertex is pointing towards the screen or not. But this will also select any surface behind the visible surface as shown on figure 8.2.

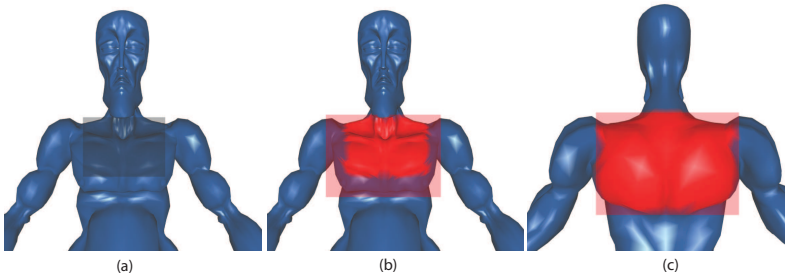


Figure 8.2: Selecting all vertices within the box.

To solve this, a single face is selected at the center of the box, and then a flooding algorithm is finding all vertices connected to this vertex and still inside the selection box and having its normal pointing towards the screen.

A version of this method was also implemented, where the normal check is skipped. This makes a very useful tool for selecting specific parts, with other visible parts behind it (figure 8.3).

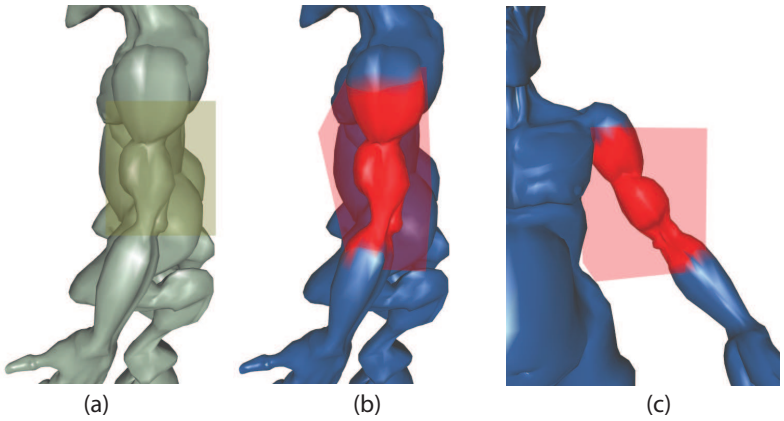


Figure 8.3: Selecting vertices only on the arm, though the box covers the body as well.

When the main selection is done, it is possible for the user to add to or subtract from a handle using either the selection box, or a more simple paint function working on single faces.

8.3.4 Handle visualization

How the handles should be visualized, was a question not easily answered. The first try, was to make an axis aligned bounding box around the handle vertices, and show this as handle. This was chosen due to the low cost and easy implementation, but it has been proven inadequate, making it too complicated for the user as no clear distinction between handle vertices and free vertices are shown.

This bounding box idea was then extended by coloring the faces on the mesh a specific color depending on if they are part of a handle or not. This combination works great; The coloring of vertices visualize the handle itself, and the bounding box is helping making selecting handles more easy.

Alternatively another primitive like a cylinder or sphere could be used in stead of the box. It should not be axis aligned though, and the handle primitive should then be subjected to the same transformations as the handle vertices. Then it would fit the model better, even when deformed. But to keep it simple, the axis aligned bounding box was used.

8.3.5 System limitations

Many of the models that can be found on the internet and the like, is not manifold, i.e. they actually consist of several joined models. If this is the case, and there is no connection between these different parts, there is no guarantee that the system will work. But as the system is meant to use laser scanned models it is not considered a huge issue.

Because of the method for rotating the differential coordinates, by using smoothing, it is critical that the extremities of the model is part of a handle. A nice case is portrayed in figure 8.4, but for some models the surface is smoothed too much, the solver fails and the surface is lost.

Another issue encountered in models, is floating vertices. The Stanford Bunny model seems to have vertices not belonging to a face, and the system fails when these are encountered, it need at least 1 neighbor.

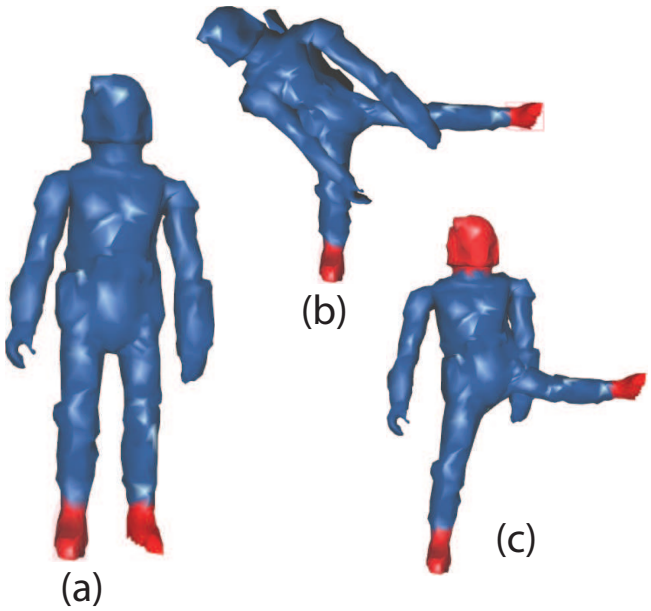


Figure 8.4: (a) Original. (b) Only two handles. (c) Three handles.

8.3.6 System flow

In this implementation, the user goes through following steps:

1. First step is to define the handles.
2. Define their rotation centers.
3. Edit the handles and save them as examples.
4. The final step is to animate, either using the pose interpolator or the path system.

8.3.7 User Guide

A short user guide can be found in appendix G.

Chapter 9

Results

In this chapter the results of the implemented method is shown.

The chapter is split in 4 parts:

Creating poses Results of the pose creation system.

Interactive pose interpolator Pictures of poses created using different objective functions and optimization methods.

Animation system Pictures from some animations.

Performance Timings and dissection of both the deformation procedure and the animation system.

The results consists mostly of models in different poses. Two models are mainly used: The satyr to demonstrate creation of poses, and the simplest Boba Fett model to demonstrate the pose interpolator and the animation system.

The models are either shown in the standard greenish color, or in the red and blue colors defining the handle structure. Which is shown for the specific picture, is based on what was thought to provide the best illustration.

9.1 Creating poses

9.1.1 The handle structure

Illustrations on how the handles can be created and edited.

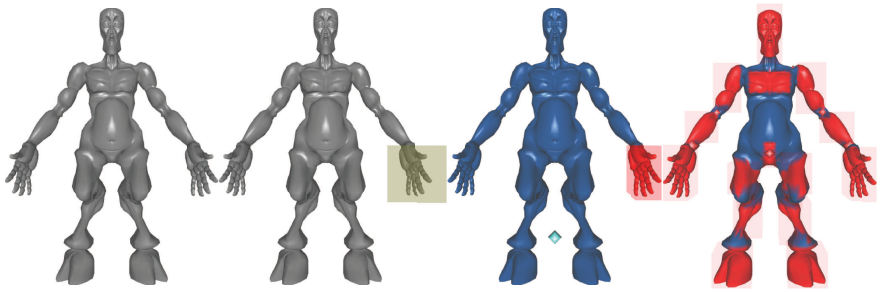


Figure 9.1: From left to right: **1.** Initial model **2.** Selection box **3.** After selection **4.** Final structure.

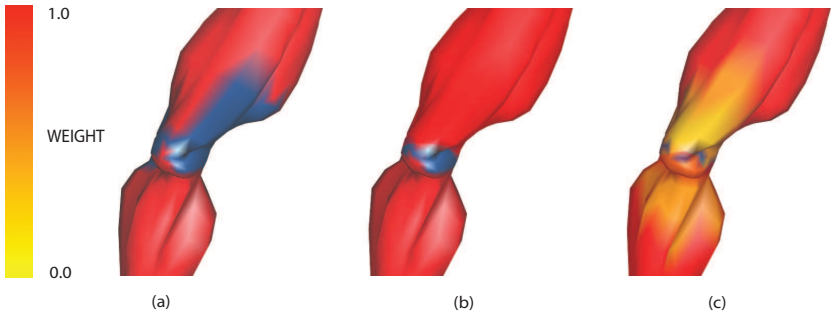


Figure 9.2: From left to right: **1.** After box selection **2.** Fine painting the handle **3.** Changing weights of handle vertices (yellow is less than 1.0).

9.1.2 Deformations

Using the handle structure from above, the following poses are created:

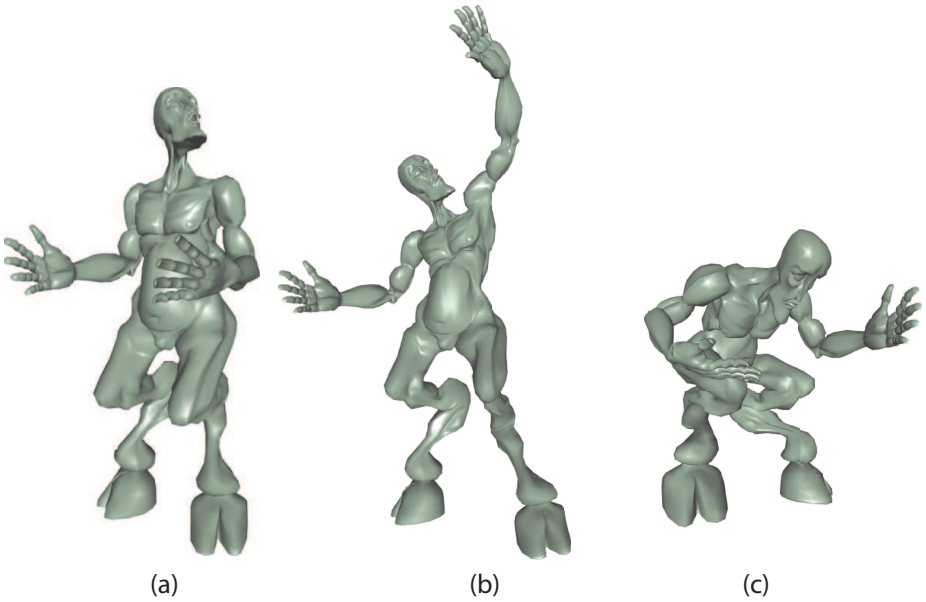


Figure 9.3: From left to right: **(a)** Example pose Looking up. **(b)** Example pose Stretching up. **(c)** Example pose Crouching.

The handle structure is extended to be more detailed on the hands. This leads to the possibility of deforming the hand and fingers:

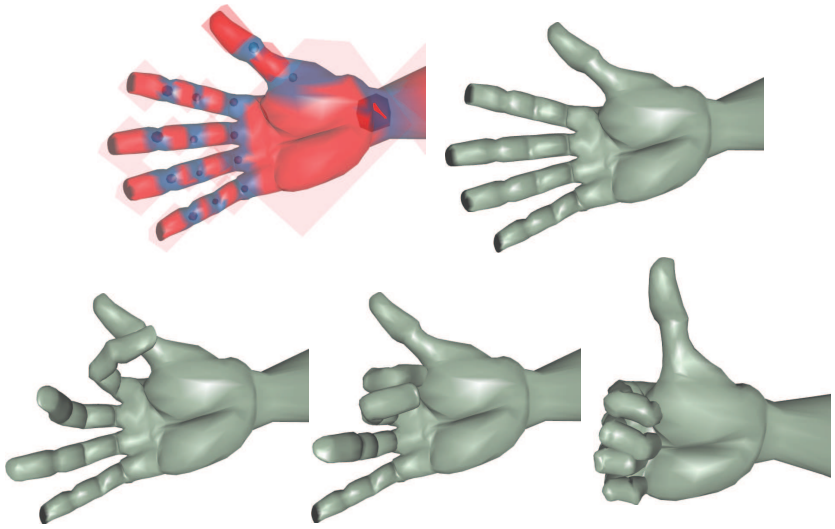


Figure 9.4: Top row: Initial hand. Bottom row: Bending the fingers to form a fist, ending in a thumbs-up gesture.

The face is also made more detailed. Using many small handles, facial expression can be created:

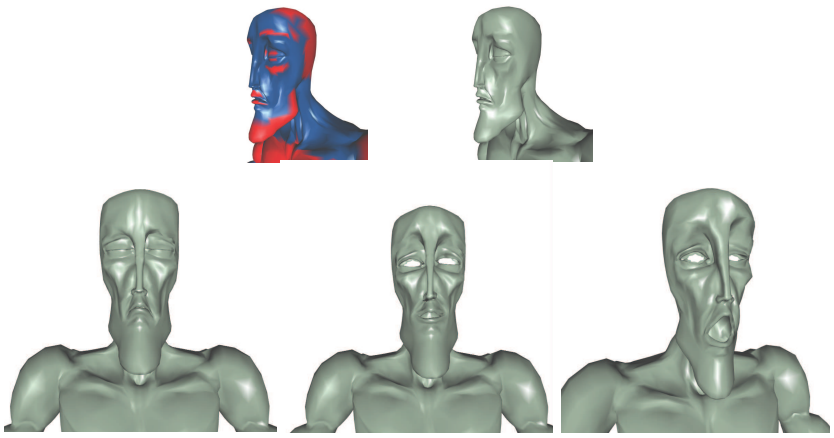


Figure 9.5: Top row: Initial face. Bottom row: Different facial expressions.

Closeups on detail areas and transition areas after deformation are shown here:

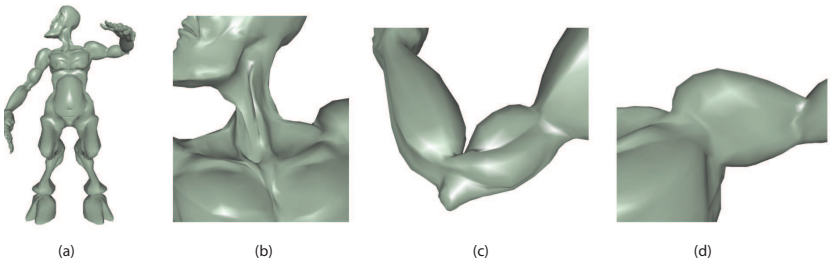


Figure 9.6: **(a)** Raised arm, bending the elbow, turned head. **(b)-(d)** Close-ups to illustrate the quality of the skinning method. Notice all the details are preserved.

To demonstrate the methods ability to deform the mesh naturally, a closeup of a high-resolution mesh is shown here:

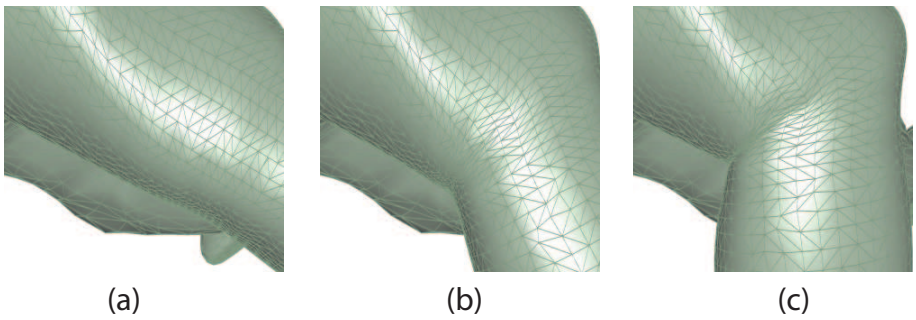


Figure 9.7: Closeup on the mesh deformation occuring.

Although the Deformation Detail Layer functionality is not a direct part of this method, a result of the usage is illustrated:

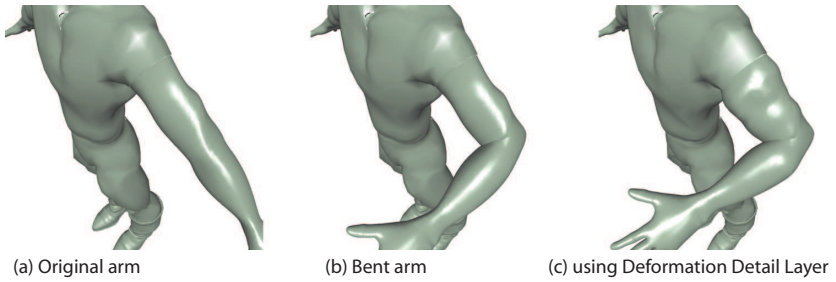


Figure 9.8: Illustrating the possibilities of using the Deformation Detail Layer. The model to the right (c), is deformed in a more vivid way, as the muscles bulges.

A 'wrong' bending of an arm demonstrates the ability to change the surface's appearance during deformations using handle weights.

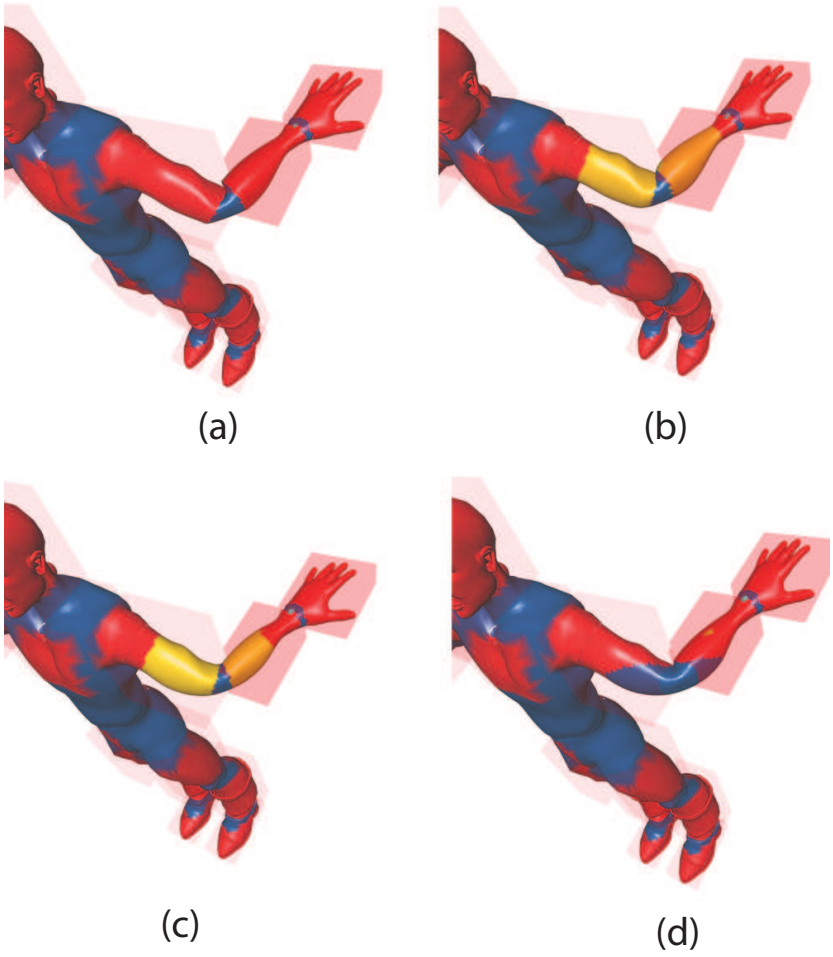


Figure 9.9: (a) Fully rigid handles. (b) 2/3 rigid handles. (c) 1/3 rigid handles. (d) non-rigid handles.

9.2 Interactive pose interpolator

This system is where the user manipulates some constraints and then the system adapts the pose to fit these.

9.2.1 Example sets

Using the pose creator two example set are created. One to be used in an animation sequence, defining good poses (Set A), and one better suited for the pose interpolator, by defining extreme limb positions (Set B).

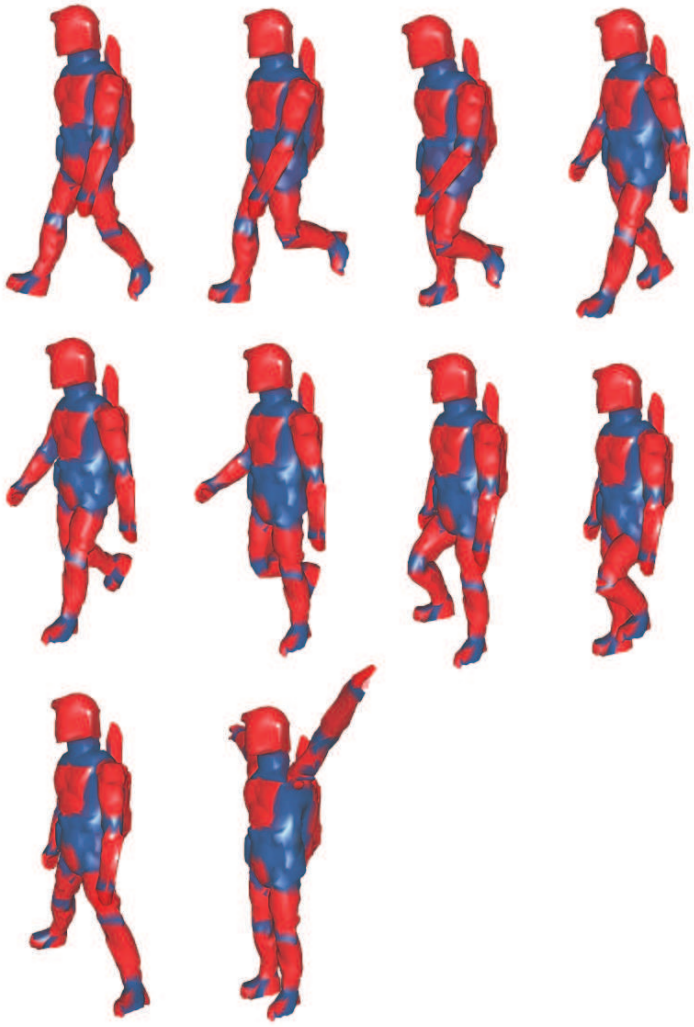


Figure 9.10: Example set A. 6 first is the walk cycle. 2 next is to walk up or down. Last two examples, spreading the legs and arms, are included to add some flexibility in these dimensions.

Figure 9.11: 16 examples of set B. Defining the extreme positions of the limbs.

9.2.2 Constraints

Here are shown examples of four different constraint settings, for both set A and set B.

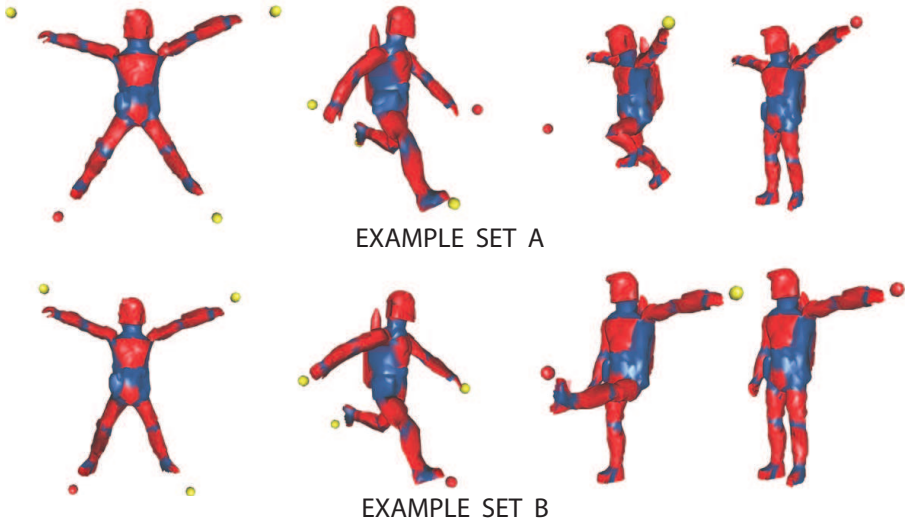


Figure 9.12: 4 different sets of constraint targets. Using set B results in the best interpolation of the constraints, because set A is solely constructed as a walk cycle.

9.2.3 Objective functions

Here results concerning the objective function, and the different terms which was tested, is presented.

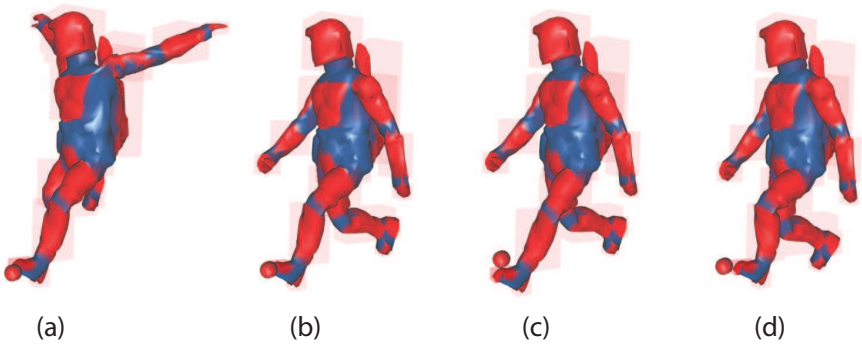


Figure 9.13: Using 1 constraint and example set A. **(a)** Objective function consist of distance to target (DtT) only. **(b)** DtT + minimize sum of weights. **(c)** DtT plus Minimize One minus sum of weights. **(d)** DtT + Difference from examples. From these it can be seen that any expansion of the DtT energy is an improvement. Which of these is the best, is hard to tell from a single pose.

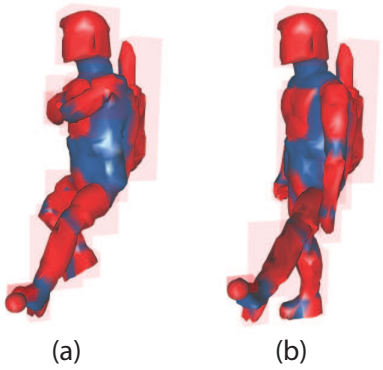


Figure 9.14: Using 1 constraint and example set B. **(a)** Objective function consisting of distance to target (DtT) only. **(b)** DtT + any other. With only DtT energy, the result is a very strange mixture. But with the addition of any other term, it is corrected.

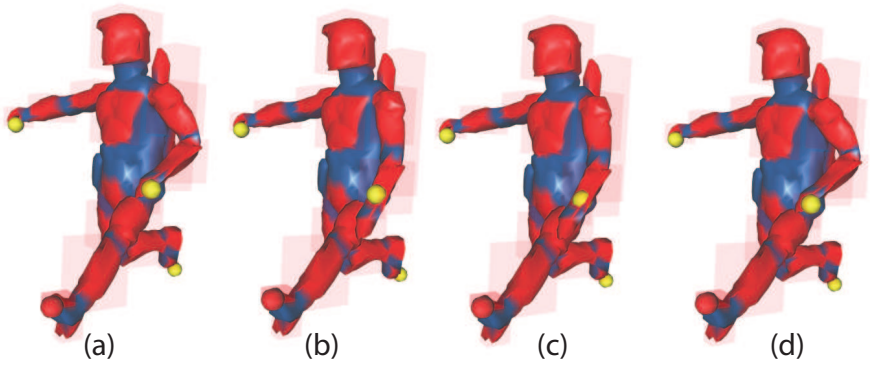


Figure 9.15: 4 constraints, example set B. **(a)** Objective function consist of distance to target (DtT) only. **(b)** DtT + minimize sum of weights. **(c)** DtT plus Minimize One minus sum of weights. **(d)** DtT + Difference from examples. All is actually good, but (b) and (c) is less accurate than the simple DtT (Can be seen looking at the left arm).

Next is demonstrated what happens if the number of examples wanted active is limited:

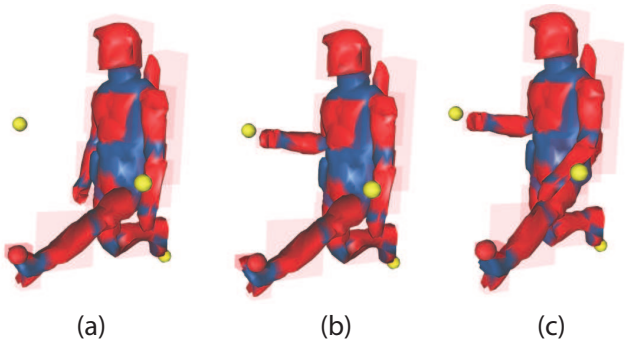


Figure 9.16: 4 constraint, example set B. **(a)** Objective function contains Number of examples = 2. **(b)** Objective function contains Number of examples = 2, with less weight. **(c)** Objective function contains Number of examples = 4

It takes 8 examples to match the pose on figure 9.15 completely.

In table 9.1, the observations made during testing is given for each objective function.

Objective function	**Pros**	**Cons**
Distance to target	Always needed, making sure the pose follows the constraints.	Does not control the pose style.
Sum of weights	Keeps the weights as simple as possible.	Prefers the original pose defined by weights being all zeroes.
Difference from examples	Favors poses which resembles the examples.	Can create 'energy holes' around the examples, which can be hard to get out of.
1 - sum of weights	Controlling the weights, making them sum to 1, leads to simple weights without favoring the original pose.	Sometimes weights not summing to 1 is wanted.
Number of weights	Favors poses constructed by using fewer weights (keeping it simple).	Should not have too large a weight, as it will easily limit the pose to consist of only 1 example.
Distance to previous	Makes the animation more smooth.	Produces an 'energy hole' around last pose, which can be hard to get out of.
Number of examples in use	Keeps the poses simple, by removing less important examples, so it uses a user-defined number of examples.	Details can be lost, if too few examples are used.

Table 9.1: Terms in the objective function. Pros and Cons.

9.2.4 Optimization functions

In table 9.2 the optimization functions are commented.

Method	Pros	Cons
S.D. Hill Climbing	Iterative method, which thereby is scalable.	Unpredictable running times unless controlled.
Simple Hill Climbing	Very fast and produce a very smooth animation.	Can get too attached to a local minima.
Hooke & Jeeves	Wide search, can escape local minima.	Very unpredictable running times, and slow in general.
Simple Hooke & Jeeves	Fast and produces a smooth animation.	-
Global exhaustive	Does not get stuck in local minima.	Slow and can be jumpy.

Table 9.2: Optimization functions. Pros and Cons.

9.3 Animation system

Here paths are setup for the constraints, which the system then adapts the pose to, creating a continues motion. Videos for the results presented here, can be found on the accompanying CD.

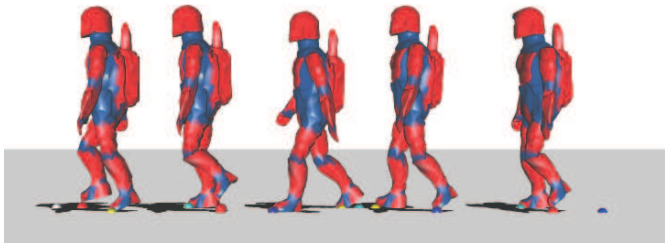


Figure 9.17: Walk path created over flat terrain. Example set A is used. Only two handles needs to b constrained, namely the feet. (Running at around 20fps)

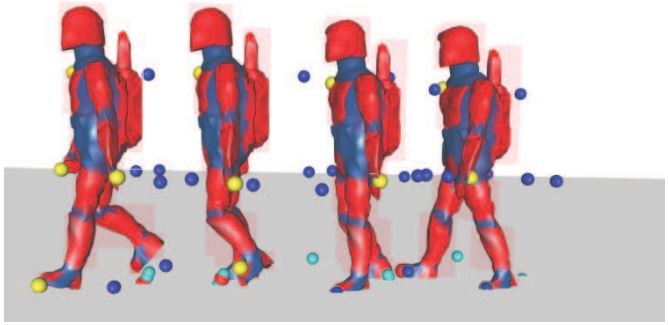


Figure 9.18: Walk path created over flat terrain. Example set B is used. Note that 5 handles are constrained; the feet, the hands and the chest to define the global movement. (Running at just under 20fps)

Results of the animation performed using a walk path with different objective functions, and different optimization functions can be found in appendix F.

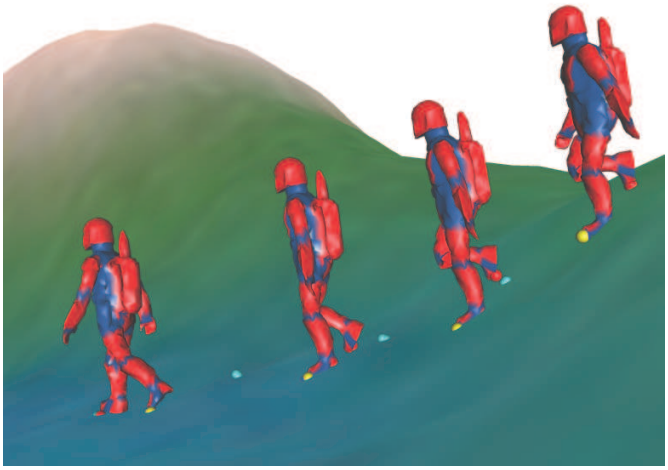


Figure 9.19: Walk path created over terrain with hills. Example set A is used.

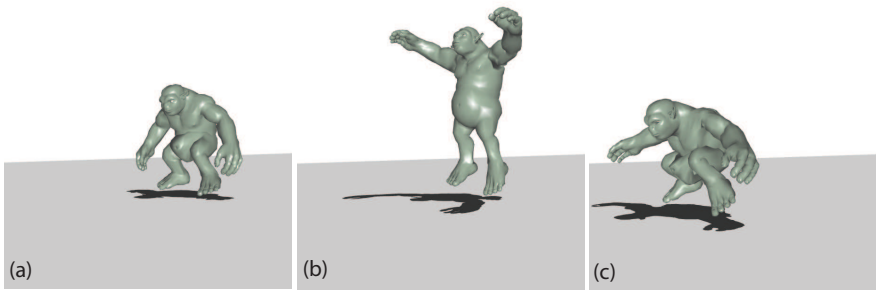


Figure 9.20: A jumping orc, created from 2 examples. More pictures can be found in appendix F.2. It is running at around 15 fps.

9.4 Performance

9.4.1 Deformation

The performance of the deformation system used to create poses, is almost solely dependent on the speed of the Laplacian Reconstruction. Rotating the handles and the rigid vertices in these handles, only takes a fraction of the time.

Below the Laplacian Reconstruction system is examined and timed.

Main test case:

- 4250 faces / 2138 vertices (Boba Fett 2k vertices)
- 18 handles
- 1635 vertices in ROI (free vertices + edge of handles)

This gives a Laplacian matrix, $\mathbf{L}$, with the size of 1635x2511 ($M \times N$).

9.4.1.1 Pre-computation

As described in section 4.1, the Laplacian system is solved using factorization and back substitution. This factorization can be quite time consuming, so this step has been dissected and timed below.

Step	Time
Create $\mathbf{L}$ (Laplacian matrix)	0.0097s
Transpose $\mathbf{L}$	0.001s
Multiply $\mathbf{L}$ and $\mathbf{L}^T$	0.021s
Factorize matrix	0.025s
Prepare Laplacians	0.033s
Prepare in all	0.090s

Table 9.3: Timings for the steps in creating the Laplacian system and Cholesky factorization (in seconds). Prepare Laplacians means to calculate the local frames and represent the Laplacians in these (used when rotated, as described in section 4.1.7.1).

For this particular example, the preparation time is so low, that the user will not notice any delay. And as the preparation of the system is done only once, and in the transition from one step to the next, a delay of up to a couple of seconds would actually be acceptable. As can be seen in table 9.4, this means that when looking at preparation time alone, a model with 40.000 ROI vertices (model size between 45.000 and 90.000 vertices depending on setup of the handle structure), is quite doable, as it takes less than 4 seconds.

Model (vertices)	M	N	Time
Boba Fett (2138)	1635	2511	0.09
Boba Fett (10600)	8800	9648	0.52s
Armadillo (17300)	13160	14019	1.041s
Boba Fett (45000)	37874	38985	3.45s

Table 9.4: Timings for the preparation for different sizes of models (in seconds). M and N are the size of the matrix $\mathbf{L}$.

9.4.1.2 Per frame performance

So pre-computational time is not an issue even with large models, but the per frame cost is the real factor defining how big models can be worked with. In table 9.5 timings for different models are given.

The timings are dissected into rotating the Laplacian coordinates, solving the system (three times, one time for each axis), and the total frame-time.

Model (vertices)	Rotate $\boldsymbol{\delta}$ [s]	Solve x 3 [s]	Frame-time [s]
Boba Fett (2138)	0.021	0.0075	0.040
Boba Fett (10600)	0.120	0.050	0.183
Armadillo (17300)	0.200	0.084	0.30
Boba Fett (45000)	0.55	0.26	0.826

Table 9.5: Timings for one deformation action (in seconds).

As seen in the above table, the biggest cost is to rotate the Laplacian coordinates, so this will be dissected further.

The procedure involves:

1. Solving the linear system for the 3 axis, with 0's on the RHS

2. Calculating the smooth normals
3. Creating the local frames
4. Representing the laplacian vectors in the local frames

From table 9.5 it is known that step number 1 takes less than half of the time (0.0075s out of 0.021s). Of the remaining three steps, calculating the smooth normals are by far the biggest cost, using nearly the rest of the time spent.

9.4.2 Animation

Test case:

- 4250 faces / 2138 vertices
- 18 handles
- 10 examples

Timings on the optimization functions and the different energy terms are listed below.

Method	# objective evaluations	Average time
Simple Hill Climbing	20-30	0.005s
S.D. Hill Climbing[1]	60-800	0.018s
Simple Hooke & Jeeves	40-70	0.013s
Hooke & Jeeves	150-20000	0.4s
Global exhaustive	560	0.11s

Table 9.6: Timings on different optimization functions. Middle column is how many times the objective function is evaluated. Last column is a mean timing in seconds for the optimization function. Objective function used: 'Distance to target' + 'Sum to 1'

The cost of the individual call of the objective function is not as critical, as the overall cost of the optimization functions. Changing optimization function has a big effect on the performance, as the number of evaluations of the objective function varies a lot.

[1] Can be limited to a specific number of iterations, which eliminates the randomness.

Objective function	Mean time
Distance to target	2.0e-06s
Sum of weights	6.0e-06s
Difference from examples	2.9e-05s
1 - sum of weights	2.3e-06
Number of weights	1.9e-06s
Distance to previous	5.4e-06s
Number of examples used	1.7e-06s

Table 9.7: Timings on one evaluation of the different objective terms in seconds.

To test the effect of the number of handles and number of examples used in an animation, some tests were performed and the results shown in table 9.8.

	Nr examples		
Nr handles	8	12	20
18	17.4 fps	15.4 fps	12.4 fps
24	17.2 fps	14.9 fps	11.7 fps

Table 9.8: Frames per second, dependencies with number of examples and number of handles. These timings are using the small Boba Fett model, simple Hooke and Jeeves and an objective function consisting of DtT + Sum to One + Distance to Pose

Part V

Discussion

Chapter 10

Discussion

In this chapter the results are discussed, and a more general discussion regarding the proposed method is presented.

10.1 Advantages

Other methods rely on data from motion capture, several scans of a model in different poses, or a skilled animator using a lot of time.

This proposed method can work from a very simple foundation. Just a single model obtained by a 3D scan or from a modeling program, and then a relatively complex animation can fairly quickly be created.

The surface-based handle structure acting as skeleton, is simplifying the process for the user. Painting on the surface, to define the rigid areas, is simpler than setting up a skeleton structure. This also avoids the manual skinning process.

The animation is not based on direct interpolation between successive keyframes but is using these as examples of 'good' poses, which can be blended arbitrarily, in numbers and weights, to obtain a totally new pose. This makes it an adaptive method, which can be used in interactive animation. When an animation needs

to be able to adapt to a certain situation, all that is needed, is some additional examples covering this movement. Using the best suited examples the system can adapt to changing environment at run-time.

The proposed method also overcomes the problem of having to use two different animation methods for for example animating limbs and faces. Using the handle structure both large scale deformations like a walk animation can be created, but it can just as easily be used to change a characters facial expression.

10.2 Quality of method

When talking about quality of the method, two different topics should be looked at: The surface quality under deformation, and the methods ability to dynamically construct poses when animating.

10.2.1 Deformation

The quality of the deformations, is a direct result of the quality of the Laplacian Editing used and the setup of the handles.

The Laplacian Editing used is a straightforward and fast method, but for large rotations it may not offer the best results, as it does not include good volume preservation. Other types of Laplacian Editing might do a better job, but that would more than likely be a tradeoff with performance.

With the simple approach chosen, fairly good quality deformations can be achieved, but still artifacts like 'candy-wrapping' can be visible at larger deformations. Especially when thinking on the time used to setup the handle structure, the resulting deformation quality is impressive.

As an addition, using the Detail Deformation Layer is enabling the user to get just the kind of surface at a given pose as he wants. This is a powerful feature, which can result in some very realistic deformations containing muscle bulges wrinkles and so on.

10.2.2 Animation

Under the assumption that the examples are well created and suited for the animation wanted, the quality of the animations is directly related to the optimization and objective function, and also how the constraints are set up.

The optimization function is mainly responsible for creating a smooth and non-jumpy animation, and the objective function is responsible for keeping the resulting poses close to the examples, but still be flexible.

In this discussion, two different types of example sets are looked at: Sets defining the extends of limbs, and more targeted sets defining a sequence.

10.2.2.1 Examples defining limits

Examples defining the extends of the limbs, is best suited to be used in the pose interpolator. They can also, to some extend, be used in the example animation system, but a keyframe animation is impossible to create from these, as the poses are extremes and not the actual wanted poses.

If used in the animation system, many constraints are usually needed, one for each moveable part, to create lifelike animations. This complicates the user interaction a great deal, so in these cases, input from motion capture would certainly be preferred.

Which objective function is used is not critical, as long as the Distance to Target term is used along with one of the following terms:

- Distance to pose
- Minimize sum of weights
- Minimize one minus sum of weights

Which optimization function is used, is of less importance here, so one of the faster is preferred, simple Hill Climbing or simple Hooke and Jeeves.

Using this types of examples is actually making the pose interpolator act pretty much as a traditional inverse kinematics system: It gives a lot of freedom, but because of this it also requires a lot of control to achieve a wanted result.

10.2.2.2 Examples defining good poses

If the examples is defining good poses, like all being part of the walk cycle, the animation system becomes much more interesting. With some tweaking of the weights, all three energy terms above can actually create some good results, but the setting which continuously performs better is:

$$Energy = DtT + 0.2 * OMSoW + 0.3 * DfE$$

(DtT plus the Distance to pose term with a weight around 0.3 and the Sum to One term with a weight of 0.2).

It really limits the resulting poses to be within an acceptable pose space defined by the examples. Optimizing this function using the simple Hooke and Jeeves method, and also having the system reorder by distance, is apparently giving the best results.

Using the regular Hooke and Jeeves or the Exhaustive search is making the animation far to unstable, both in terms of speed but also smoothness. So one of the three fastest should be used, and as mentioned above, simple Hooke and Jeeves seems like the best choice.

Using the proposed method, it is actually possible to create rather good animations, using just a few examples, and some good target paths (see the jumping orc appendix F.2).

When basing animation on examples like this, the result is sometimes not exactly what the user imagined, but then it can usually be corrected by using a more thorough target path, which more clearly defines the wanted animation.

10.2.2.3 Summing up

Although a very general inverse kinematics system can be made using examples defining the extreme poses of the limbs, the real strength of the method is to have the system 'learn' good poses from the examples, by making them all part of a specific type of animation (like set A, figure 9.10). Using this method, enables a user, to fairly quickly setup a few examples, which the system can combine to a far more complex animation, like the 'jumping orc' on figure 9.20 created using only 2 examples.

10.2.3 Comparison with Blender

A comparison with a traditional animation system, Blender, has been made. Here is a few images and a discussion.

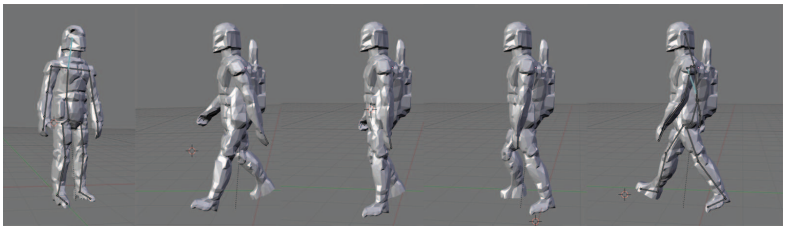


Figure 10.1: Animation using Blender.

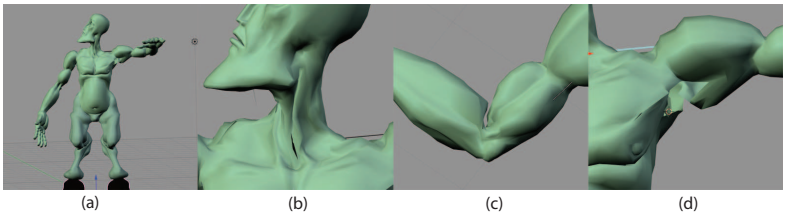


Figure 10.2: Deformations using Blender: **(a)** Raised arm, bending the elbow, turned head. **(b)-(d)** Closeups to illustrate the quality of the skinning method. Notice all the details are preserved.

Rigging and skinning the models took quite some time, maybe an hour, and then a lot of small tweaking followed.

There is no immediate visual advantages in the deformations when using Blender, but there is a lot more manual labor. The animation is a keyframe animation, with the usual 8 walk poses [WALK1]. The result is very much like the result in the presented method's keyframe system.

An expert animator will without a doubt be able to create a better rigging and animation than this, and probably use less time on the creation. Also to create a lifelike skeleton based animation, as much as 140 bones are sometimes used (information from IO Interactive, appendix E). A lot of these bones are used to create secondary deformations, like muscle movement and to preserve the volume. This has not been done here, only the basic bones are created. But using this many bones, also increases the workload, so preparing a single model can take as much as 1 day.

Compared with Blender, the proposed method is definitely offering a smoother setup process, saving a lot of time. The deformations seems, with some tweaking, to be equal in quality, and likewise the keyframe animation created. But note that the Blender animation is a static keyframe animation, without the possibility to change it on the fly, like in the proposed method's example based system using constraints.

More on the Blender workflow can be found in appendix D.

10.3 Performance of method

The goal of the method has not been to make a faster type of animation, as the traditional methods are simple and highly optimized, and are thereby very fast and hard to beat.

As the method stands, it is probably too slow to be used directly in realtime applications like a game, but it can be used in offline tools and non-realtime animations.

The bottle neck of the method, is definitely solving the Laplacian system. This is done using TAUCS, and can not be improved without a change of solver. A new package CHOLMOD has been released which apparently offers a solving time of only a third of TAUCS. This package has not been used in this project due to its late release time. If implemented, the use of this package would probably cut the frame time in half.

Also the method to compute the smooth normals are rather slow, and maybe using the method to estimate the smooth normals, shown in [LSCO04], can improve the performance.

Around 45.000 vertices are the upper limit on the editing system, above that number the per frame time goes above 1 second and the editing experience will be very bad.

For the interactive animation system, the optimization function, finding poses in the pose space, is usually fast enough, no matter which optimization method is used. Using Simple Hill Climbing, the search can be done in less than 0.005 seconds, which means that it will not affect the frame rate considerably.

This again makes the deformation system set the limit for how big models can be worked with. The pose interpolation system is therefor limited to about the

same number of vertices as the editing system, 45.000. This gives about 1 frame per second, slower than this is definitely not good.

As animation has even higher requirements for interactivity than plain posing, the vertex count should be a lot less to ensure smooth and good looking animation. Here around 2.000 vertices must be seen as the maximum number. If the animation result is not needed for realtime application, there is off course no limit to the size of model, as each frame can be captured and then pieced together afterwards.

10.4 Applications of method

As a stand-alone tool this method has several uses:

- Interactive model posing for quick visualization or export.
- Creating poses for keyframe animation, and testing these.
- Animation using constraints and paths.

Exported poses could be used in morphing animation, or as poses for use in Skinning Mesh Animations [JT05], to aid in the production of a traditional skeleton animation.

10.5 Future work

Here a list of ideas for future work is given, in random order.

Auto generation of handles Auto generation of handles, either from template or from analyzing the model.

As a plugin Implement the method as a plugin to commercial programs like 3DStudio Max and Maya.

Motion captured data Connect it to motion captured data. This data should be added directly as constraints to corresponding handles, and could thereby create some good animations, using a for more complicated path structure than an animator would be able to do by hand.

Other deformation system Different Laplacian Editor for deformation, as the method to rotate the details may not be the best. Unfortunately the better methods is also slower. Especially volume preservation would be very nice to include.

New solver Different solver for linear systems, as CHOLMOD seems to offer a big increase in speed. The method does not require a specific solver, so there is no problem in substituting the solver package.

Better GUI GLUI does not offer a nice GUI, but in return it is platform independent and fast to implement.

Tools More tools for editing the mesh, constructing the handles and deforming the handles. Especially a twisting function for the handles is attractive.

Detail Deformation Layer The Detail Deformation Layer is only implemented as a proof of concept, seems like a very promising addition to the method. More tools to alter the layer should be developed, plus a better integration to the rest of the method to reduce the performance loss.

As help for skinning It should be examined if the Laplacian Editing method can somehow be used to simplify the skinning in a traditional skeleton development.

Chapter 11

Conclusion

The goals of the project were stated as:

1. Pose creator: Develop method to deform a model, without the use of a traditional skinned skeleton. These deformations, poses, can be used as keyframes or 'good' example poses in the animation system.

2. Interactive Pose Interpolator: Create a system, that is able to blend created poses, and use this to construct new poses, from a pose space, by setting one or more handle constraints. The system must find weights for pose-blending to create a new pose satisfying the constraints given. The pose space is defined by the examples of 'good' poses, which the user has created.

3. Animation system: For demonstration and testing purposes, create a small world for an animation to work within. The animation can be created with the interpolation system above, or a simpler keyframe system.

By using a handle structure instead of a traditional skeleton and Laplacian Editing to deform the surface, a pose creator was successfully created. It utilizes some good principles from skeleton based animation, but manages to simplify the setup process a lot. Even though the process is simplified and automated,

the method still creates some very good deformations. The poses created can be exported or saved for use in the animation system.

Functionality has been created to enable loading poses or part of a pose. This leads to blending of poses and to enrichment of the pose space, plus it is the foundation of a animation system implemented.

An optimization system was constructed which can find the best possible blend of some example poses, to satisfy some user defined constraints, this makes out the Interactive Pose Interpolator. Several different optimization and objective functions have been implemented and can be used depending on which is a priority; speed or precision.

This system was extended to enable time dependent animation following a path setup by the user, which allows for real interactive animation in a small world, which is very useful for evaluation of the method.

Overall a working alternative to traditional skeleton based animation has been proposed. It is able to run with large meshes in interactive environments, and produces good quality mesh deformations. Performance-wise it can unfortunately not be seen as a replacement of skeleton-based animation yet, but as it has some clear advantages both in form of setup time, and also with the possibility to create animations which adapts to changes instantly. Also it handles both large scale animations like limbs for a walk sequence, and smaller feature changes like facial expressions with the same method.

With the improvements mentioned in section 10.5, Future work, this novel method can be further improved, both in terms of possibilities and speed. So it seems promising, and with a bit more refinement, it can in the near future be a very useful animation system.

List of Tables

List of Figures

Part VI

Appendix

Appendix A

Discrete Laplacian Operator

Here is presented the derivation of the discrete Laplacian Operator, also known as the umbrella operator ([Kob00], [HS04], [Jab05]).

The Laplacian operator is a second order differential operator in the n-dimensional Euclidean space, defined as the divergence of the gradient. It can be expressed as the sum of the second partial derivatives.

$$\Delta f = f_{uu} + f_{vv}$$

A local surface approximation is found by least squares fitting a second order polynomial surface to the one ring neighborhood of a vertex

$$f(u,v) = f + uf_u + vf_v + \frac{u^2}{2}f_{uu} + uvf_{uv} + \frac{v^2}{2}f_{vv}$$

Now a pair of parameters must be assigned to the vertex and its neighbors. The center vertex $\mathbf{p}_0$ is given the coordinates $(0,0)$, so $f(0,0) = f = \mathbf{p}_0$, and the others (u_i, v_i) are given a uniform parametrization (where n equals the number of neighbors):

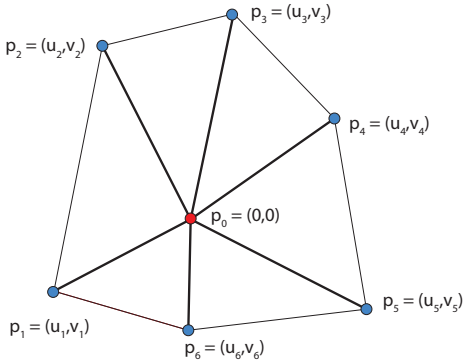


Figure A.1: One ring neighborhood of vertex p_0 and their parameter coordinates.

$$(u_i, v_i) = (cos(2\pi i/n), sin(2\pi i/n))$$

This parametrization places the neighbors uniformly on a unit circle, so it does not take the geometry into account. This might not be accurate, but it simplifies the problem as it only depends on the valency n.

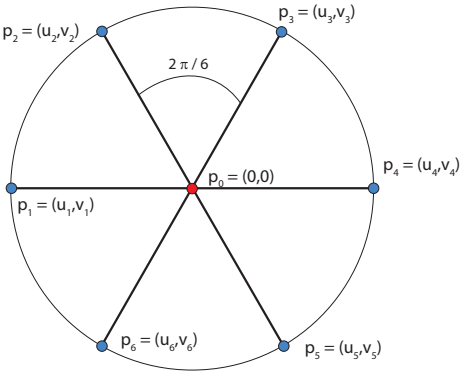


Figure A.2: One ring neighborhood of vertex p_0 and their parameter coordinates, when above parametrization is used.

Using the fact that $f = \mathbf{p}_0$ the following matrix system can be set up:

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ u_i & v_i & \frac{u_i^2}{2} & u_i v_i & \frac{v_i^2}{2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} f_u \\ f_v \\ f_{uu} \\ f_{uv} \\ f_{vv} \end{bmatrix} = \begin{bmatrix} \vdots \\ p_i - p_0 \\ \vdots \end{bmatrix} \tag{A.1}$$

The matrix from the above equation we denote $\mathbf{V}$, and the system $\mathbf{Vf} = \mathbf{p}$. Assuming p_0 has at least 5 neighbors, the system can be solved in a least squares sense:

$$\mathbf{f} = \mathbf{Dp} = ((\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T)\mathbf{p} \tag{A.2}$$

Going back to Δf, it can now be expressed as:

$$\Delta f = \mathbf{f}_3 + \mathbf{f}_5 = (\mathbf{D}_3 + \mathbf{D}_5)\mathbf{p} \tag{A.3}$$

Using the parametrization mentioned before:

$$(u_0, v_0) = (0, 0)$$
$$(u_i, v_i) = (cos(2\pi i/n), sin(2\pi i/n))$$

in equation A.3 the following simple result is obtained:

$$\mathbf{D}_3 + \mathbf{D}_5 = \left[\frac{1}{n}\frac{1}{n}\dots\frac{1}{n}\right] \tag{A.4}$$

now we can get a nice and simple expression for Δf:

$$\Delta f = \frac{1}{n}\sum_{i\in[1:n]}(\mathbf{p_i} - \mathbf{p_0}) \tag{A.5}$$

This is finally the discrete Laplacian Operator, and for vertex $\mathbf{p}_i$ it is:

$$\boldsymbol{\delta}_\mathbf{i} = \Delta f = \frac{1}{n}\sum_{j\in[1:n]}(\mathbf{p_j} - \mathbf{p_i}) \tag{A.6}$$

Appendix B

Energy plots

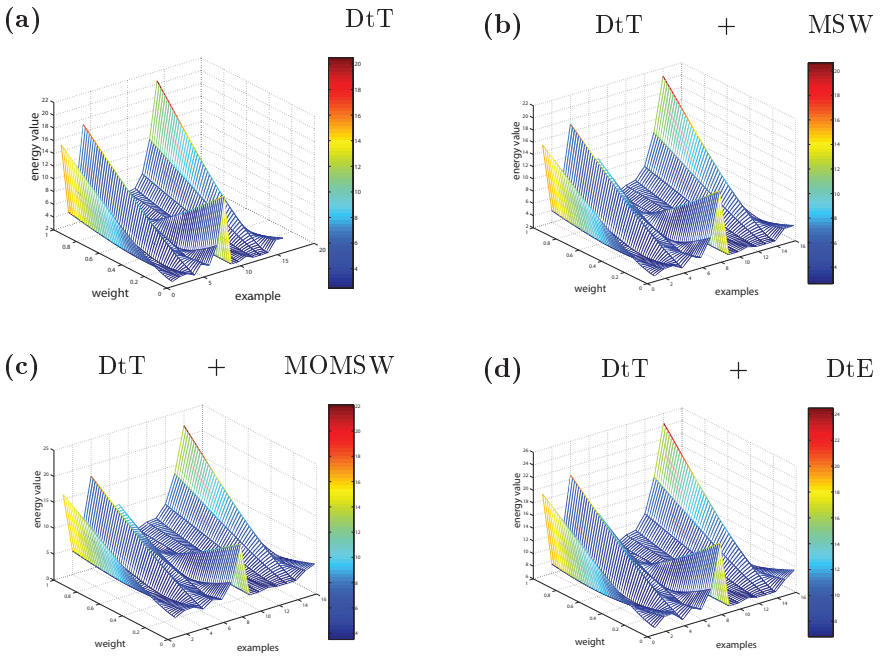


Figure B.1: Using example set B, the examples are tried with different weights, and the results are seen here.

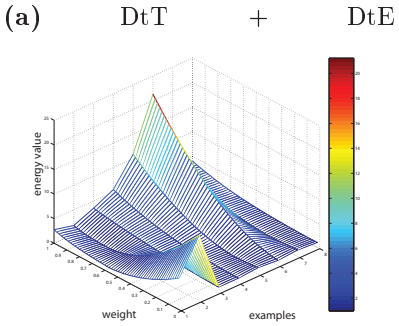
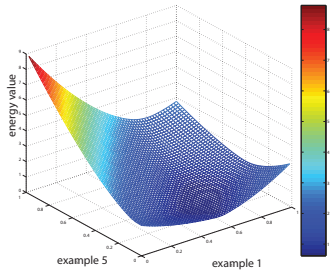


Figure B.2: Using example set A, the examples are tried with different weights, and the results are seen here.

(a) Example 1 and 5



(b) Example 4 and 5

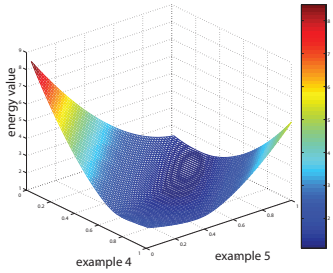


Figure B.3: Using example set A, the examples are tested two at the time, the result is seen here.

Appendix C

3D scanning

During this project, a 3D scanning of a model was made. This is a description of this process.

First step was to coat the model in a white powder substance, to avoid reflections by the laser.

14 scans was taken from different angles, and they where stitched together as nice as possible using a program which came with the scanner (a Minolta).

Unfortunately there were still many holes in the mesh, especially under the arms.

The scanner software tried to fill these holes, but it unfortunately resulted in the arms getting welded onto the body.

One arm was surgically detached from the body, the other was left attached. The feet was also a huge problem, as they where not scanned accurately and not from bellow.

Basically the model was too small for the scanner. A larger model would require more scans, but it would give a lot better result.

To be able to use the scanned model appropriately, Rapidform 2006 were used to correct the errors of the scan; the other arm was detached from the body, and the mesh was decimated to more suitable numbers of vertices around 20.000, 10.000 and 2.000.

Figure C.1: The Minolta scanner.

Figure C.2: Photo of the Boba Fett model.

0

Figure C.3: Resulting scan of the Boba Fett model.

Appendix D

Blender Workflow

1. Import mesh
2. Setup scene
3. Create bones
4. Attach mesh to bone structure
5. Go through all bones, see their vertex groups
6. Correct vertex groups
7. Try to edit the pose. Result is usually bad, as some vertices may not be attached to the proper bone.
8. Correct weights of vertex groups
9. Iterate until satisfied

When satisfied with the rigging and skinning, deformations can begin. Poses are created at keyframes, and then a keyframe animation can be created.

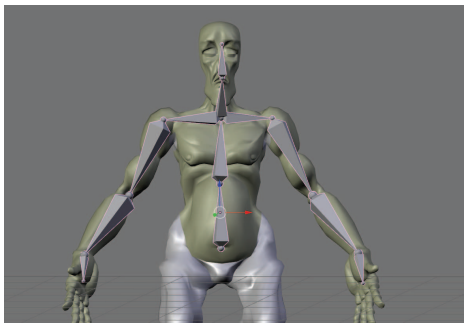


Figure D.1: Skeleton created.

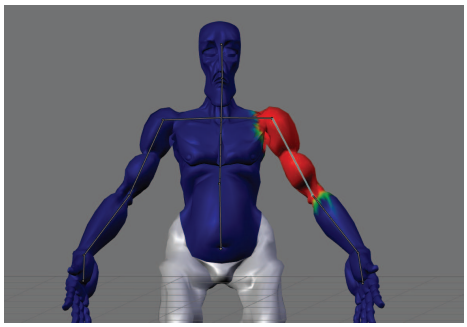


Figure D.2: Weighting vertices (Skinning).

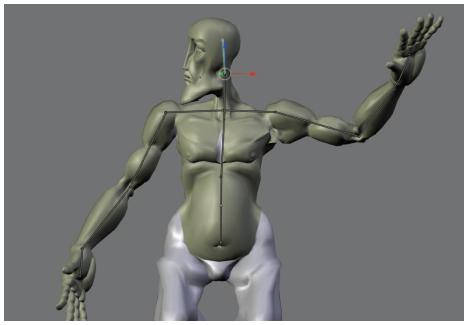


Figure D.3: Deformation.

Appendix E

Meeting with IO Interactive

This is a small resume of my meeting with people from the game company IO Interactive on August 14.

The following people were present: Steffen Toksvig (CTO), Karsten Lund (Lead Animator), Tom Isaksen (Lead Character Artist).

Following views of the method was presented:

The presented method seems much simpler than traditional rigging and skinning, which can take up to a whole day. But the bone structure is a very big part of the character system today, used for various things, not only animating. So it is not possible to drop it.

From an animators point of view, fast and automated is not necessarily good, as a lot of tweaking options is lacking, and they like to have complete control.

They liked the quality of the deformations, but especially one problem exists, which they also knew from skeleton-based systems: When rotating for example the wrist (opposed to normal bending), is causing difficulties as this rotation should be spread over the length of the arm. In their systems this is solved using many extra bones to control the deformations.

Another issue which exists in skeleton based animation is volume preservation, where again several extra bones is used to avoid this. But from looking at the results of the proposed method they were optimistic.

The proposed method is too slow as a general in-game method, but could be interesting as offline tool. Or maybe to help in the skinning process, if a method was found to use Laplacian Editing to create vertex weights.

Appendix F

Animation examples

F.1 Walking Boba Fett

Figure F.1: Poses from walk animation using Simple Hill Climbing on example set A. From top down: 1) DtT only. 2) DtT+SoW. 3) DtT+OMSoW. 4) DtT+DfE.

Figure F.2: Poses from walk animation using DtT+DfE on example set A. From top down: 1) Simple Hill Climbing. 2) Hooke and Jeeves. 3) Simple Hooke. 4) Global exhaustive.

F.2 Jumping orc

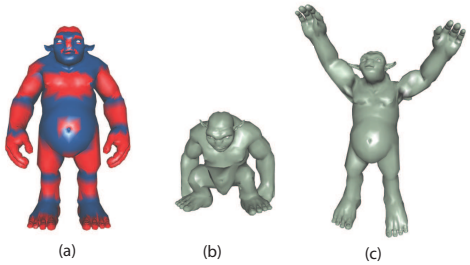


Figure F.3: (a) Orc handle structure. (b) Example 1. (c) Example 2.

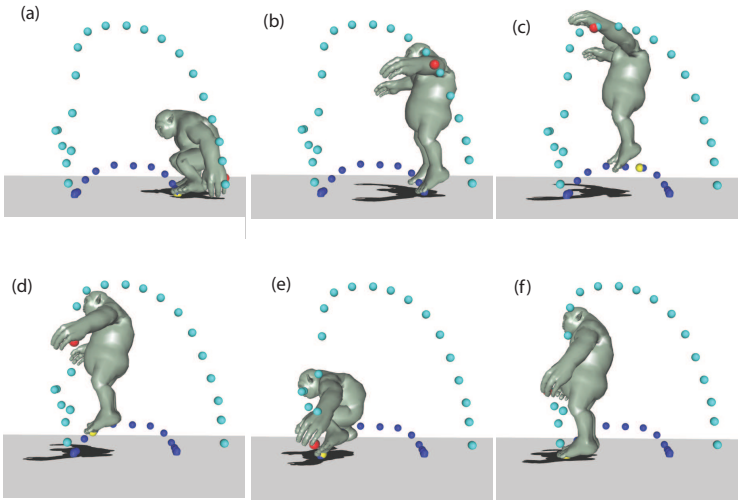


Figure F.4: Using the above 2 examples, a path for th left foot, and a path for the left hand, this jumping animation is created. It is running at around 15 fps.

Appendix G

Users Guide

In this appendix, a very short guide to the program is given.

G.1 Installation

No specific installation file has been created. Just place the RUNTHIS.exe file in same directory as needed DLL files and run.

If you wish to import other models than the default, place all files from the 'PROGRAM' folder on the CD in a folder called 'freemotion' on your C-drive. Now you can drag an OBJ models onto the DRAGNDROP.exe shortcut file.

G.2 General options

The Main Tools panel contains several options like viewing, transparency, visibility, load, save and export functions.

Figure G.1: Main tools panel.

G.3 Creating handle structure

Press New or 'n' to create a new handle next time vertices are selected using paint or box-selection.

After new handle is created, it goes back into 'add' mode, where you can use paint or box-selection to add vertices to the selected handle.

Holding down left shift, remove vertices when selected.

When handle structure is done, click OK.

Figure G.2: Handle tools panel.

Now you can grab each rotation center ball and move it to its appropriate position. The model can be made transparent for easier navigation.

The complete handle structure can be saved for later use.

Click OK.

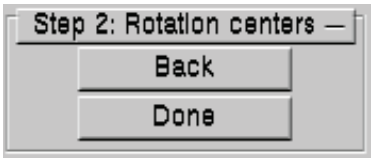


Figure G.3: Rotation Center panel.

G.4 Create an example

Left click on handle to select as Main handle. Holding down shift when left clicking selects a handle as secondary handle.

Pressing 's', saves secondary handles for the main handle.

By clicking and dragging, the selected handles can be rotated. The rotating is performed around the rotation center of the main handle with an axis perpendicular to the screen.

Holding down Control, enables translation of the selected handles.

When a wanted pose is obtained it can be saved to a file.

As many examples as wanted can be created.

Click OK.

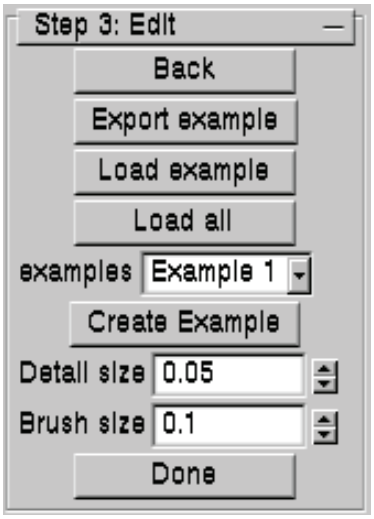


Figure G.4: Create Pose panel.

G.5 Keyframe animation

When in Animation mode, the examples can be run as a keyframe animation by pressing 'k'.

Examples can be left out, by deselecting them in the 'Poses On/Off' panel.

G.6 Interactive Pose Interpolator

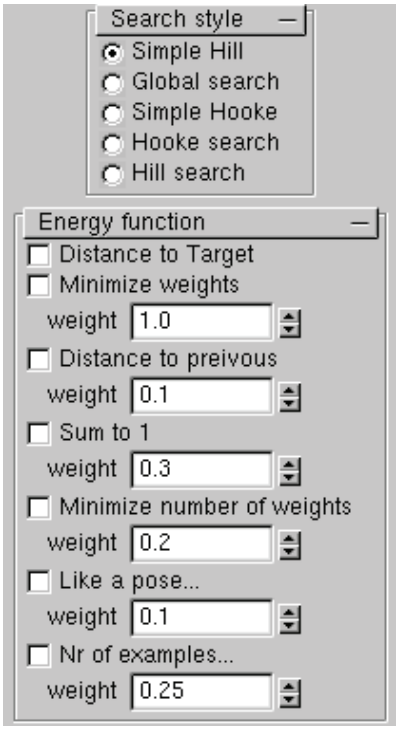


Figure G.5: Optimization and Energy panel panel.

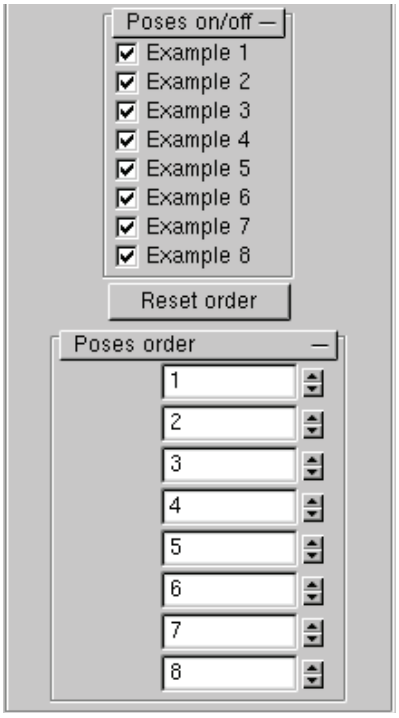


Figure G.6: Optimization and Energy panel panel.

Select a handle by left clicking on it. Press 'z' to constrain this handle.

The constraint can now be moved by clicking and dragging.

The optimization function and objective function can be changed.

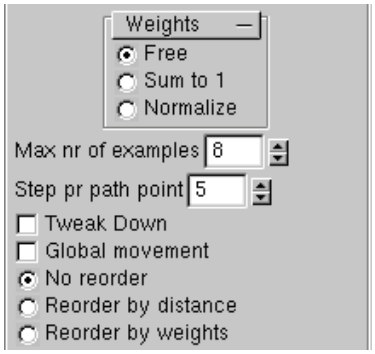


Figure G.7: Different settings.

G.7 Animation

Right click to create path points.

Remember to enable global movement.

Press 'a' to enable auto animation, the model will now follow the path.

The created paths can be saved and loaded.

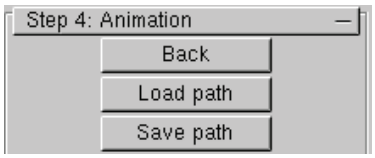


Figure G.8: Load/Save paths.

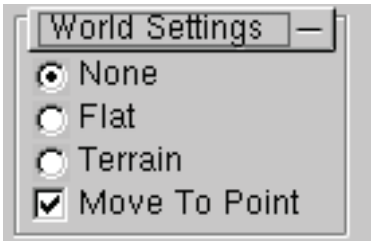


Figure G.9: Different surfaces can be visualized.

Appendix H

Poster for Vision Day 2006

Example-based Animation of Scanned Models

Kristian Evers Hansen - s001678@student.dtu.dk
Informatics and Mathematical Modelling, Technical University of Denmark

Abstract

In this project I present a method for easy and userfriendly creating animations of scanned models. The method makes no use of skeleton or bones in a traditional way, but is using handles painted on the surface of the model to control the movement.

The handles are transformed by the user, and the surface between the handle is deformed accordingly using Laplacian Editing.

Using this technique, the user creates example poses, which can be used in later animation. The method's key advantage is knowing the exact transformations of each handle for a given example.

The system is still work in progress, but seems very promising.

A) Original model with the handle structure defined.

The handles (red areas) are the rigid areas of the model.

The handles can be easily modified to tweak the resulting deformation.

B) Using the right upper arm as the main handle, the arm is lowered.

The lower arm and hand are secondary handles, just following the main.

This hierarchy is not fixed, it is defined by the user at each deformation.

C) Finally the elbow is bent, the left leg is lifted and the knee is bent.

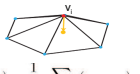
This whole process can be managed in a matter of minutes, and the system can handle tens of thousands of vertices with interactive framerates on a normal PC.

Creating Examples

First step is to create the handle structure, which consists of a number of handles and a rotation point for each of these. The user can freely create and modify this structure, using simple but powerful tools.

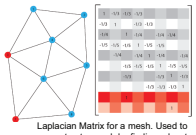
When the handle structure is complete, the user can then manipulate the handles, to get the wanted pose.

Laplacian Editing: The part of the mesh not part of a handle, is deformed using Laplacian Editing. Which is a method for altering a model, while preserving low frequent details, using the *discrete mesh laplacian operator* to reconstruct the mesh.



$$L(v_i) = \frac{1}{d_i} \sum_{j \in N(i)} (v_i - v_j)$$

Laplacian Operator at vertex *v*
N contains neighbors, *d* is the |*N*|



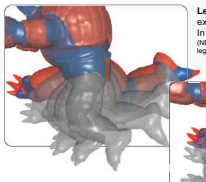
Laplacian Matrix for a mesh. Used to reconstruct a mesh by finding a least squares solution to a **Ax = b** linear system.

Animation

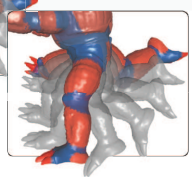
The main idea is that the user drags a part of the model to a new position, and the system must then search for the best combination of examples to fit this position best.

The system searches the example space pairing the examples two at the time. For each pair it finds the optimal interpolation of these two. The system takes advantage of the fact that it knows the exact transformations used to create a pose. This can be utilized to speed up the search.

When the best pose is found, the handles are moved to these positions, and the surrounding surface is reconstructed using Laplacian Editing.



Left : Animation using only two examples (coloured models).
In this case weight1 = 1 - weight2
(NB! The lower leg is bent inward when the leg is forward.)



Right : Same animation, but with an extra example in between. This new example with the leg straightened, gives a different and better looking animation.

Appendix I

CD-Rom content

THESIS Contains this thesis as a PDF.

PROGRAM Contains the program as an executable, the needed DLLs and the used models, which can be dragged onto the program file.

MOVIES A selection of movies, matching the examples presented in this thesis.

SOURCE All source code for the program.

POSTER The poster presented at Visionday 2006.

OTHER Abstract which participated in Dansk Virtual Reality Selskabs competition.

models All 3D models obtained during this project.

Bibliography

Bibliography

[Ale01] M Alexa, *Local Control for Mesh Morphing*
2001, p. 209 -215. In: Shape Modeling and Applications, SMI 2001 International Conference on...

[Ale02] M Alexa, *Linear Combination of Transformations*
2002, p. 380-387. In: ACM Transactions on Graphics 21.

[Ale03] M Alexa, *Differential Coordinates for Local Mesh Morphing and Deformation*
2003, 19 (2-3) , p. 105-114. In: The Visual Computer.

[Ale03b] Alexa, *Laplacian Mesh Editing – Slides.*

[ATLF06] Au, Tai, Liu, Fu, *Dual Laplacian Editing for Meshes*
2006.

[AWC] Alexis, Wyvill, Cani, *Sweepers: Swept User-Defined Tikks for Modelling by Deformation.*
Shape Modeling International - june 2004

[AWCK] Alexis, Cani, Wyvill, King, *Swirling-Sweepers: Constant-Volume Modelling*
2004, p. 10-15. In: Proceedings. 12th Pacific Conference on Computer Graphics and Applications.

[BBK] Botsch, Bommes, Kobbelt, *Efficient Linear System Solvers for Mesh Processing*
2005, p. 62-83. Lecture Notes in Computer Science.

[BBM] Bloom, Blow, Muratori, *Errors and Omissions in Marc Alexa's Linear Combination of Transformations*
2004 unpublished.

[BCK96] Bäck, *Evolutionary Algorithms in Theory and Practice*
1996, Oxford University Press.

[BK04a] Botsch, Kobbelt, *An Intuitive Framework for real-time freeform modelling*
2004, 23 (3) , p. 630-634. In: ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2004.

[BK04b] Botsch, Kobbelt, *A Remeshing Approach to Multiresolution Modeling*
Eurographics Symposium on Geometry Processing 2004, pp. 189-196.

[BK05] Botsch, Kobbelt, *Real-Time Shape Editing using Radial Basis Functions*
Eurographics 2005, pp. 611-621.

[BPGK06] Botsch, Pauly, Gross, Kobbelt, *PriMo: Coupled Prisms for Intuitive Surface Modeling*
Eurographics 2006.

[boost] `www.boost.org/libs/graph/doc/minimum_degree_ordering.html`

[cholmod] `www.cise.ufl.edu/research/sparse/cholmod/`

[DKT96] Deussen, Kobbelt, Tücke, *Using Simulated Annealing to Obtain Good Nobal Approximations of Deformable Bodies.*
1996.

[DMS99] Desbrun, Meyer, Schröder, Barr, *Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow*
1999.

[FT05] Fu, Tai, *Mesh Editing with Affine-Invariant Laplacian Coordinates*
jan-2005 Computer Science Technical Report, http://hdl.handle.net/1783.1/2024.

[GEL] `http://www2.imm.dtu.dk/ jab/GEL/`

[GMHP04] Grochow, Martin, hertzmann, Popovic, *Style-Based Inverse Kinematics*
Proceedings of SIGGRAPH 2004.

[GUP02] Gupta, *Recent Advaces in Direct Methods for Solving Unsymmetric Sparse Systems of Linear Equations*
2002.

[HJ61] R. Hooke and T. A. Jeeves, *Direct Search Solution of Numerical and Statistical Problems*
Journal of the ACM, Vol. 8, April 1961, pp. 212-229

[HS04] Hansen and Spence *Free-form Surface Modelling using Curve Constraints*
2004, Midterm Project at IMM, DTU.

[Jab05] Bærentzen, *Mesh Smoothing and Variational Subdivision*
2005, Lecture note.

[Jak01] Jakobsen, *Advanced Character Physics*
2001.

[JT05] James, Twigg, *Skinning Mesh Animations*
2005, SIGGRAPH 2005.

[KGV83] Kirkpatrick, Gelatt, Vecchi, *Optimization by Simulated Annealing*
Science, New Series, Vol. 220, No. 4598, 1983.

[KJS06] V. Kraevoy, D. Julius, A. Sheffer, *Shuffler: Modeling with Interchangeable Parts*
2006 http://www.cs.ubc.ca/ vlady/

[Kob98] Kobbelt et al, *Interactive Multi-resolution Modeling on Arbitrary Meshes*
1998, p. 105-114. In: Proceedings of the ACM SIGGRAPH Conference on Computer Graphics.

[Kob99] Kobbelt et al, *Multiresolution Hierarchies on Unstructured Triangle Meshes*
1999, 14 (1-3) , p. 5-24. In: Computational Geometry.

[Kob00] Kobbelt, *Discrete Fairing and Variational Subdivision for Freeform Surface Design*
2000, The Visual Computer

[KS04] V. Kraevoy and A. Sheffer, *Shape Preserving Mesh Deformation*
2004 http://www.cs.ubc.ca/ vlady/

[KS05] Krayevoy, Sheffer, *Boneless motion reconstruction*
2005, Technical sketch at SIGGRAPH 2005 http://www.cs.ubc.ca/~vlady/Papers/sketch_0280.pdf

[LAPACK] `www.netlib.org/lapack/`

[LCF00] Lewis,Cordner,Fong, *Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation*
2000, Proceedings of ACM SIGGRAPH 2000.

[LCL06] Lipman, Cohen-Or, Levin, *Volume and Shape Preservation via Moving Frame Manipulation*
2006, under revision to ACM TOG.

[LSCO04] Lipman, Sorkine, Cohen-Or, Levin, *Differential Coordinates for Interactive Mesh Editing*
2004, p. 181-190. In: Proceedings - Shape Modeling International SMI 2004.

[LSLCO05] Lipman, Sorkine, Levin, Cohen-Or, *Linear Rotation-invariant Coordinates for Meshes*
2005, Proceedings of ACM SIGGRAPH 2005.

[MFT04] Moccozet, Dellas,Thalmann, *Animatable Human Body model Reconstruction from 3D Scan Data using Templates*
2004.

[MG03] Mohr, Gleicher, *Building Efficient, Accurate Character Skins from Examples*
2003, SIGGRAPH 2003.

[Mic92] Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*
1992, Springer.

[ML04] Minku, Ludermir, *Evolutionary Strategies and Genetic Algorithms for Dynamic Parameter Optimization of Evolving Fuzzy Neural Networdks.*
2004.

[NSAC] Nealen, Sorkine, Alexa, Cohen-Or, *A Sketch-Based Interface for Detail-Preserving Mesh Editing*
2005, Proceedings of ACM SIGGRAPH 2005.

[Pow] Powell, *Five Lectures on Radial Basis Functions*
2005, Technical report DTU.

[SJN03] Sibiryakov, Ju, Nebel *A New Automated Workflow For 3D Character Creation Based On 3D Scanned Data*
2003

[SK04] A. Sheffer and V. Kraevoy, *Pyramid coordinates for morphing and deformation*
2004 http://www.cs.ubc.ca/ vlady/

[SLCO04] O Sorkine, Y Lipman, D Cohen-Or, M Alexa,C Rössl,H-P Seidel, *Laplacian Surface Editing*
2004, p. 179-188. Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing.

[Sor] Sorkine, *Least-squares Meshes*
2004, p. 191-199. In: Shape Modeling Applications, 2004. Proceedings.

[Sor05] Sorkine, *Laplacian Mesh Processing*
2005, Presented at Eurographics 2005.

[SRC00] Sloan, Rose, Cohen, *Shape and Animation by Example*
2000, Technical Report, Microsoft Research.

[SZGP05] Sumner,Zwicker,Gotsman,Popovic,*Mesh-Based Inverse Kinematics*
SIGGRAPH 2005.

[SYBF06] Shi,Yu,Bell,Feng,*A Fast Multigrid Algorithm for Mesh Deformation*
SIGGRAPH 2006.

[Tau95] Taubin, *A Signal Processing Approach to Fair Surface Design*
1995, SIGGRAPH 1995.

[Tau00] Taubin, *Geometric Signal Processing on Polygonal Meshes*
2000, EURO-GRAPHICS 2000.

[Tau01] Taubin, *Dual Mesh Resampling*
Proc. Conf. Pacific Graphics, p. 94-113, 2001.

[TAUCS] `www.tau.ac.il/ stoledo/taucs/`

[Trss01] Trosset, *What is Simulated Annealing*
Optimization and Engineering 2, 201-213, 2001.

[WALK1] `www.anticz.com/Walks.htm` A Simple Method for creating Walk Cycles by Mike Brown.

[WALK2] `http://www.mcmxi.com/ jpr/teaching/GPH213/walkcycle/walk.html`
Walk Cycles - Historical Background - Eadweard Muybridge's Motion Studies.

[YM93] Yuret, Maza, *Dynamic Hill Climbing: overcoming the limitations of optimization techniques*
1993.

[YZX04] Yu, Zhou et al, *Mesh editing with Poisson-Based Gradient Field Manipulation*
2004, 23 (3) , p. 644-651. In: ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2004.

[Zal97] Zalzala, *Genetic algorithms in engineering systems*
Control Engineering Series 55, 1997.